Module 4: Construction

CPSC 310

Reid Holmes & Nick Bradley

C0

- 285/539 repos have been checked out (53% have started).
- AutoTest latency was < 5 minutes yesterday until 1800.
  - Grew to 75 mins in the late evening.
  - 1200+ jobs on the queue.
- 55+ PROFICIENT solutions.
- 3 AutoTest runs in 24h, instead of 2 for rest of project.
- Deadline 1800 this Friday.

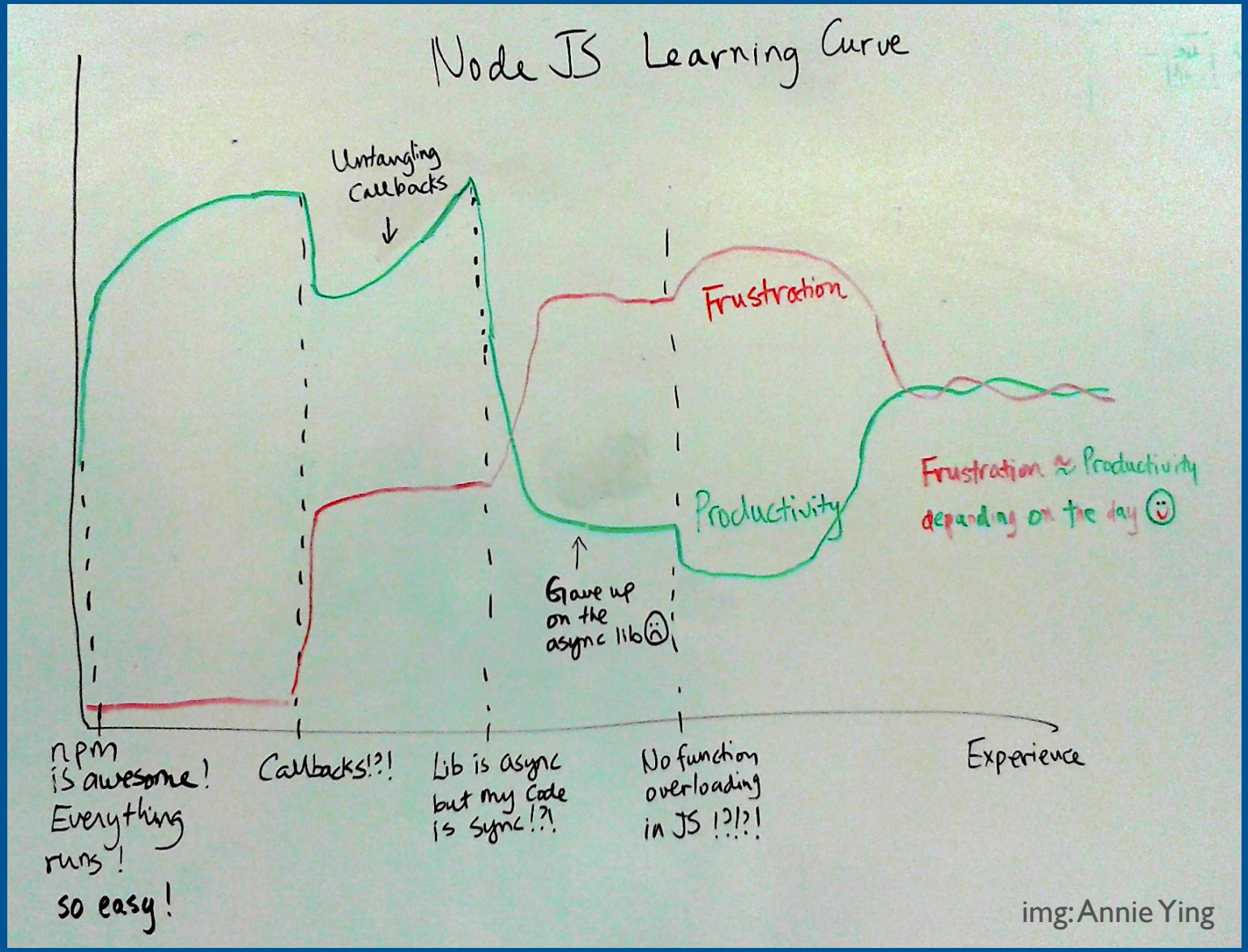
Project journal

- Details about the first project journal have been posted online.
- Each deliverable will have a journal component.
- This is intended for meta-reflection.
  - Mostly individual, but there will be some team components.
- Two components:
  - Reflect on some specific aspects of past checkpoint.
  - Perform some specific planning tasks for next checkpoint.
- Due exactly 7 days after each project checkpoint.
- Oral discussion with TA the lab after journal submission.

Examinable skills:

- Explain why different languages exist.
- Differentiate syntax and semantics.
- Describe how static and dynamic typing are different.
- Understand how static analysis works.
- Explain how linting is used to improve the quality of most software systems.
- Describe why async functions exist.
- Describe how async functions execute in Node/browser.
- Be able to write and understand straightforward functions using async/await, promises, and callbacks.

Async


Node JS Learning Curve
Untangling Callbacks
Frustration
Productivity
Frustration ≈ Productivity depending on the day
Gave up on the async lib
npm is awesome! Everything runs! so easy!
Callbacks!?!
Lib is async but my code is sync!?!
No function overloading in JS !?!?!
Experience
img: Annie Ying

Multithreaded programming

Concurrency

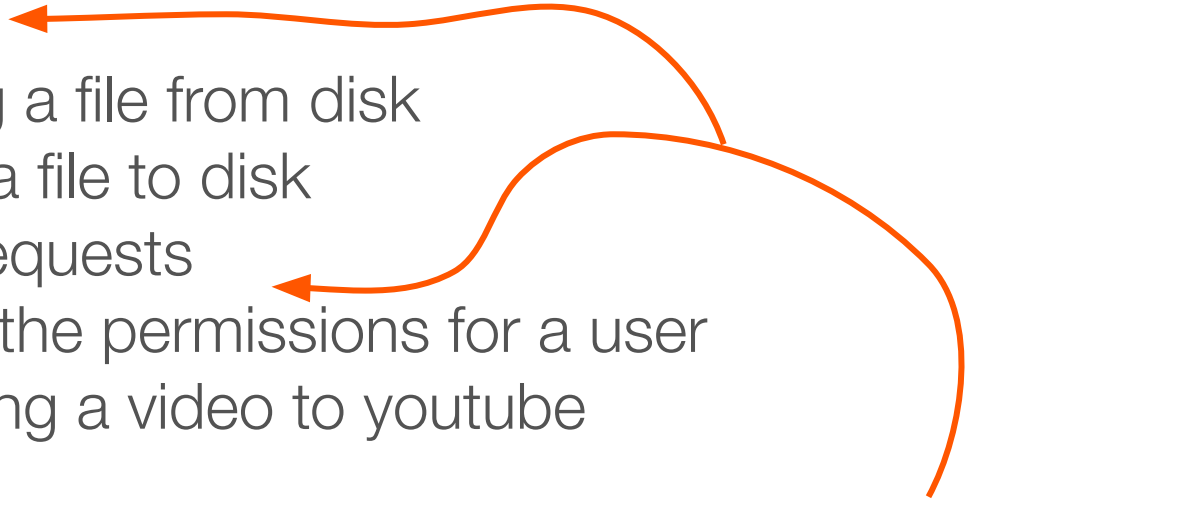
- What is concurrency?
  - Executing multiple program parts at the same time.
  - Actual ordering is indeterminate.
- Why is it hard?
  - Shared state! W before R diff than R before W
  - Deadlock / live lock / resource starvation
- Why would we ever want to run code concurrently?
  - Computation gains driven by cores; avoids blocking

Why is blocking a problem?

http://rbyers.github.io/scroll-latency.html

Async is for long running processes

- `setTimeout()`
- File I/O
  - Reading a file from disk
  - Writing a file to disk
- Network Requests
  - Getting the permissions for a user
  - Uploading a video to youtube



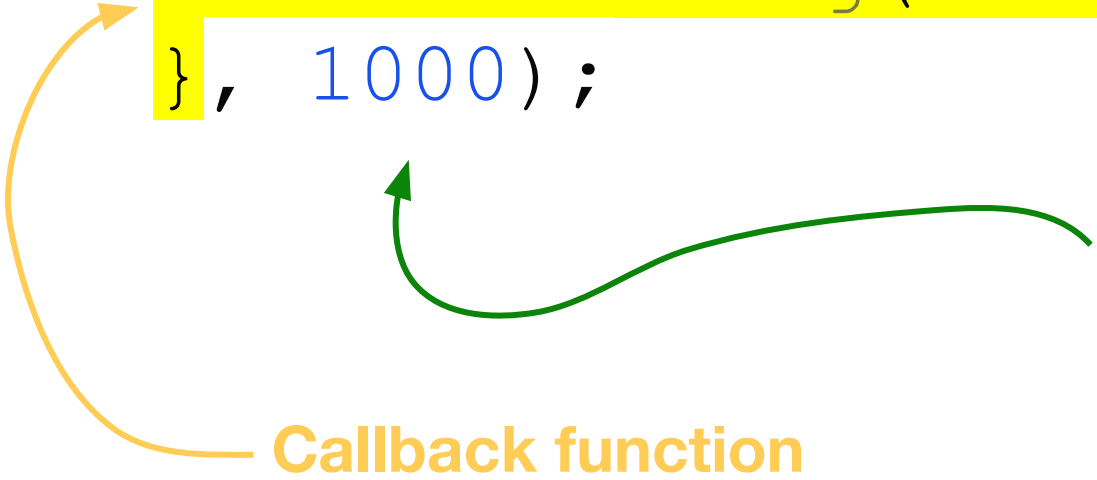
Used in the project!

setTimeout(...)

```
setTimeout(function() {
    console.log("Hello");
}, 1000);
```

setTimeout(...)

```
setTimeout(function() {
    console.log("Hello");
}, 1000);
```



```
function start() { doIt(); }

function doIt() {
    console.log('start');
    setTimeout(function() {
        console.log('done');
    }, 2000);
    console.log('end');
}

start();
```

What will the output be?

Why would we do this?

```
function makeItSo() {
   …
   setTimeout(function() {
      // do the long thing
   }, 0);
   …
}
```

Anatomy of a callback

```
const onComplete = function() {
  // function assigned to a var!
};

setTimeout(onComplete, 100);
```

Error-first callback idiom

```
foo.action(msg, function(err, data){
    if (err) {
        // handle
        return;
    } else {
        // worked, use data
    }
});
```

Non-trivial example

Read first file in a directory.

Non-trivial example

Read first file in a directory.

```
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

Non-trivial example

Read first file in a directory.

```
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    //Process data
}
```

Remember!
File I/O is Asynchonous

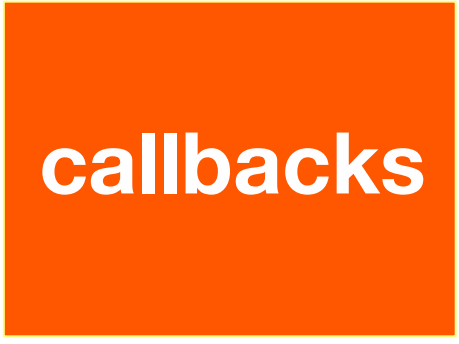
Non-trivial example

Read first file in a directory.

```
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

We need to *handle* these methods

We can handle async… I *promise*



Pre-2015(ish)

2009-2015

2017

We can handle async… I *promise*

callbacks

Pre-2015(ish)

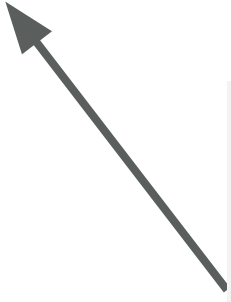
```
setTimeout(function() {
    console.log("Hello");
}, 1000);
```

Callback function

```
fs.readdir(source, function(err, files) {
```

callbacks

Read first file in a directory.

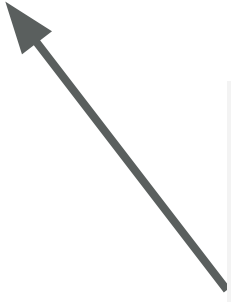


```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

```
fs.readdir(source, function(err, files) {
 if (err) {
   // handle
 } else {
```

callbacks

See if it worked

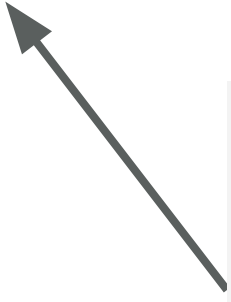


```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

```
fs.readdir(source, function(err, files) {
 if (err) {
   // handle
 } else {
   fs.stat(files[0], function(err, stat) {
     if (err) {
       // handle
     } else {
```

callbacks

Check if it's a file

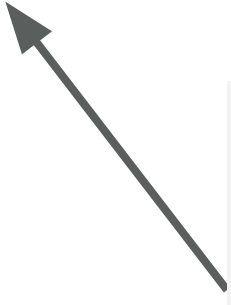


```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

```
fs.readdir(source, function(err, files) {
 if (err) {
   // handle
 } else {
   fs.stat(files[0], function(err, stat) {
     if (err) {
       // handle
     } else {
       if (stat.isFile()) {
         fs.readFile(files[0], function(err, data) {
           if (err) {
             // handle
           } else {
             // process data
           }
```

callbacks

Read the file



```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

```
fs.readdir(source, function(err, files) {
 if (err) {
   // handle
 } else {
   fs.stat(files[0], function(err, stat) {
     if (err) {
       // handle
     } else {
       if (stat.isFile()) {
         fs.readFile(files[0], function(err, data) {
           if (err) {
             // handle
           } else {
             // process data
           }
         });
       }
     }
   });
 }
});
```

callbacks

Add many braces

```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

```
fs.readdir(source, function(err, files) {
 if (err) {
   // handle
 } else {
   fs.stat(files[0], function(err, stat) {
     if (err) {
       // handle
     } else {
       if (stat.isFile()) {
         fs.readFile(files[0], function(err, data) {
           if (err) {
             // handle
           } else {
             // process data
           }
         });
       }
     }
   });
 }
});
```

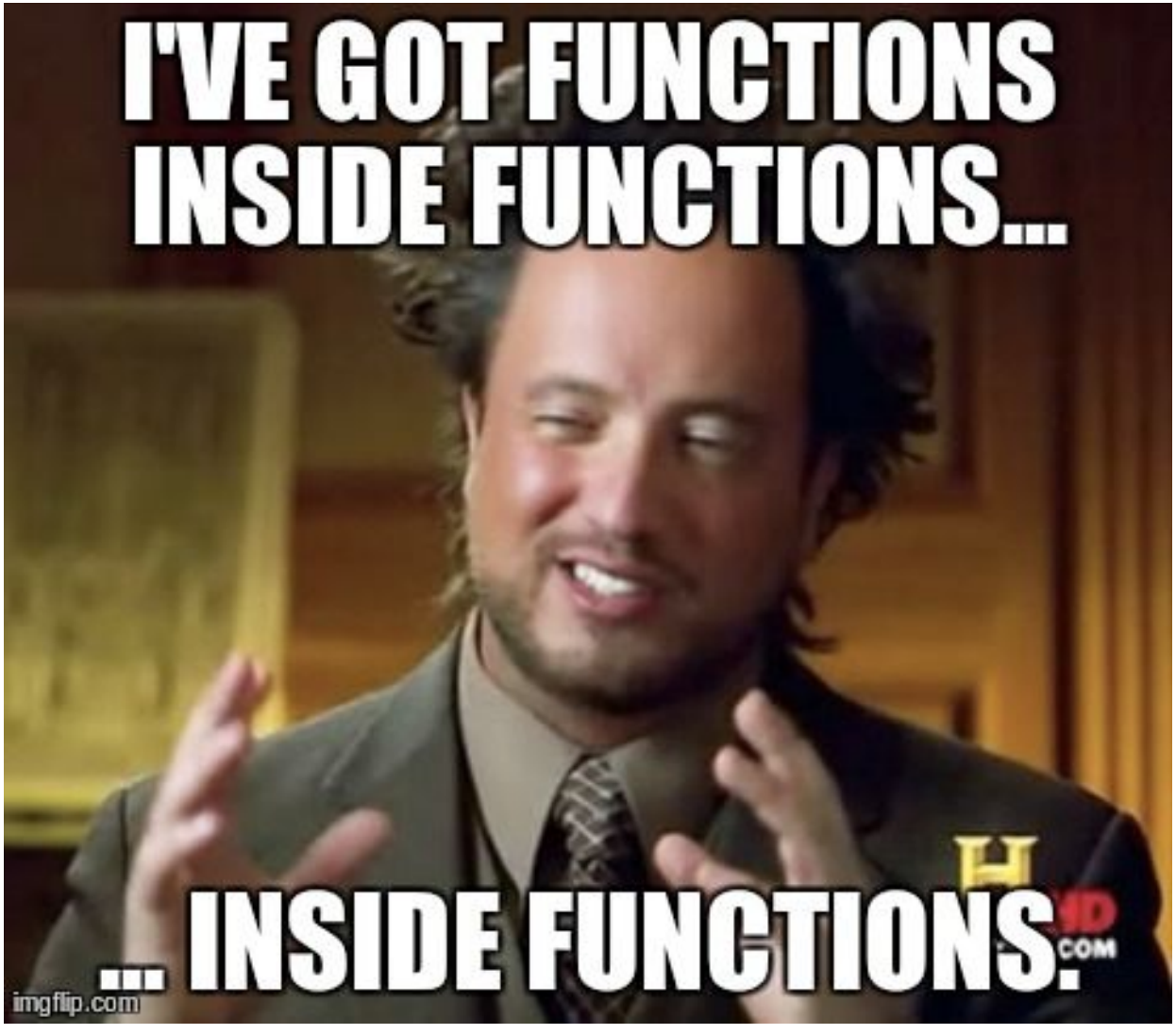
callbacks

Add many braces

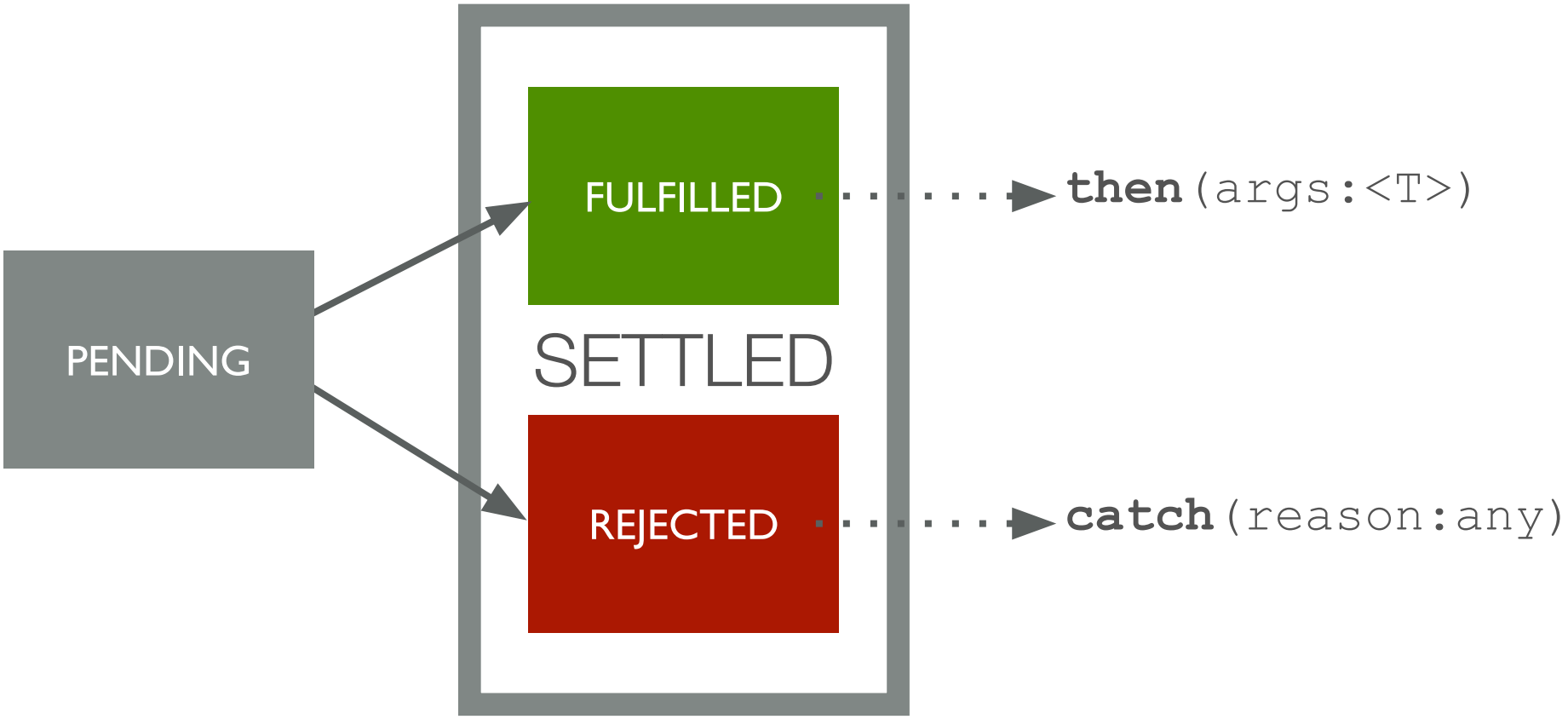
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

Classy Team Provisioning

- Provisioning the projects:
  - 1) Get the Classy data `[getCourse(name, cb)]`
  - 2) In a huge loop:
    - a) Create a repo `[createRepo(name, cb)]`
    - b) Import the base repo `[importRepo(repo, import, cb)]`
    - c) Create the team `[createTeam(name, cb)]`
    - d) For each team member:
      - Add member to the team `[addMember(team, name, cb)]`
    - e) Put the team on the repo `[addTeam(repo, team, cb)]`
    - f) Find the staff team `[findTeam(name, cb)]`
    - g) Put the staff team on the repo `[addTeam(repo, team, cb)]`


I'VE GOT FUNCTIONS
INSIDE FUNCTIONS...
... INSIDE FUNCTIONS.
imgflip.com

Promise execution states



Promises (simplified)

```
interface Promise<T> {
    then(result: <T>): Promise<T>;
    catch(reason: any): Promise<T>;
}
```

Comparing to callbacks

callbacks

```
fs.readdir(source, function(err, files)
{
 if (err) {
   // handle error
 } else {
   // handle success
 }
}
```

promises

```
fs.readdir(source)
```

Comparing to callbacks

callbacks

```
fs.readdir(source, function(err, files)
{
 if (err) {
   // handle error
 } else {
   // handle success
 }
}
```

promises

```
fs.readdir(source).then((files) => {
 // handle success
});
```

Comparing to callbacks

callbacks

```
fs.readdir(source, function(err, files)
{
 if (err) {
   // handle error
 } else {
   // handle success
 }
}
```

promises

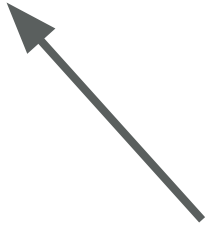
```
fs.readdir(source).then((files) => {
 // handle success
}).catch((err) => {
 // handle error
});
```

Revisiting reading file

```
let file;
fs.readdir(source)
```

promises

Read first file in a directory.



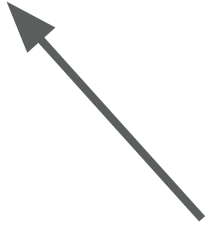
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

Revisiting reading file

promises

Check if it's a file

```
let file;
fs.readdir(source).then(function(files) {
    file = files[0];
    return fs.stat(file);
})
```



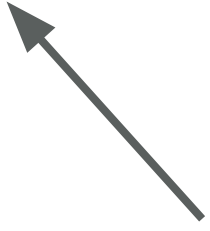
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

Revisiting reading file

promises

Read the file

```
let file;
fs.readdir(source).then(function(files) {
    file = files[0];
    return fs.stat(file);
}).then(function(fileStat) {
    if (fileStat.isFile())
        return fs.readFile(file);
})
```



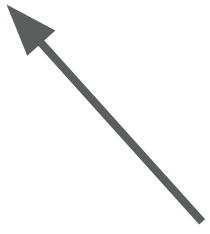
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

Revisiting reading file

promises

Process data

```
let file;
fs.readdir(source).then(function(files) {
    file = files[0];
    return fs.stat(file);
}).then(function(fileStat) {
    if (fileStat.isFile())
        return fs.readFile(file);
}).then(function(fileData) {
    // process data
})
```



```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

Revisiting reading file

promises

Handle errors

```
let file;
fs.readdir(source).then(function(files) {
    file = files[0];
    return fs.stat(file);
}).then(function(fileStat) {
    if (fileStat.isFile())
        return fs.readFile(file);
}).then(function(fileData) {
    // process data
}).catch(function(err) {
    // handle errors
});
```

```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

Revisiting reading file

promises

Handle errors

```
let file;
fs.readdir(source).then(function(files) {
    file = files[0];
    return fs.stat(file);
}).then(function(fileStat) {
    if (fileStat.isFile())
        return fs.readFile(file);
}).then(function(fileData) {
    // process data
}).catch(function(err) {
    // handle errors
});
```

```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

async & await

- Syntactic sugar, TS compiles these to promises:

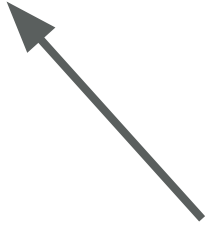
```
async // declare that a result
will eventually be returned

await // tell the code to
suspend execution until result
exists
```

Revisiting reading file

async/await

Read first file in a directory

```
const files = await fs.readdir(source);
```



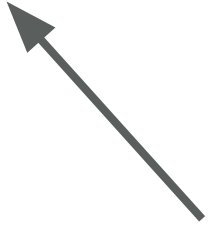
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

Revisiting reading file

async/await

Read first file in a directory

```
const files = await fs.readdir(source);
const file = files[0];
```



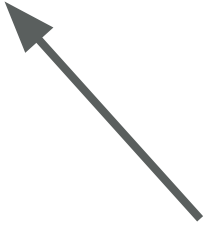
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

Revisiting reading file

async/await

Check if it's a file

```
const files = await fs.readdir(source);
const file = files[0];
const fileStat = await fs.stat(file);
```



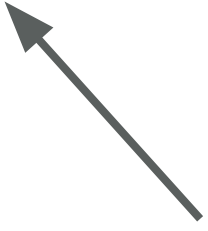
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

Revisiting reading file

async/await

Check if it's a file

```
const files = await fs.readdir(source);
const file = files[0];
const fileStat = await fs.stat(file);

if (fileStat.isFile()){
```



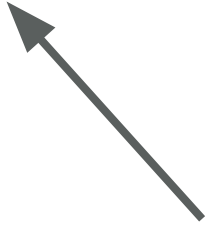
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

Revisiting reading file

async/await

Process Data

```
const files = await fs.readdir(source);
const file = files[0];
const fileStat = await fs.stat(file);

if (fileStat.isFile()){
    const fileData = await fs.readFile(file);
    // process data
}
```



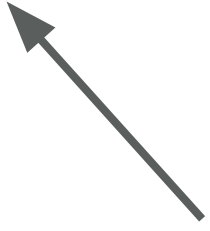
```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
   const data = fs.readFile(files[0]);
   // process data
}
```

Revisiting reading file

async/await

Handle Errors

```
try {
    const files = await fs.readdir(source);
    const file = files[0];
    const fileStat = await fs.stat(file);

    if (fileStat.isFile()){
        const fileData = await fs.readFile(file);
        // process data
    }
} catch(err) {
    // handle errors
}
```



```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

Revisiting reading file

async/await

Handle Errors

```
try {
    const files = await fs.readdir(source);
    const file = files[0];
    const fileStat = await fs.stat(file);

    if (fileStat.isFile()){
        const fileData = await fs.readFile(file);
        // process data
    }
} catch(err) {
    // handle errors
}
```

```
// if this were synchronous
const files = fs.readdir(source);
if (fs.stat(files[0]).isFile()) {
    const data = fs.readFile(files[0]);
    // process data
}
```

What about `async`?

The function returns a promise

```
async function readDirectory() {
    try {
        let file;
        const files = await fs.readdir(source);
        const file = files[0];
        const fileStat = await fs.stat(file);

        …
}
```

```
fs.readdir(source).then(function(files) {
    file = files[0];
    return fs.stat(file);
}).then(function(fileStat) {
    if (fileStat.isFile())
        return fs.readFile(file);
}).then(function(fileData) {
    // process data
}).catch(function(err) {
    // handle errors
});
```

```
fs.readdir(source, function (err, files) {
    if (err) {
        // handle errors
    } else {
        fs.stat(files[0], function(err, stat) {
            if (err) {
                // handle errors
            } else {
                if (stat.isFile()) {
                    fs.readFile(files[0],
                    function(err, data) {
                        if (err) {
                            // handle errors
                        } else {
                            // process data
                        }
})}}})}});
```

```
try {
    const files = await fs.readdir(source);
    const fileStat = await stat(files[0]);
    if (fileStat.isFile())
        const fileData = await fs.readFile(file);
        // process data
    }
} catch(err) {
    // handle errors
}
```

Promises (simplified)

```
// all must pass, else catch
// best choice if complete success is required
Promise.all(promiseList: <T>[]): Promise<T>

// always then, but with status for each promise
// best choice for large queues where retries are ok
Promise.allSettled(promiseList: <T>[]): Promise<T>

// first complete
// rarely used, but sometimes handy
Promise.race(promiseList: <T>[]): Promise<T>
```

Await/Promises and performance/errors

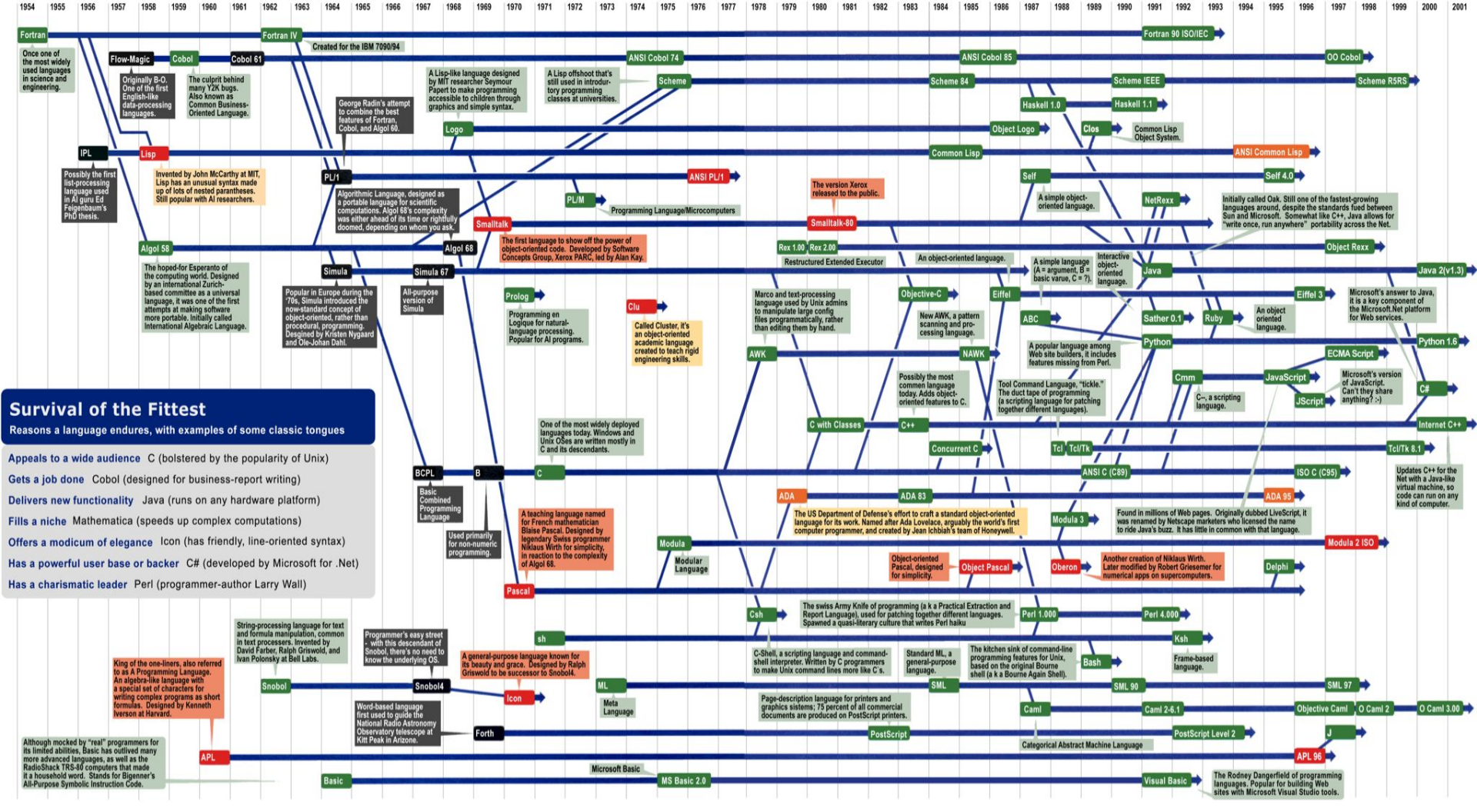
- Promises:
  - Promisified code does not block main thread.
  - Rejected promises will fail silently if not caught.
  - Playground Example
- Async/await:
  - Async code is executed sequentially but does not block main thread.
  - Async errors will fail silently if not caught.
  - Async errors need to be caught in async functions
  - Playground Example

Languages

What does a programming language do?

The process of **transforming a mental plan** of desired actions for a computer **into a representation that can be understood by the computer**.

— Jean-Michel Hoc and Anh Nguyen-Xuan



http://www.digibarn.com/collections/posters/tongues/ComputerLanguagesChart.png

What does this program do?

```
Multp:  .equ    32
Multd:  .equ    64
        .org    4096
        lar     r30, Done
        lar     r31, Loop
        la      r1, Multd
        la      r2, Multp
        andi    r3, r3, #0
        andi    r4, r4, #0
        addi    r5, r3, #1
        neg     r3, r5
Loop:   add     r4, r4, r2
        add     r1, r1, r3
        brzr    r30,r1
        br      r31
Done:   st      r4, Result
        stop
        .org    8192
Result: .dw     1
```

What does this program do?

```
Multp:  .equ    32              ; multiplicand
Multd:  .equ    64              ; multiplier
        .org    4096            ; 0x1000
        lar     r30, Done       ; Load address of Done for branch
        lar     r31, Loop       ; Load address of Loop for branch
        la      r1, Multd       ; Load multiplier
        la      r2, Multp       ; Load multiplicand
        andi    r3, r3, #0      ; clear r3
        andi    r4, r4, #0      ; clear r4
        addi    r5, r3, #1      ; place 1 in r5
        neg     r3, r5          ; place -1 in r3
Loop:   add     r4, r4, r2      ; add multiplicand to running sum
        add     r1, r1, r3      ; start loop, decrement multiplier
        brzr    r30,r1          ; jump to Done if multiplier = 0
        br      r31             ; jump back to Loop
Done:   st      r4, Result      ; store result
        stop
        .org    8192            ; 0x2000
Result: .dw     1               ; storage for result
```

Low-level languages are hard!

```
mult.asm        Wed Feb 02 15:42:09 2000        1

; Impliments a simple 32 bit integer multiply by successive addition
; R. H. Klenke, Sun Jan 31 10:45:14 EST 1999
;

Multp:  .equ    32              ; multiplicand
Multd:  .equ    64              ; multiplier
        .org    4096            ; 0x1000
        lar     r30, Done       ; Load address of Done for branch
        lar     r31, Loop       ; Load address of Loop for branch
        la      r1, Multd       ; Load multiplier
        la      r2, Multp       ; Load multiplicand
        andi    r3, r3, #0      ; clear r3
        andi    r4, r4, #0      ; clear r4
        addi    r5, r3, #1      ; place 1 in r5
        neg     r3, r5          ; place -1 in r3
Loop:   add     r4, r4, r2      ; add multiplicand to running sum
        add     r1, r1, r3      ; start loop, decrement multiplier
        brzr    r30,r1          ; jump to Done if multiplier = 0
        br      r31             ; jump back to Loop
Done:   st      r4, Result      ; store result
        stop
        .org    8192            ; 0x2000
Result: .dw     1               ; storage for result
```

Languages matter

```
mult.asm        Wed Feb 02 15:42:09 2000        1

; Impliments a simple 32 bit integer multiply by successive addition
; R. H. Klenke, Sun Jan 31 10:45:14 EST 1999
;

Multp:  .equ    32              ; multiplicand
Multd:  .equ    64              ; multiplier
        .org    4096            ; 0x1000
        lar     r30, Done       ; Load address of Done for branch
        lar     r31, Loop       ; Load address of Loop for branch
        la      r1, Multd       ; Load multiplier
        la      r2, Multp       ; Load multiplicand
        andi    r3, r3, #0      ; clear r3
        andi    r4, r4, #0      ; clear r4
        addi    r5, r3, #1      ; place 1 in r5
        neg     r3, r5          ; place -1 in r3
Loop:   add     r4, r4, r2      ; add multiplicand to running sum
        add     r1, r1, r3      ; start loop, decrement multiplier
        brzr    r30,r1          ; jump to Done if multiplier = 0
        br      r31             ; jump back to Loop
Done:   st      r4, Result      ; store result
        stop
        .org    8192            ; 0x2000
Result: .dw     1               ; storage for result
```

```
z: number = x * y;
```

Static vs dynamic types

Variables have types in statically typed languages.

Java:

- int age;

- Compiler enforces consistency:
  - age = 1; // ok
  - age = false; // !ok

Static vs dynamic types

Variables have types in statically typed languages.

TypeScript:

- let age: number;

- Compiler enforces consistency:
  - age = 1; // ok
  - age = false; // !ok

Static vs dynamic types

Variables have types in statically typed languages.

TypeScript:

- let age: number;
- Compiler enforces consistency:
  - age = 1; // ok
  - age = false; // !ok

Values have types in dynamically typed languages.

- JavaScript:
  - var age = 1;
  - age = 100; // number
  - age = false; // boolean
  - age = “CPSC”; // string

Static vs dynamic types

Variables have types in statically typed languages.

TypeScript is *optionally* statically typed.

TypeScript:

- let age: number;
- Compiler enforces consistency:
  - age = 1; // ok
  - age = false; // !ok

- TypeScript type Inference:
  - let age = 1; // number
  - age = 100; // ok
  - age = false; // !ok
  - age = “CPSC”; // !ok

Binding context

`function() {} vs. () => {}`

- JavaScript syntax has evolved over many years.
  - E.g., callbacks -> promises -> async/await
- Variable scoping issues led to ‘fat arrow’ notation:

```
class DataProcessor {
    private data = 0;

    public compute() {
        console.log(this.data);
        setTimeout(function() {
            this.data++;
            console.log(this.data);
        }, 0);
        console.log(this.data);
    }
}
```

```
class DataProcessor {
    private data = 0;

    public compute() {
        console.log(this.data);
        setTimeout( () => {
            this.data++;
            console.log(this.data);
        }, 0);
        console.log(this.data);
    }
}
```

```
const proc = new DataProcessor();
proc.compute()
```

Value comparison (== vs ===)

```
if (x === y) {
    console.log("x and y are strictly equal");
}
if (x == y) {
    console.log("x and y are loosely equal");
}

undefined == null; // true
undefined === null; // false

1 == '1'; // true
1 === '1'; // false
```

Static Analysis

Syntax & semantics

- Syntax defines the set of valid tokens that are allowed.
  - ‘`if (token === token)`’ is syntactically valid.
- Semantics define what the tokens actually mean.
  - ‘`if (token === token)`’ is semantically meaningless.
- Compilers check for valid syntax (e.g., `javac, tsc`).
- Semantics will mostly be evaluated at runtime.
  - e.g. by using or testing the system.
  - Linters can detect some semantic issues before runtime.

Static analysis: Formatting & linting

- Static analyses are any programmatic analysis of a program that only inputs the program text itself and does not need to execute the program.
- Dynamic analyses need to run the program, so also require inputs (often via test cases or fuzzing).
- Static analyses are performed in compilation, but also during:
  - Formatting: enforcing consistent code *syntax*
    - Tabs vs spaces, line length, whitespace, etc
    - Aids collaboration and readability
  - Linting: enforcing good *semantics* of code
    - No unused vars, no globals, no misused promises
    - Prevents bugs

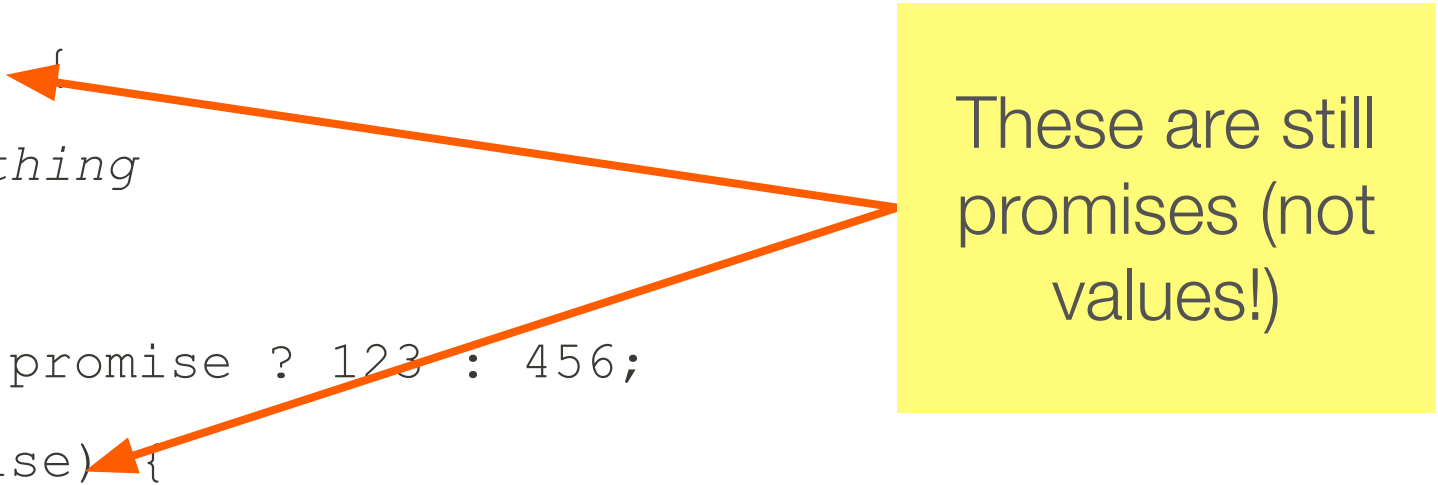
Linting example

```
const promise = Promise.resolve('value');

if (promise) {
  // do something
}

const val = promise ? 123 : 456;

while (promise) {
  // do something
}
```

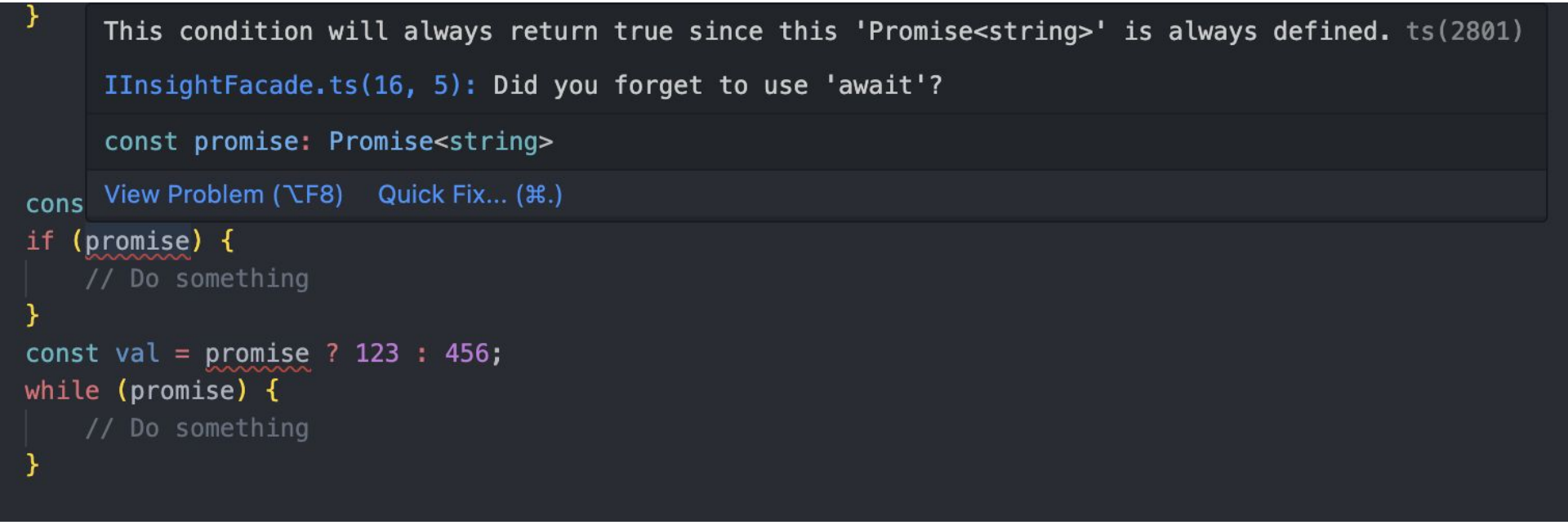
What is wrong with this code?

Linting example

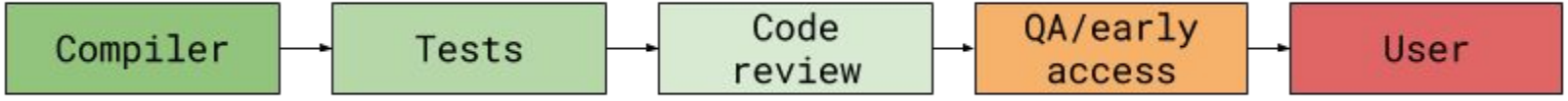
```
const promise = Promise.resolve('value');

if (promise) {
  // do something
}

const val = promise ? 123 : 456;

while (promise) {
  // do something
}
```



We can catch this with lint!



Move bugs to the left



https://samwho.dev/blog/move-your-bugs-to-the-left/

Linters help developers:

- Avoid some semantic errors
- Ensure code consistency
  - Don’t need to think about representation
  - E.g., naming conventions: camelCase vs PascalCase vs snake_case
- Improve code comprehension
  - Consistent code is easier for other developers (including future you) to understand
  - E.g., `const` vs `let`; function return types
- Improve design
- Ease evolution
- Reduce bloat

Linting: Error avoidance

It can be easy to write syntactically valid code that introduces subtle semantic errors.

Example: avoid using await inside a for loop.

```
const files = await fs.readdir(source);
for (const file of files) {
    const content = await fs.readFile(file);
}
```

This code seems to run… a bit slow?

Linting: Code consistency

Conventions make it simpler for developers write code with less mental effort.

Example: use a single naming convention.

```
const userName;
const Password;
const display_name;
```

What do I name the next variable to hold an email address?

Linting: Code comprehension

Enforce code that is easy for other developers (including future you) to understand.

```
// specify function return types
async function getContent(name: string){
    const buffer = await fs.readFile(name);
    return buffer.toString("base64");
}

// prefer const.
let PI = 3.14;
```

What do I get if I call `getContent(..)`?

Should the value for PI ever be reassigned?

Linting: Code evolution

Decisions that you make today can lead to unexpected errors in the future when more code is added.

Example: always use braces for single-line if statements.

```
if (user.isAdmin)
  grantFullAccess(user);
```

What happens if I add a console.log(..) statement above grantFullAccess(..)?

Linting: Security

- Linters can also check for some security vulnerabilities.

```
// @no-eval
public myAction(action: string) { eval(action); }
myAction(“console.log(‘hi 310!’)”);       //
```

Linting: Security

- Linters can also check for some security vulnerabilities.

```
// @no-eval
public myAction(action: string) { eval(action); }
myAction(“console.log(‘hi 310!’)”);       //
myAction(“await fs.remove(‘/home/’” +
   “{ recursive: true, force: true });”);
```

Linting: Security

- Linters can also check for some security vulnerabilities.

```
// @no-eval
public myAction(action: string) { eval(action); }
myAction(“console.log(‘hi 310!’)”);        //
myAction(“await fs.remove(‘/home/’” +
   “{ recursive: true, force: true });”);

// @security/detect-object-injection
public handler(key:string, value:string) {
   this.authenticatedUser[key] = value;
}
handler(“isAuthenticated”,“true”);
```

Refactoring

Current status

- Due tomorrow @ 1800.
- J1: due in 8 days @ 1800.
- C2: Released before the first lab on Monday.
  - Looking at this will be crucial for completing your J1 user stories.

Current status

- Due tomorrow @ 1800.
- J1: due in 8 days @ 1800.
- C2: Released before the first lab on Monday.
  - Looking at this will be crucial for completing your J1 user stories.
- LA buckets: First release by tomorrow.
  - Not all LA's have had enough coverage to be released.
  - Each update can update all previously-released LAs (up or down).
    - Tomorrow (Q0, J0):
      - LA1: Testing (sufficient coverage)
    - After Q1 & J1:
      - LA0: Process
      - LA4: Construction
    - After Q2, J2:
      - LA2: High Level Design
      - LA5: Security & Ethics
    - After Q3, J3:
      - LA3: Low Level Design

Examinable skills

You should be able to…

- Explain what technical debt and code smells are.
  - Be able to describe how these challenges emerge as code evolves.
- Explain the concept of refactoring.
- Explain when to refactor and when not to refactor.
- Identify code smells and be able to identify and carry out appropriate refactorings:
  - Feature envy/Law of Demeter violation → move method and where to put the new method.
  - Magic numbers → Constants or variables.
  - Comments → extract methods and pipe through parameters.
  - Same method in multiple subclasses → pull up method.
  - Almost duplicate code → refactor to template.
  - Switch on type → refactor to use polymorphism.
  - Divergent changes → extract class.
  - Shotgun surgery → introduce class(ses) and move methods.
  - Long Methods → introduce sub-methods.
  - Long Parameter List → introduce class for parameters.
- Explain that the underpinning principle of design is that changes should be as localised as possible.
- Achieve well named methods, identify method aesthetics, and when they are violated.
- Be able to decrease feature envy by moving a method and know where to put the new method(s).
- Be able to identify and fix of violations of the Law of Demeter.

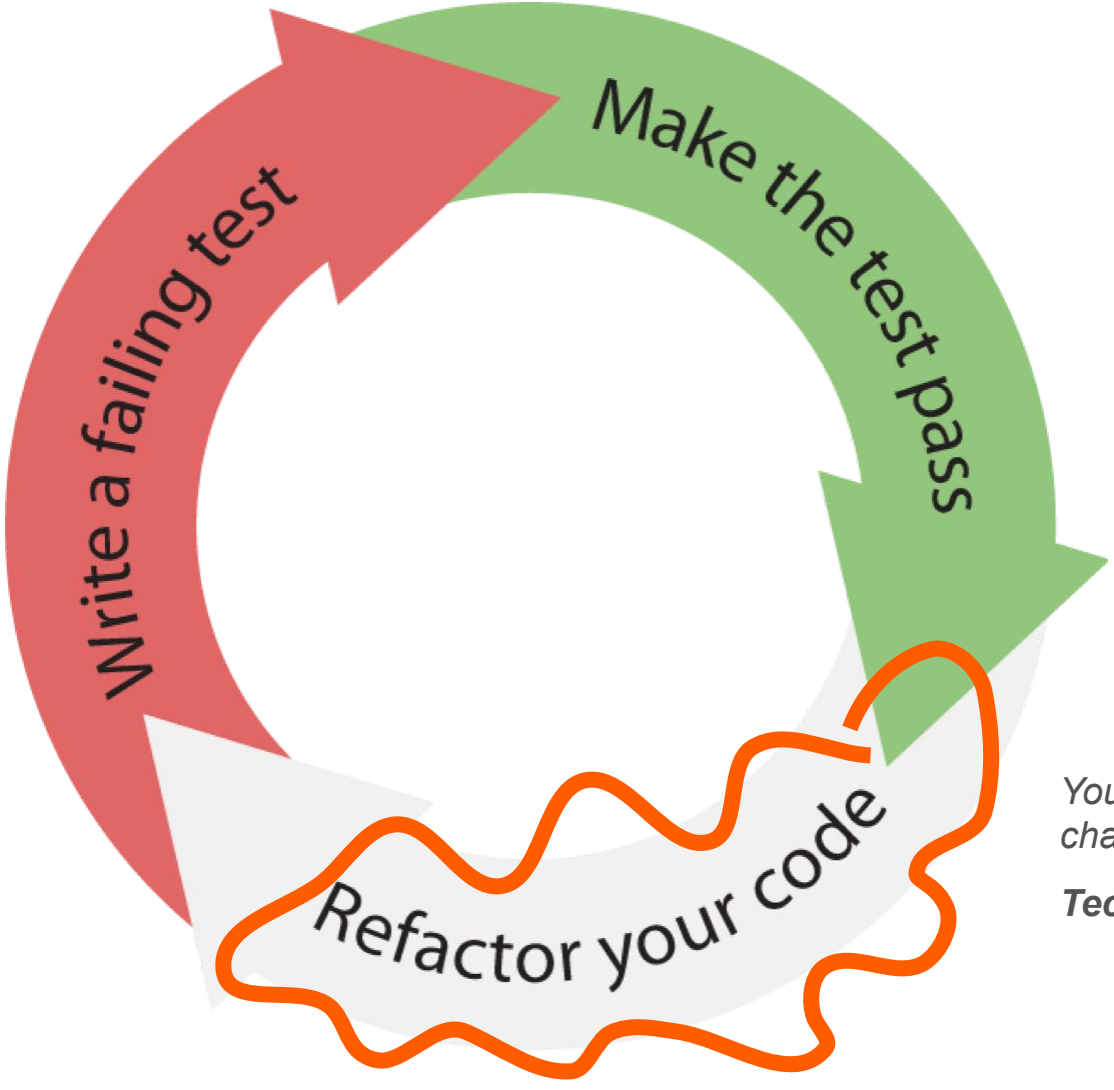
Agile software process: A deeper look

Test driven design (write tests first).

Emergent design: Start at the **Minimum Viable Product**, then grow from there, identifying duplication and introducing abstractions as needed so solve duplication.

Refactor code: rather than doing big design first (make pragmatic choices along the way -- this is not a license to write terrible code) AND the architectural spike must come first.

Tests against spec



You want to make a change but….

Technical debt!!!

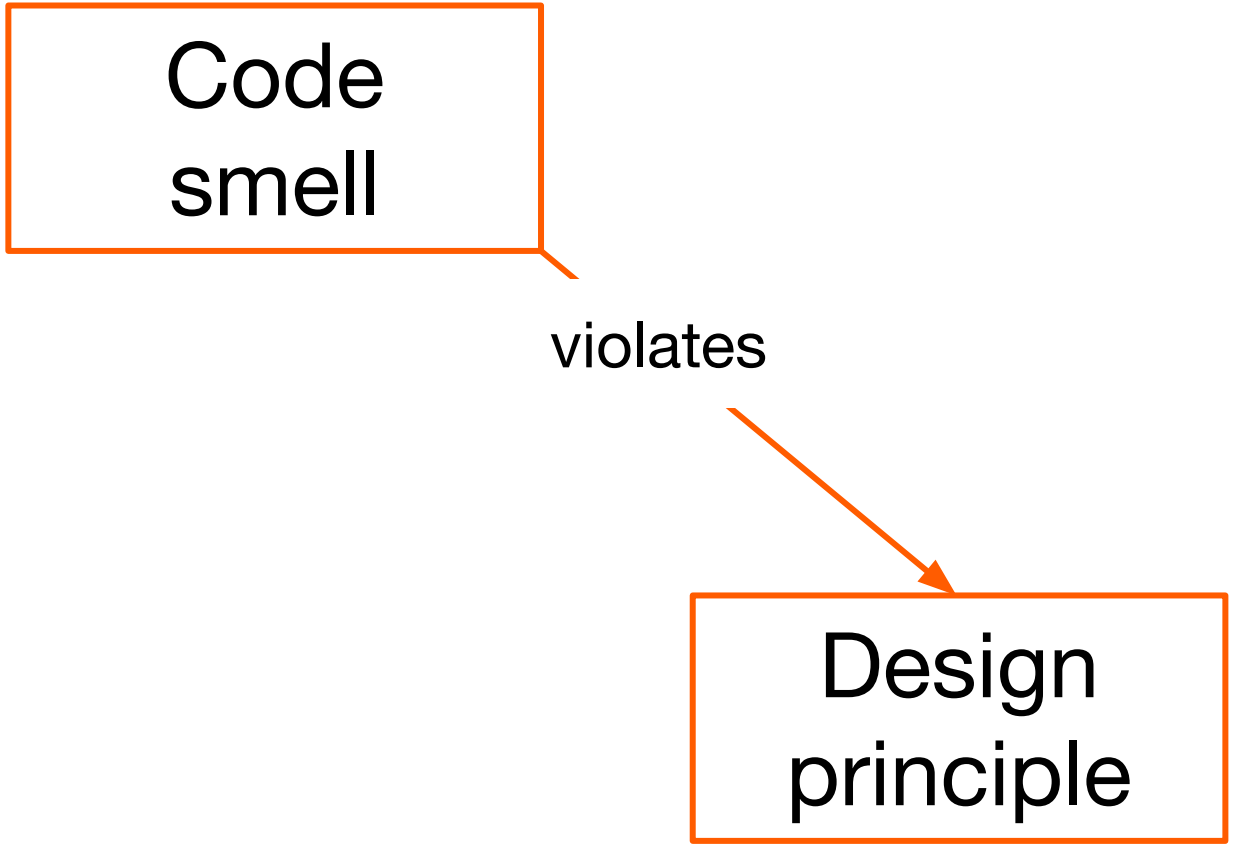
Technical debt

“Listening” to code

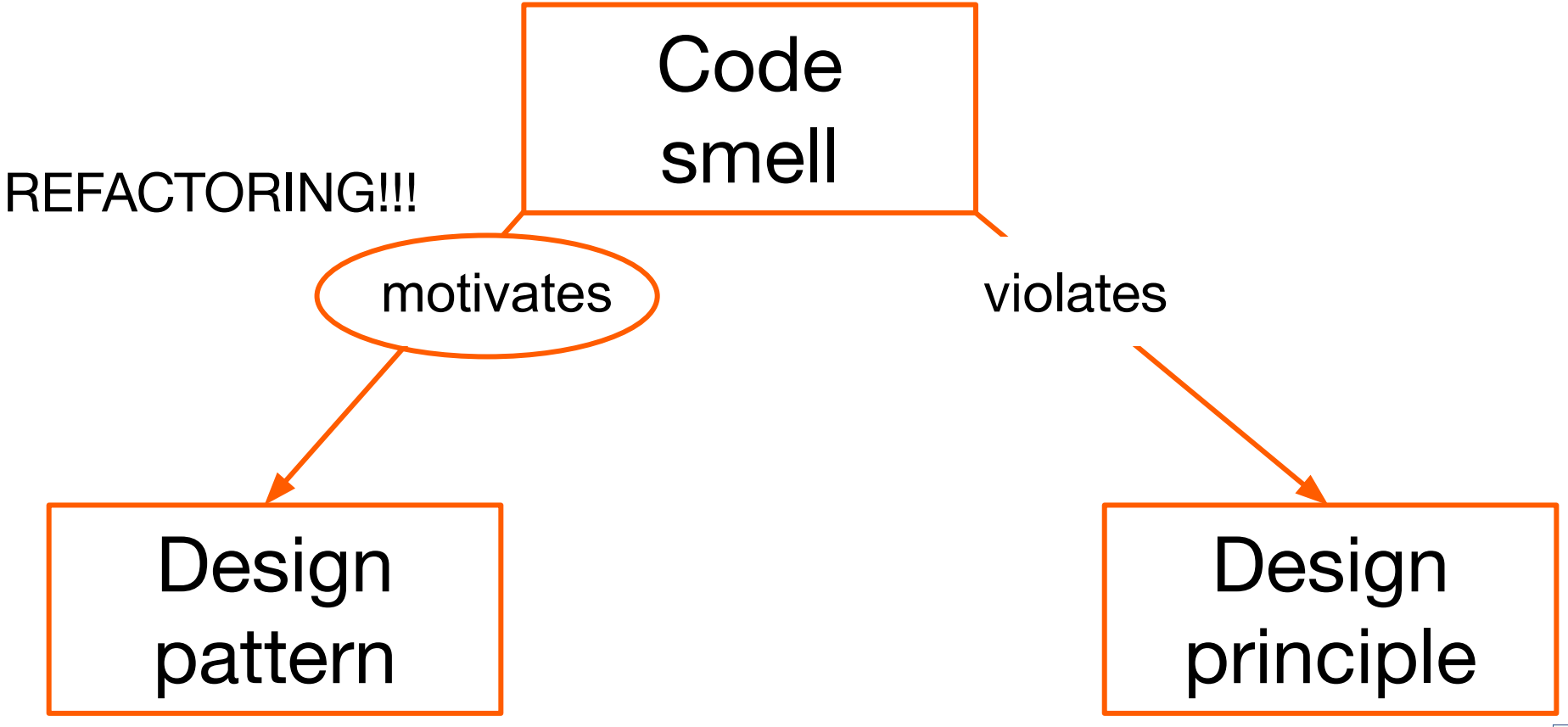
“Listening” to code

Code smell

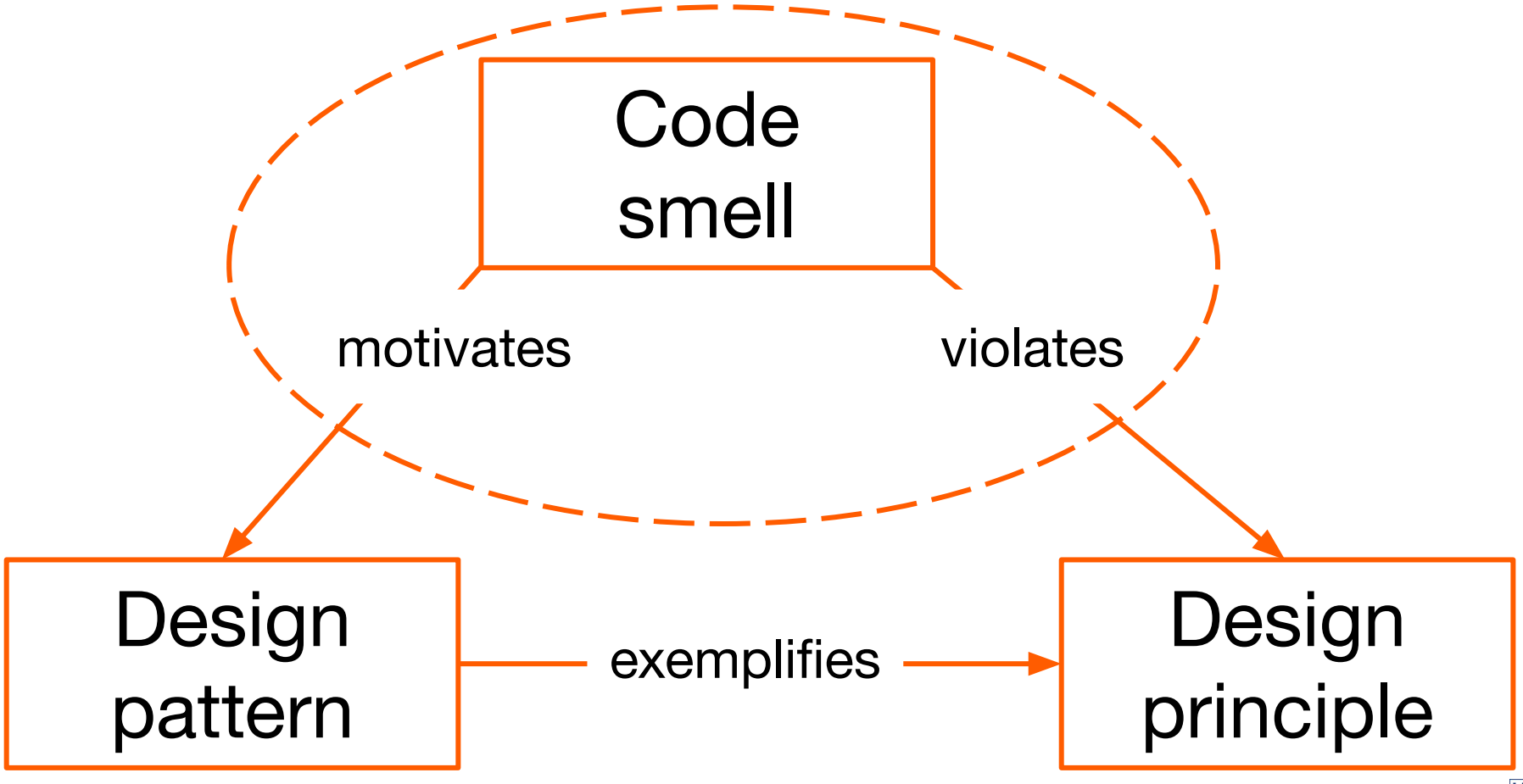
“Listening” to code



“Listening” to code



“Listening” to code



Refactoring timeline

- NOT:
  - 2 weeks of every 6 months.
  - When the tests are failing.
  - When you should just rewrite the code.
  - When you fix a bug.
- Instead, opportunistically during development:
  - If you recognize a warning sign (smell).
  - Before you start a new function.
  - After you finish a new function.
  - When you review code/your code is reviewed.

What is refactoring?

Predictable and meaning* preserving code transformations.

*meaning == semantics == behaviour

Craftsmanship manifesto

Not only **working** software,
but also **well-crafted** software;
Not only **responding** to change,
but also steadily **adding value**.

Broken code

- Code must be able to do three things; if any of these are missing, it can be considered broken:
  - Its job (execute according to its purpose).
  - Afford change.
  - Be understandable.

```
// ex 1:
q= ((p<1) ? (p?0:1) : (p==-4) ? 2 : (p+1));

// ex 2:
while (*a++ = *b--);

// ex 3:
while (n—) {
  for (s=t,d=e; *s; s++) {
    for (p=v+3; *p; p++) {
      // … and more
    }
}
```

```
1  // This function takes in name as a parameter
2  // and logs a string which greets that name
3  // using the information passed
4  function  sayHi (name) {
5       console.log ("Hi" + name + ", nice to meet you." )
6   }
7
8  sayHi("Sam");
```

Minified Code

```
1  function sayHi(o) {console.log ("Hi"+o+", nice to meet you.") }sayHi("Sam");
```

https://www.cloudflare.com/learning/performance/why-minify-javascript-code/#:~:text=This%20includes%20the%20removal%20of,results%20in%20compact%20file%20size.&text=Minification%20speeds%20up%20webpage%20loading,visitors%20and%20search%20engines%20happy.

Technical debt

Design choices that were made in the interest of **time** or **budget**, rather than **technical reasons**.

These accrue over time and often require broad system **restructuring** to decrease debt introduced by past poor decisions.

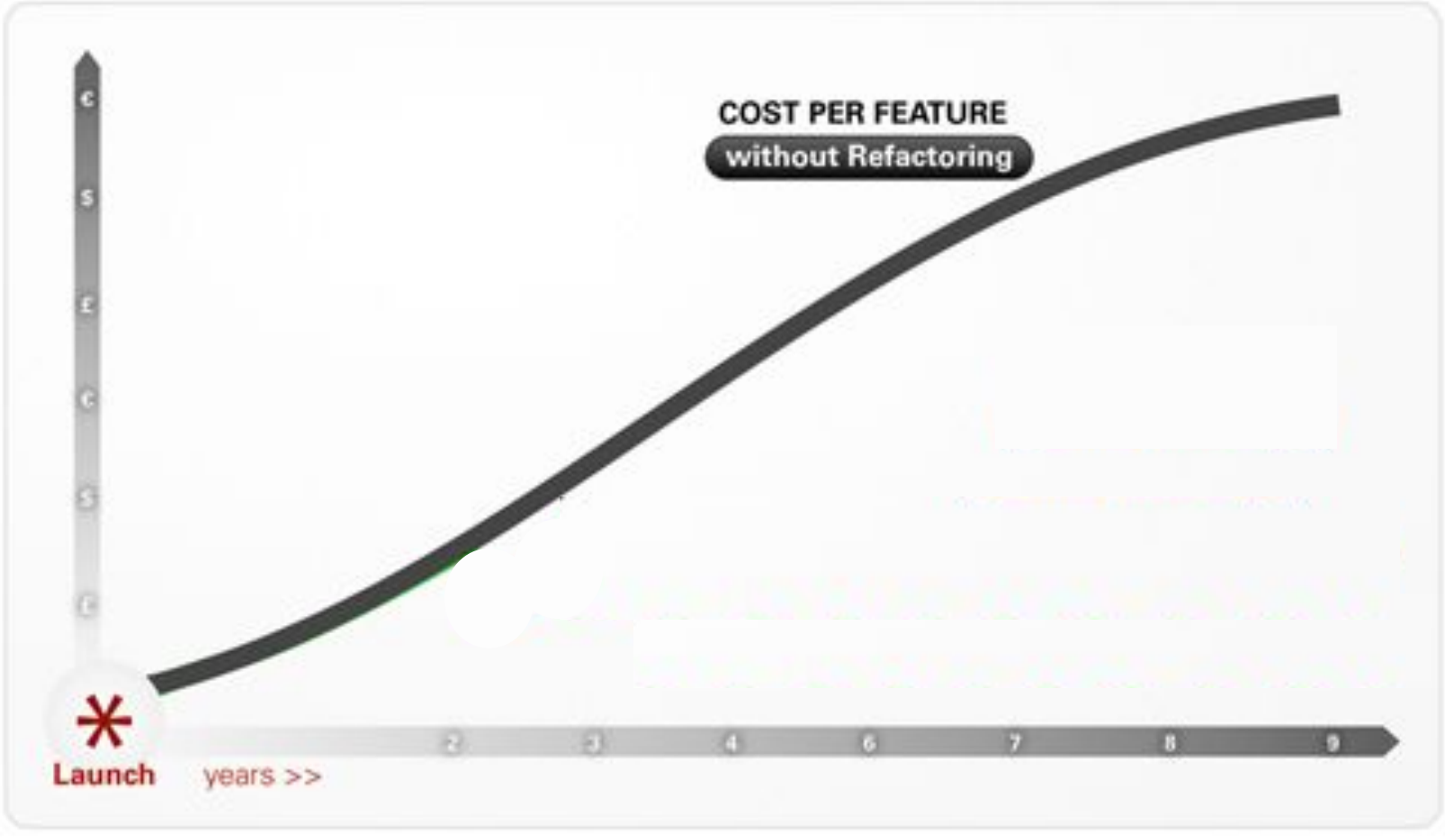
Technical debt *needs to be allocated on a different "budget" or put into an engineering task and discussed with the customer so that they can decide about its value.*

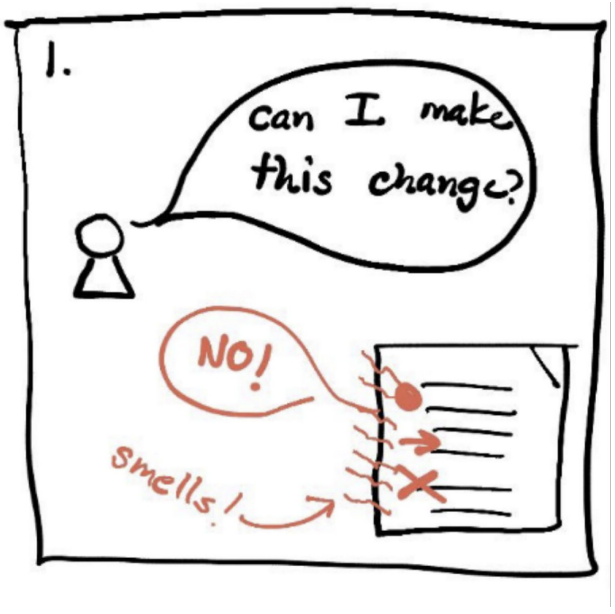
Additionally - some technical debt related user stories can exist, in which case you might see the "user" as a subsystem. "As the payment system, I need refactoring of the underlying code to facilitate the x change to the costing algorithm". This is sometimes a necessity, but not the norm.

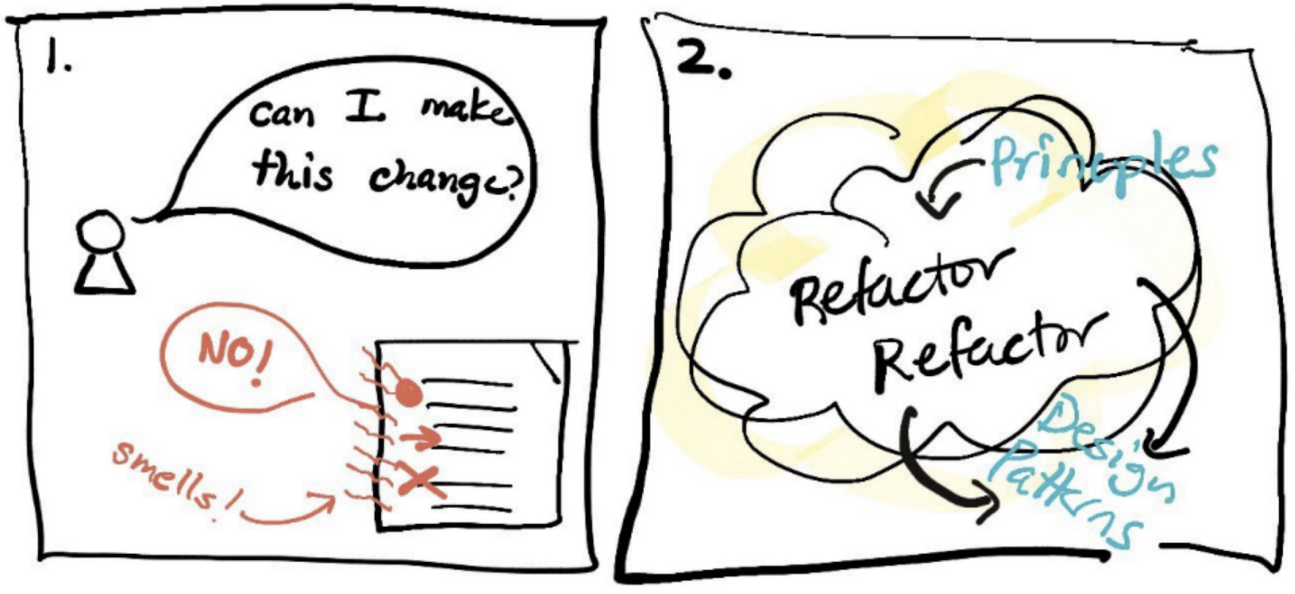
	Reckless	Prudent
Deliberate		
Inadvertent		

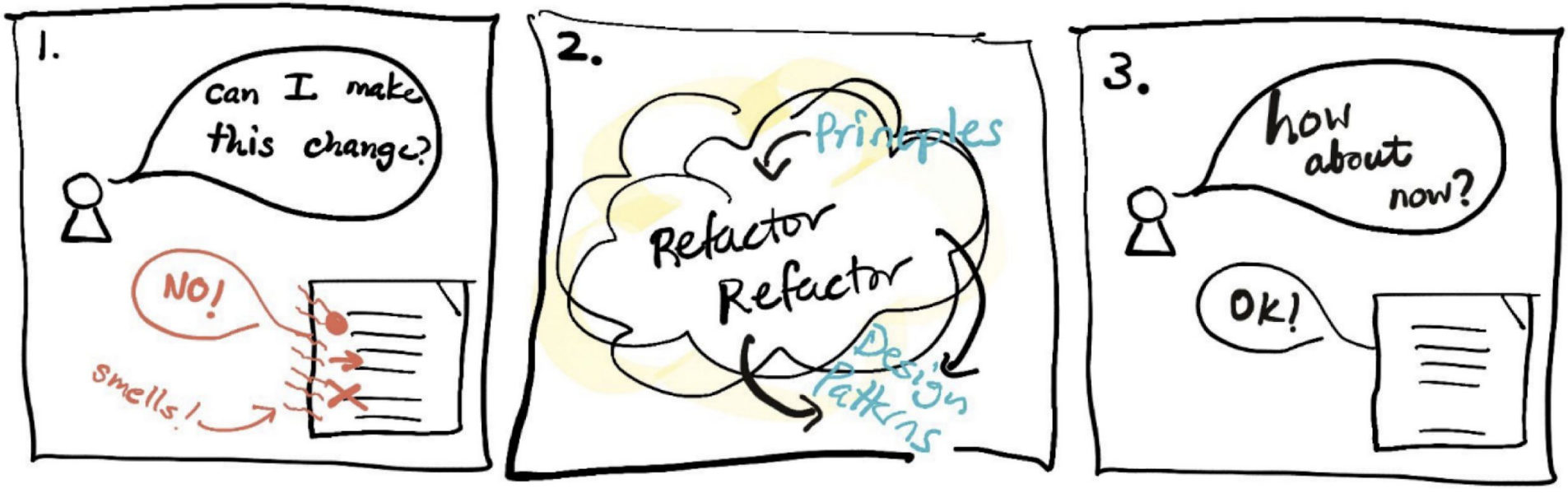
	Reckless	Prudent
Deliberate	"We don't have **time** to think about it"	"Our deadline is firm, but the **risk** is worth it"
Inadvertent	"It **will work** in production without any testing"	"We were **lucky**, but next time we can do better"

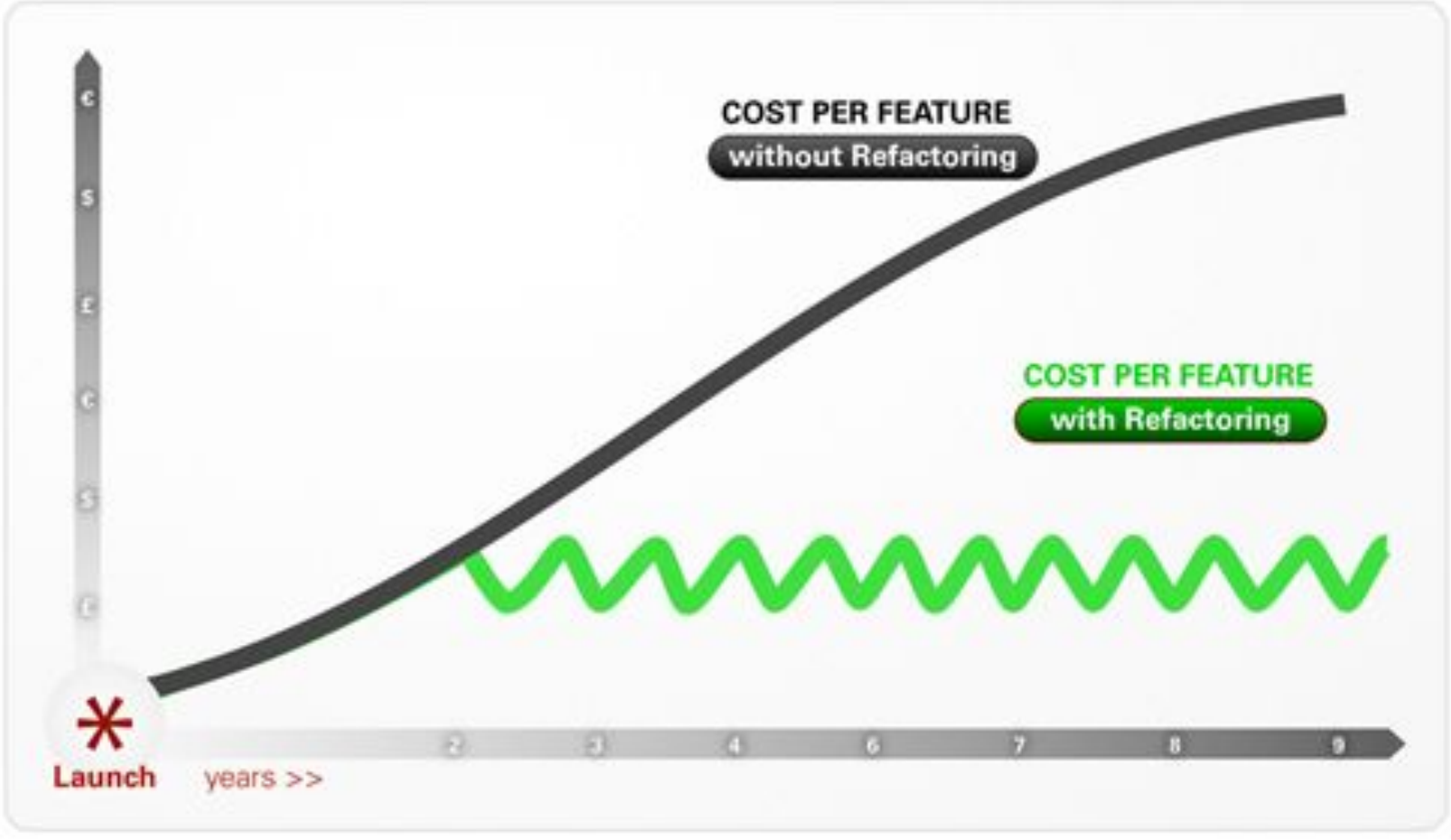
	Reckless	Prudent
Deliberate	Irresponsible	Intentional
Inadvertent	Incompetent	Accidental


COST PER FEATURE
without Refactoring
Launch
years >>


1.
can I make this change?
NO!
smells!


1.
Can I make this change?
NO!
smells!
2.
Principles
Refactor
Refactor
Design Patterns

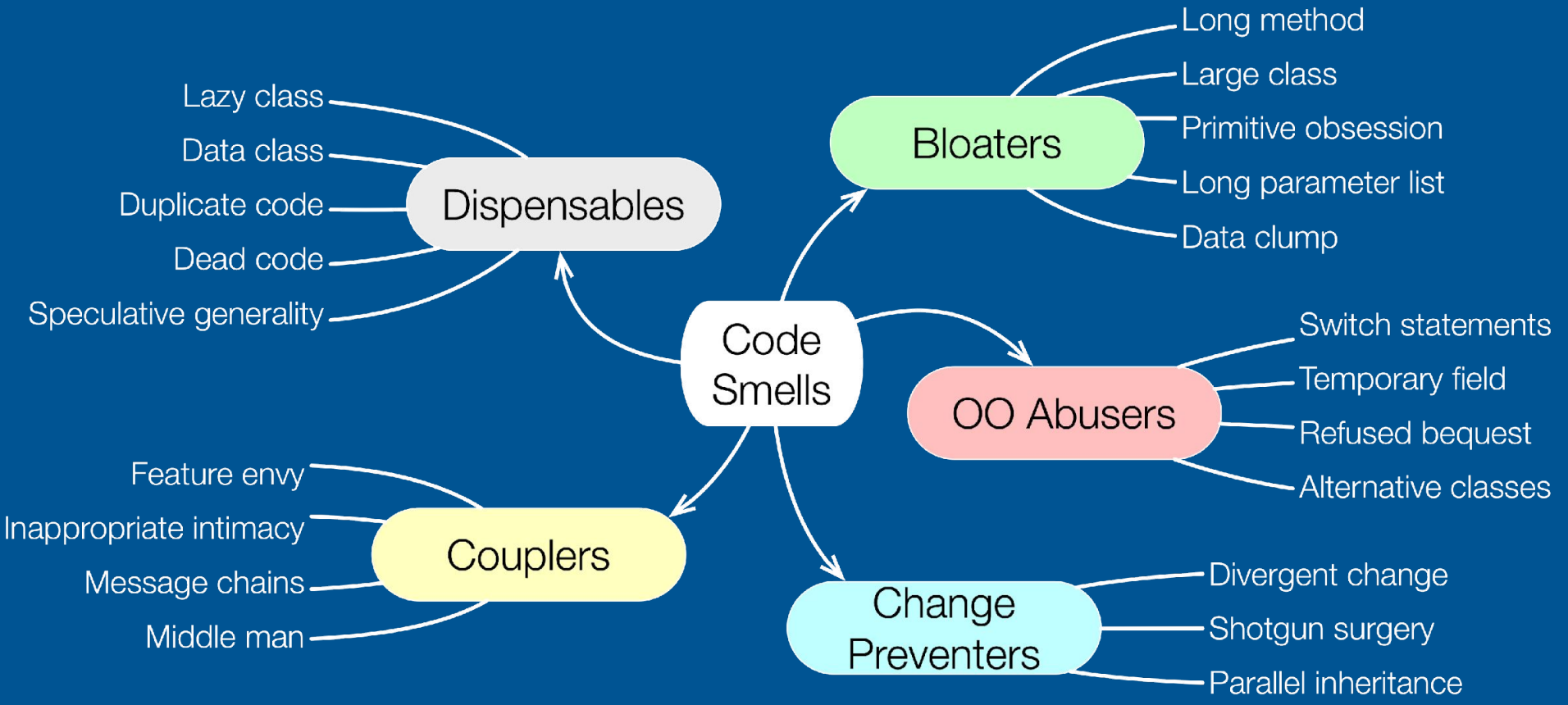

1.
can I make this change?
NO!
smells!
2.
Principles
Refactor
Refactor
Design Patterns
3.
how about now?
Ok!


COST PER FEATURE
without Refactoring
COST PER FEATURE
with Refactoring
Launch
years >>

2 Tricks of Agile:

1) TESTS. Write tests that ensure that you know that your code is meeting spec (or the tests fail until you meet spec).
2) Fluency with REFACTORING.

Symptoms of technical debt



Motivation for refactoring …

- As the code changes, code issues emerge.
- If it's larger scale it's called **Technical Debt.**
- If it's a cosmetic issue you can see, it's called a **Code Smell**.
- Technical debt/smells are **expected** in agile methodologies!
- **Technical Debt and Code Smells make the code hard to evolve.** It's difficult to add new features when you have technical debt or code smells that make simple changes complex to implement.
- Technical debt and code smells both refer to code that needs be restructured **before** changes can be safely made.

Emergent code smells / tech. debt

```
public statement(): string {
    let totalAmount = 0;
    let result = "Rental Record for " + this.getName() + ":\n";
    for (const rental of this.rentals) {
        let rentalAmount = 0;
        switch (rental.getMovie().getPriceCode()) {
            case Movie.REGULAR:
                rentalAmount += 2;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1.5;
                }
                break;
            case Movie.NEW_RELEASE:
                rentalAmount += rental.getDaysRented() * 3;
                break;
        }
        totalAmount += rentalAmount;
    }
    result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
    return result;
}
```

Just one more thing!

```
public statement(): string {
    let totalAmount = 0;

    let result = "Rental Record for " + this.getName() + ":\n";
    for (const rental of this.rentals) {
        let rentalAmount = 0;
        // determine amounts for rental movie rented
        switch (rental.getMovie().getPriceCode()) {
            case Movie.REGULAR:
                rentalAmount += 2;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1.5;
                }
                break;
            case Movie.NEW_RELEASE:
                rentalAmount += rental.getDaysRented() * 3;
                break;
            case Movie.CHILDRENS:
                rentalAmount += 1.5;
                if (rental.getDaysRented() > 3) {
                    rentalAmount += (rental.getDaysRented() - 3) * 1.5;
                }
                break;
        }
        totalAmount += rentalAmount;
    }
    result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
    return result;
}
```

I lied!
Now a
delocalized
feature.

```
public statement(): string {
    let totalAmount = 0;
    let frequentRenterPoints = 0;

    let result = "Rental Record for " + this.getName() + ":\n";
    for (const rental of this.rentals) {
        let rentalAmount = 0;
        // determine amounts for rental movie rented
        switch (rental.getMovie().getPriceCode()) {
            case Movie.REGULAR:
                rentalAmount += 2;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1.5;
                }
                break;
            case Movie.NEW_RELEASE:
                rentalAmount += rental.getDaysRented() * 3;
                break;
            case Movie.CHILDRENS:
                rentalAmount += 1.5;
                if (rental.getDaysRented() > 3) {
                    rentalAmount += (rental.getDaysRented() - 3) * 1.5;
                }
                break;
        }
        totalAmount += rentalAmount;
        frequentRenterPoints++;
    }
    result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
    result += "You earned " + frequentRenterPoints.toFixed(0) + " points."
    return result;
}
```

Tapping into new markets.

```
public statement(): string {
    let totalAmount = 0;
    let frequentRenterPoints = 0;

    let result = "Rental Record for " + this.getName() + ":\n";
    for (const rental of this.rentals) {
        let rentalAmount = 0;
        // determine amounts for rental movie rented
        switch (rental.getMovie().getPriceCode()) {
            case Movie.REGULAR:
                rentalAmount += 2;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1.5;
                }
                break;
            case Movie.NEW_RELEASE:
                rentalAmount += rental.getDaysRented() * 3;
                break;
            case Movie.CHILDRENS:
                rentalAmount += 1.5;
                if (rental.getDaysRented() > 3) {
                    rentalAmount += (rental.getDaysRented() - 3) * 1.5;
                }
                break;
            case Movie.SENIOR:
                rentalAmount += 1.5;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1;
                }
                break;
        }
        totalAmount += rentalAmount;
        frequentRenterPoints++;
        if ((rental.getMovie().getPriceCode() === Movie.SENIOR)) {
            frequentRenterPoints += rental.getDaysRented() -1;
        }
    }
    result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
    result += "You earned " + frequentRenterPoints.toFixed(0) + " points.";
    return result;
}
```

And then student discounts.
Then more details about movies.
How about mid-week deals?
Who doesn't love the classics?!

...

...

```
public statement(): string {
    let totalAmount = 0;
    let frequentRenterPoints = 0;

    let result = "Rental Record for " + this.getName() + ":\n";
    for (const rental of this.rentals) {
        let rentalAmount = 0;
        // determine amounts for rental movie rented
        switch (rental.getMovie().getPriceCode()) {
            case Movie.REGULAR:
                rentalAmount += 2;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1.5;
                }
                break;
            case Movie.NEW_RELEASE:
                rentalAmount += rental.getDaysRented() * 3;
                break;
            case Movie.CHILDRENS:
                rentalAmount += 1.5;
                if (rental.getDaysRented() > 3) {
                    rentalAmount += (rental.getDaysRented() - 3) * 1.5;
                }
                break;
            case Movie.SENIOR:
                rentalAmount += 1.5;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1;
                }
                break;
        }
        totalAmount += rentalAmount;
        frequentRenterPoints++;
        if ((rental.getMovie().getPriceCode() === Movie.SENIOR)) {
            frequentRenterPoints += rental.getDaysRented() -1;
        }
    }
    result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
    result += "You earned " + frequentRenterPoints.toFixed(0) + " points.";
    return result;
}
```

Over time code is edited by many people and accumulates cruft.

- duplication
- rigidity
- coupling

```
public statement(): string {
    let totalAmount = 0;
    let frequentRenterPoints = 0;

    let result = "Rental Record for " + this.getName() + ":\n";
    for (const rental of this.rentals) {
        let rentalAmount = 0;
        // determine amounts for rental movie rented
        switch (rental.getMovie().getPriceCode()) {
            case Movie.REGULAR:
                rentalAmount += 2;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1.5;
                }
                break;
            case Movie.NEW_RELEASE:
                rentalAmount += rental.getDaysRented() * 3;
                break;
            case Movie.CHILDRENS:
                rentalAmount += 1.5;
                if (rental.getDaysRented() > 3) {
                    rentalAmount += (rental.getDaysRented() - 3) * 1.5;
                }
                break;
            case Movie.SENIOR:
                rentalAmount += 1.5;
                if (rental.getDaysRented() > 2) {
                    rentalAmount += (rental.getDaysRented() - 2) * 1;
                }
                break;
        }
        totalAmount += rentalAmount;
        frequentRenterPoints++;
        if ((rental.getMovie().getPriceCode() === Movie.SENIOR)) {
            frequentRenterPoints += rental.getDaysRented() -1;
        }
    }
    result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
    result += "You earned " + frequentRenterPoints.toFixed(0) + " points.";
    return result;
}
```

Emergent abstraction

Any time you are trying to make a change, and it is difficult because either...

- You have to make that change in more than one place.
- Changes (you can tell) will result in merge conflicts or changes to the same file as your teammate.
- The code is too obfuscated to be clearly understood.

...you run the risk of introducing bugs!

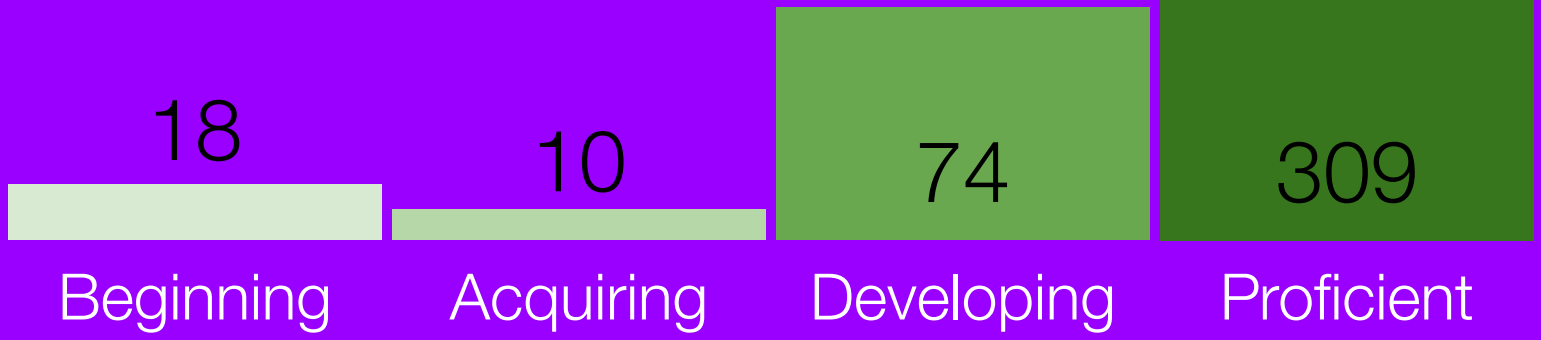
Emergent abstraction

To avoid this: Refactor before you make the change by introducing an abstraction to either:

- Solve duplication/clone/scattering that causes the multiple change locations.
- Reduce tangling between the pieces of functionality by splitting them up.
- Increase code readability to make it more understandable.

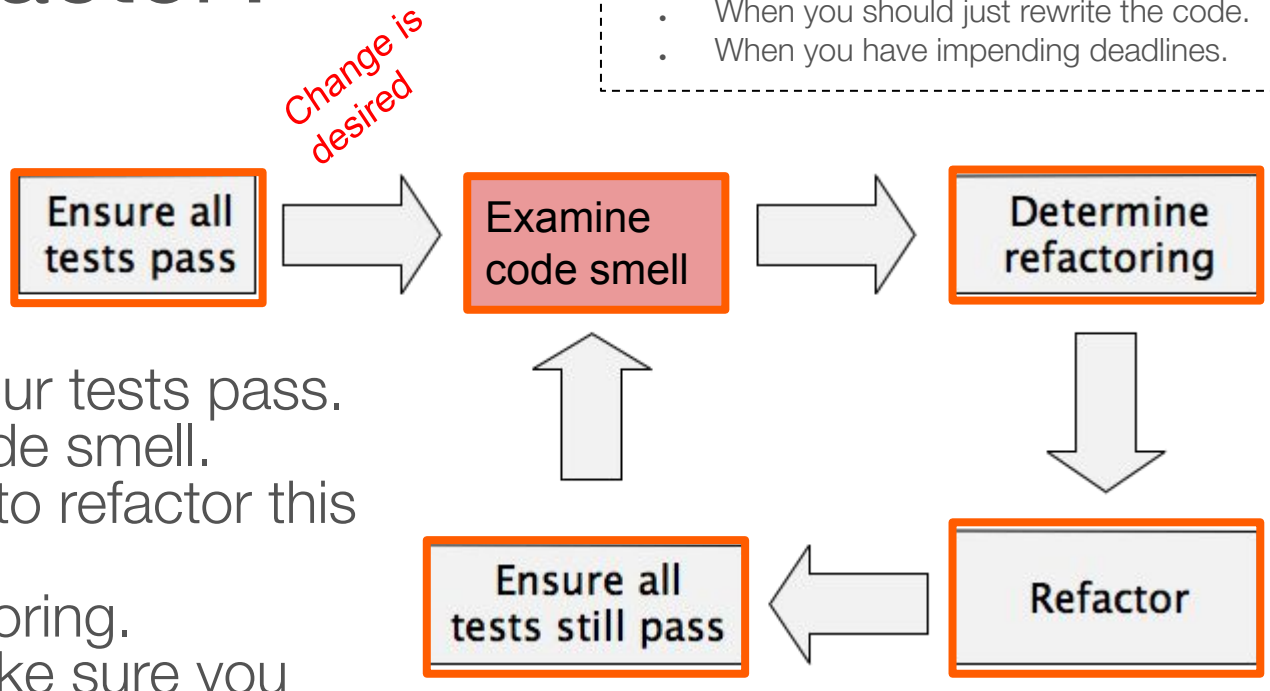
C1

- AutoTest latency < 15 mins whole deliv.
- 50% of teams at proficient by Tuesday.
- Classy should reflect your c1 bucket.
  - Your <highest,earliest> commit.
  - Issue will appear on your project repo soon.
- J1 due Friday @ 1800.
- C2 posted.
- #c2 and **#check** activated for C2.



How to Refactor?

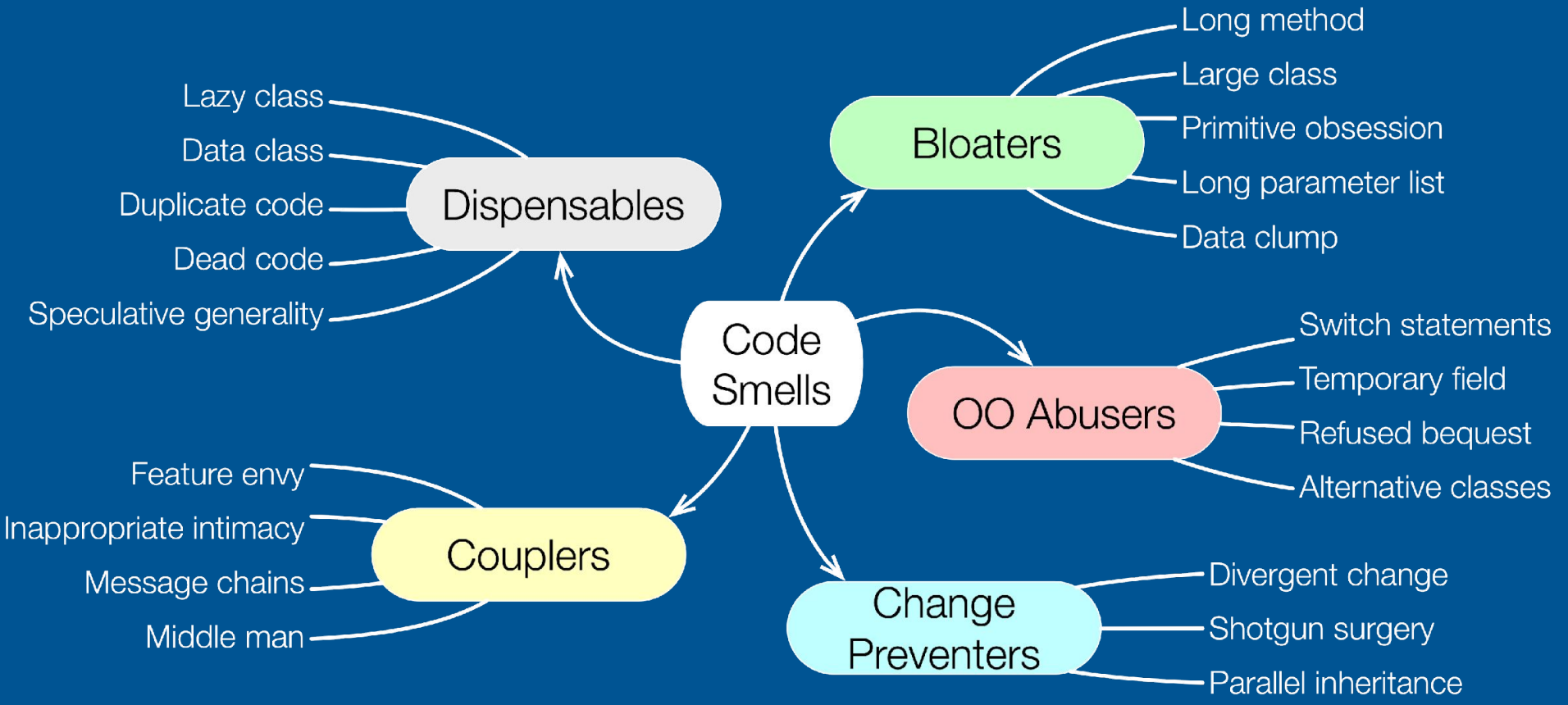
Don't refactor...
- When the tests are failing.
- When you should just rewrite the code.
- When you have impending deadlines.



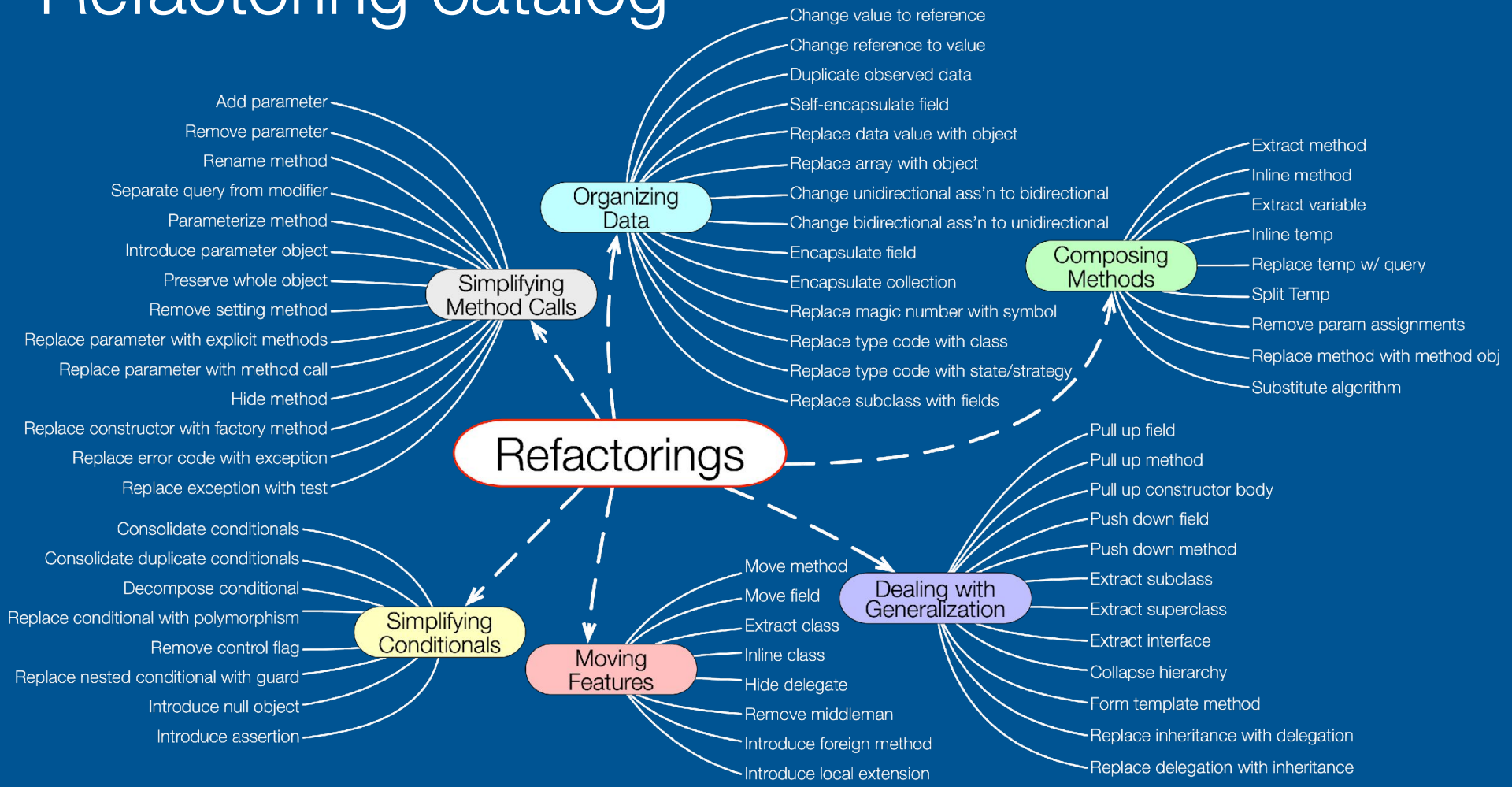
1. Make sure all your tests pass.
2. Examine the code smell.
3. Determine how to refactor this code.
4. Apply the refactoring.
5. Run tests to make sure you didn't break anything.
6. Repeat until the smell is gone.

"apply a refactoring"

Symptoms of technical debt



Refactoring catalog



How Do We Deal With Magic Numbers?

"Replace magic number with symbol" is a typical approach, but not the only one!

Any use of an actual number in the code.

```
potentialEnergy(mass: number, height: number): number {
        return mass * 9.81 * height;
}
```

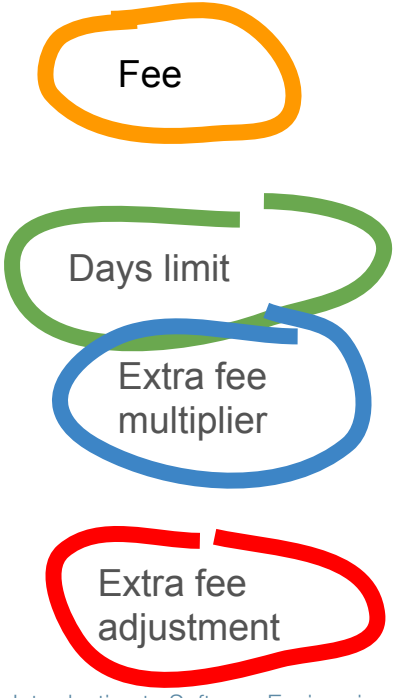
a) *You have to make that change in more than one place.*
b) *Changes (you can tell) will result in merge conflicts.*
c) *The code is too obfuscated to be clearly understood.*

Maybe make this a constant and push your meanings into the type system.

```
potentialEnergy(mass: number, height: number): number {
    const GRAVITATIONAL_CONSTANT = 9.81;
    return mass * GRAVITATIONAL_CONSTANT * height;
}
```

Magic Number Proliferation

What to do?




```
//determine amount for each movie rented
switch (rental.getMovie().getPriceCode()){
    case "Regular":
        thisAmount=2;
        if(rental.getDaysRented()>2)
            thisAmount+=rental.getDaysRented();
        break;
    case "NEW RELEASE":
        thisAmount+=rental.getDaysRented()*3;
        break;
    case "CHILDRENS":
        thisAmount += 1.5;
        if (rental.getDaysRented()>3)
            thisAmount+=(rental.getDaysRented()-3)*1.5;
}
```

Transforming Magic Numbers

If these are not constants,
we could make them variables.

```
let fee = 0;
let daysLimit = 0;
let extraFeeMultiplier = 0;
let extraFeeAdjustment = 0;
```

```
//determine amount for each movie rented
switch (rental.getMovie().getPriceCode()){
    case "Regular":
        thisAmount+=2;
        if(rental.getDaysRent
            thisAmount+=renta
        break;
    case "NEW RELEASE":
        thisAmount+=rental.ge
        break;
    case "CHILDRENS":
        thisAmount += 1.5;
        if (rental.getDaysRen
            thisAmount+=(rent
}
```

```
case "Regular":
    fee=2;
    daysLimit=2;
    thisAmount+=fee;
    if(rental.getDaysRented()>daysLimit)
        thisAmount+=rental.getDaysRented();
    break;
```

Magic Numbers Removed

```
//determine amount for each movie rented
switch (rental.getMovie().getPriceCode()){
    case "Regular":
        fee=2;
        daysLimit=2;
        thisAmount+=fee;
        if(rental.getDaysRented()>daysLimit)
            thisAmount+=rental.getDaysRented();
        break;
    case "NEW RELEASE":
        extraFeeMultiplier=3;
        thisAmount+=rental.getDaysRented()*extraFeeMultiplier;
        break;
    case "CHILDRENS":
        fee=1.5;
        daysLimit=3;
        extraFeeAdjustment=3;
        extraFeeMultiplier=1.5;
        thisAmount += fee;
        if (rental.getDaysRented()>daysLimit)
            thisAmount+=(rental.getDaysRented()-extraFeeAdj
        break;
    case "SENIORS":
        fee=1;
        daysLimit=2;
        extraFeeAdjustment=2;
        thisAmount += fee;
        if (rental.getDaysRented()>daysLimit)
            thisAmount+=(rental.getDaysRented()-extraFeeAdjustment);
        break;
}
```

```
let fee = 0;
let daysLimit = 0;
let extraFeeMultiplier = 0;
let extraFeeAdjustment = 0;
```

Long Method

```
public Application() {
    super( title: "JMapViewer Demo");
    singleton = this;
    setSize( width: 300,  height: 300);
    viewer = new LayerViewer( name: "Keywords");
    initialize();

    bing = new BingAerialTileSource();

    // Listen to the map viewer for user operations so components will
    // receive events and update
    map().addJMVListener(this);

    setLayout(new BorderLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setExtendedState(JFrame.MAXIMIZED_BOTH);


    // TODO Move this into separate function for initializing
    // ************************************************************
    final JTextField newQuery = new JTextField( columns: 5);
    JLabel queryLabel = new JLabel( text: "Enter Query: ");
    queryLabel.setLabelFor(newQuery);

    GridBagConstraints c = new GridBagConstraints();
    c.gridwidth = GridBagConstraints.RELATIVE;
    c.fill = GridBagConstraints.NONE;
    c.gridy = 0;
    c.gridx = 0;
    viewer.queryPanel.add(queryLabel, c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill = GridBagConstraints.HORIZONTAL;
    newQuery.setMaximumSize(new Dimension( width: 200,  height: 20)
    c.gridx = 1;
    viewer.queryPanel.add(newQuery, c);

    viewer.queryPanel.add(Box.createRigidArea(new Dimension( wi

    JLabel colorLabel = new JLabel( text: "Select Color: ");
    colorSetter.setBackground(getRandomColor());

    c.gridwidth = GridBagConstraints.RELATIVE;
    c.fill = GridBagConstraints.NONE;
    c.gridy = 1;
    c.gridx = 0;
    viewer.queryPanel.add(colorLabel, c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill = GridBagConstraints.BOTH;
    c.gridx = 1;
    colorSetter.setMaximumSize(new Dimension( width: 200,  height:
    viewer.queryPanel.add(colorSetter, c);

    viewer.queryPanel.add(Box.createRigidArea(new Dimension( wi

    JButton addQueryButton = new JButton( text: "Add New Query");
      c.insets = new Insets(10,0,0,0);  //top padding
    c.gridx = GridBagConstraints.RELATIVE;       //aligned wit
    c.gridwidth = 2;   //2 columns wide
```

```
viewer.queryPanel.add(Box.createRigidArea(new Dimension( width: 5,  height: 5)));

JButton addQueryButton = new JButton( text: "Add New Query");
  c.insets = new Insets(10,0,0,0);  //top padding
c.gridx = GridBagConstraints.RELATIVE;       //aligned with button 2
c.gridwidth = 2;   //2 columns wide
c.gridy = GridBagConstraints.RELATIVE;       //third row
viewer.queryPanel.add(addQueryButton, c);

viewer.queryPanel.setBorder(
        BorderFactory.createCompoundBorder(
                BorderFactory.createTitledBorder("New Query"),
                BorderFactory.createEmptyBorder( top: 5, left: 5, bottom: 5, right: 5)));

addQueryButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (newQuery.getText().equals("")) {
            return; // Only parse query if text is present.
        }
        // Parse only the first word of the entered input, use lowercase.
        String[] keywords = newQuery.getText().split( regex: " ");
        for (String keyword : keywords) {
            addQuery(keyword.toLowerCase());
        }
          addQuery(newQuery.getText().split(" ")[0].toLowerCase());
        newQuery.setText("");
    }
});

// TODO Figure out how to get "Enter" key to submit button.
this.getRootPane().setDefaultButton(addQueryButton);

colorSetter.addMouseListener((MouseAdapter) mouseClicked(e) → {
        if (e.getButton() == MouseEvent.BUTTON1) {
            Color newColor = JColorChooser.showDialog(
                    component: Application.this,
                    title: "Choose Background Color",
                    colorSetter.getBackground());
            // newColor is "null" if user clicks "Cancel" button on JColorChooser popup.
            if (newColor != null) {
                colorSetter.setBackground(newColor);
            }
        }
});
// ************************************************************
```

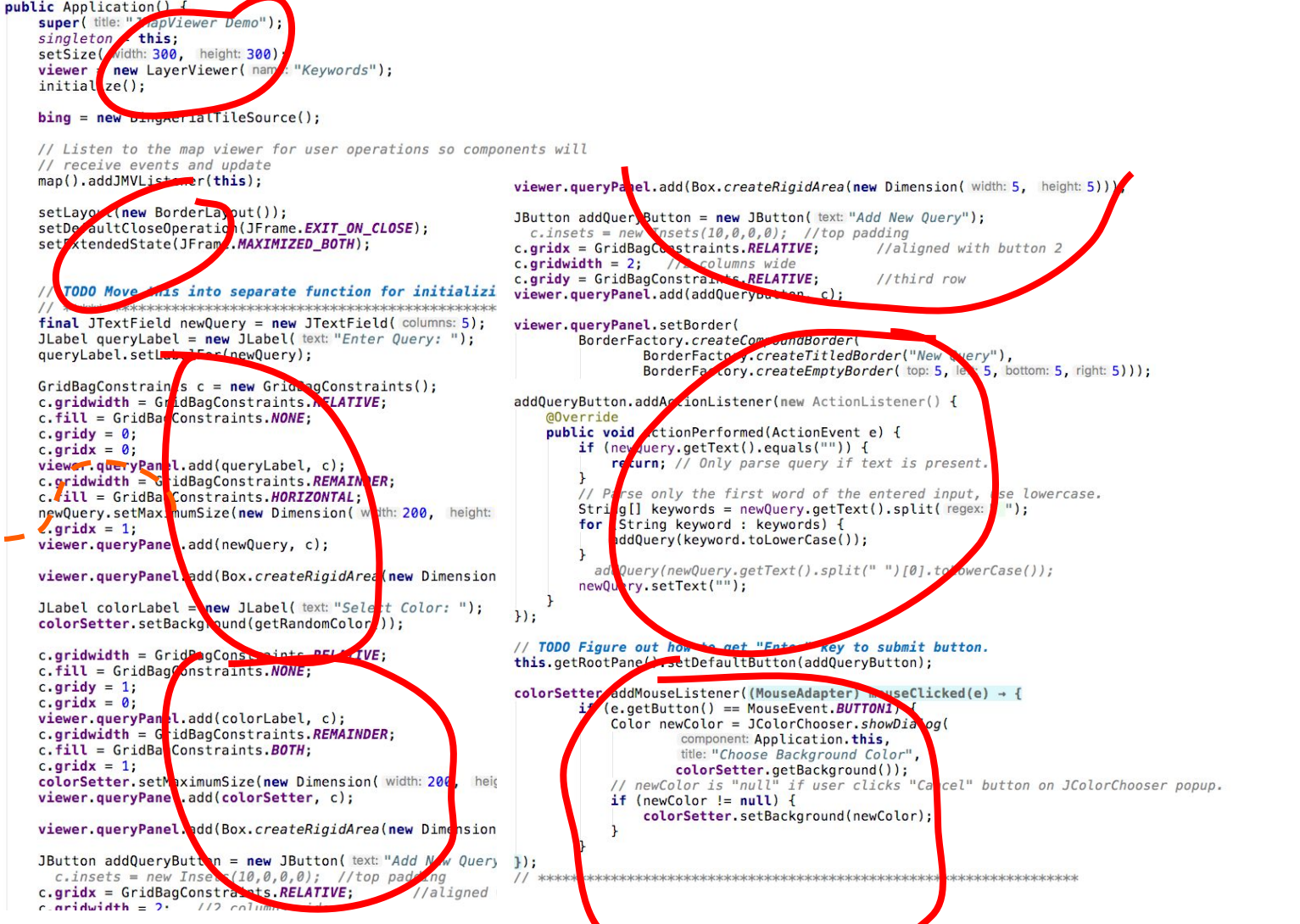
The Problem With Long Methods…

- Practically impossible to understand what's going on.
- Tests cannot help with fault localization because code blocks are not isolatable.
- Doesn't facilitate convenient extension by subtypes.

```
public Application() {
    super( title: "JMapViewer Demo");
    singleton = this;
    setSize( width: 300,  height: 300);
    viewer = new LayerViewer( name: "Keywords");
    initialize();

    bing = new BingAerialTileSource();

    // Listen to the map viewer for user operations so components will
    // receive events and update
    map().addJMVListener(this);

    setLayout(new BorderLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setExtendedState(JFrame.MAXIMIZED_BOTH);


    // TODO Move this into separate function for initializi
    // ********************************************************
    final JTextField newQuery = new JTextField( columns: 5);
    JLabel queryLabel = new JLabel( text: "Enter Query: ");
    queryLabel.setLabelFor(newQuery);

    GridBagConstraints c = new GridBagConstraints();
    c.gridwidth = GridBagConstraints.RELATIVE;
    c.fill = GridBagConstraints.NONE;
    c.gridy = 0;
    c.gridx = 0;
    viewer.queryPanel.add(queryLabel, c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill = GridBagConstraints.HORIZONTAL;
    newQuery.setMaximumSize(new Dimension( width: 200,  height:
    c.gridx = 1;
    viewer.queryPanel.add(newQuery, c);

    viewer.queryPanel.add(Box.createRigidArea(new Dimension

    JLabel colorLabel = new JLabel( text: "Select Color: ");
    colorSetter.setBackground(getRandomColor());

    c.gridwidth = GridBagConstraints.RELATIVE;
    c.fill = GridBagConstraints.NONE;
    c.gridy = 1;
    c.gridx = 0;
    viewer.queryPanel.add(colorLabel, c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill = GridBagConstraints.BOTH;
    c.gridx = 1;
    colorSetter.setMaximumSize(new Dimension( width: 200,  hei
    viewer.queryPanel.add(colorSetter, c);

    viewer.queryPanel.add(Box.createRigidArea(new Dimension

    JButton addQueryButton = new JButton( text: "Add New Quer
      c.insets = new Insets(10,0,0,0);  //top padding
    c.gridx = GridBagConstraints.RELATIVE;       //aligned with button 2
    c.gridwidth = 2;   //2 columns wide
    viewer.queryPanel.add(Box.createRigidArea(new Dimension( width: 5,  height: 5)));

    JButton addQueryButton = new JButton( text: "Add New Query");
      c.insets = new Insets(10,0,0,0);  //top padding
    c.gridx = GridBagConstraints.RELATIVE;       //aligned with button 2
    c.gridwidth = 2;   //2 columns wide
    c.gridy = GridBagConstraints.RELATIVE;       //third row
    viewer.queryPanel.add(addQueryButton, c);

    viewer.queryPanel.setBorder(
            BorderFactory.createCompoundBorder(
                    BorderFactory.createTitledBorder("New Query"),
                    BorderFactory.createEmptyBorder( top: 5, left: 5,  bottom: 5, right: 5)));

    addQueryButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (newQuery.getText().equals("")) {
                return; // Only parse query if text is present.
            }
            // Parse only the first word of the entered input, use lowercase.
            String[] keywords = newQuery.getText().split( regex: " ");
            for (String keyword : keywords) {
                addQuery(keyword.toLowerCase());
            }
              addQuery(newQuery.getText().split(" ")[0].toLowerCase());
            newQuery.setText("");
        }
    });

    // TODO Figure out how to get "Enter" key to submit button.
    this.getRootPane().setDefaultButton(addQueryButton);

    colorSetter.addMouseListener((MouseAdapter) mouseClicked(e) → {
            if (e.getButton() == MouseEvent.BUTTON1) {
                Color newColor = JColorChooser.showDialog(
                        component: Application.this,
                        title: "Choose Background Color",
                        colorSetter.getBackground());
                // newColor is "null" if user clicks "Cancel" button on JColorChooser popup.
                if (newColor != null) {
                    colorSetter.setBackground(newColor);
                }
            }
    });
    // ********************************************************************
```

Look for clusters of behaviour

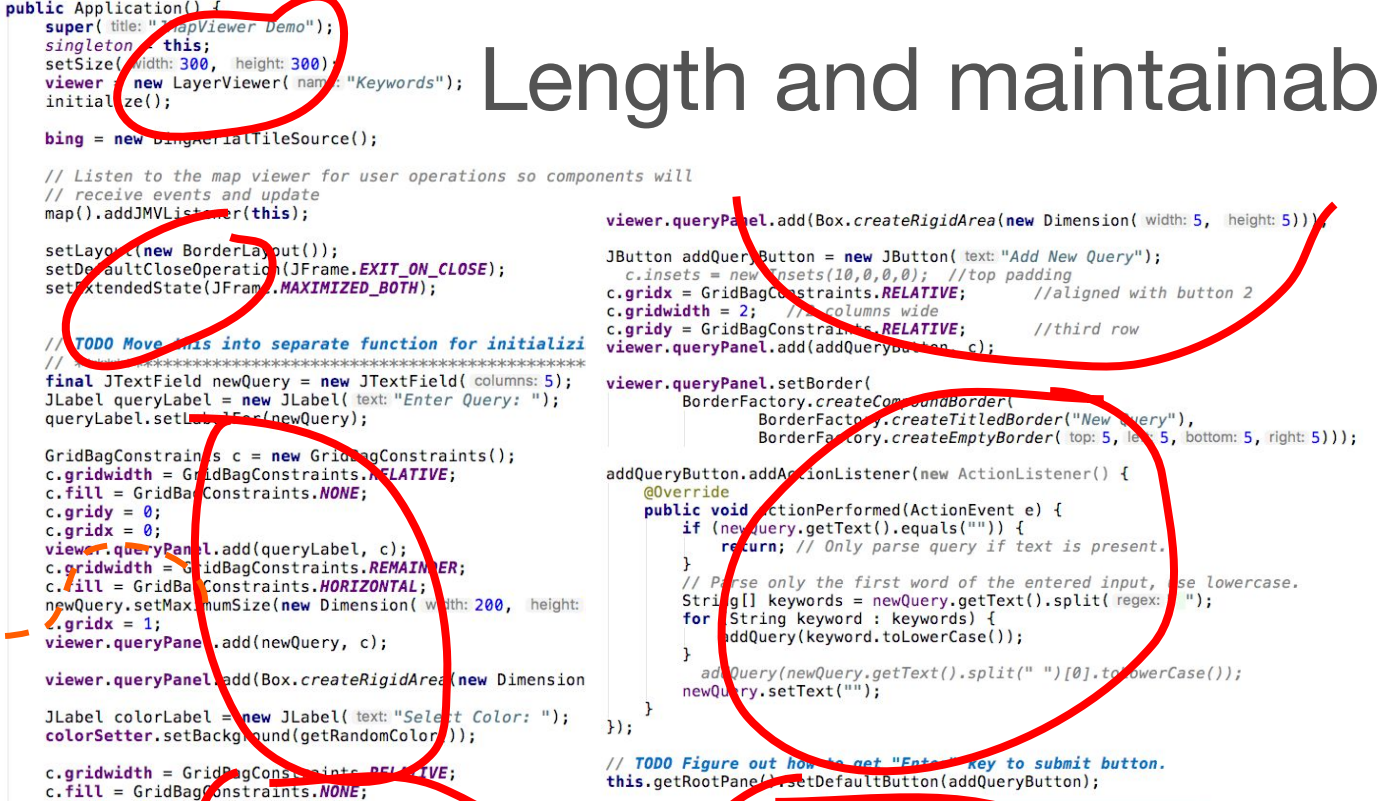
New method



```
public Application() {
    super( title: "MapViewer Demo");
    singleton = this;
    setSize( width: 300,  height: 300);
    viewer = new LayerViewer( name: "Keywords");
    initialize();

    bing = new BingAerialTileSource();

    // Listen to the map viewer for user operations so components will
    // receive events and update
    map().addJMVListener(this);

    setLayout(new BorderLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setExtendedState(JFrame.MAXIMIZED_BOTH);


    // TODO Move this into separate function for initializi
    // ***************************************************
    final JTextField newQuery = new JTextField( columns: 5);
    JLabel queryLabel = new JLabel( text: "Enter Query: ");
    queryLabel.setLabelFor(newQuery);

    GridBagConstraints c = new GridBagConstraints();
    c.gridwidth = GridBagConstraints.RELATIVE;
    c.fill = GridBagConstraints.NONE;
    c.gridy = 0;
    c.gridx = 0;
    viewer.queryPanel.add(queryLabel, c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill = GridBagConstraints.HORIZONTAL;
    newQuery.setMaximumSize(new Dimension( width: 200,  height:
    c.gridx = 1;
    viewer.queryPanel.add(newQuery, c);

    viewer.queryPanel.add(Box.createRigidArea(new Dimension

    JLabel colorLabel = new JLabel( text: "Select Color: ");
    colorSetter.setBackground(getRandomColor());

    c.gridwidth = GridBagConstraints.RELATIVE;
    c.fill = GridBagConstraints.NONE;
    c.gridy = 1;
    c.gridx = 0;
    viewer.queryPanel.add(colorLabel, c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill = GridBagConstraints.BOTH;
    c.gridx = 1;
    colorSetter.setMaximumSize(new Dimension( width: 200,  heig
    viewer.queryPanel.add(colorSetter, c);

    viewer.queryPanel.add(Box.createRigidArea(new Dimension

    JButton addQueryButton = new JButton( text: "Add New Query
      c.insets = new Insets(10,0,0,0);  //top padding
    c.gridx = GridBagConstraints.RELATIVE;       //aligned
    c.gridwidth = 2;   //2 columns wide
```

```
viewer.queryPanel.add(Box.createRigidArea(new Dimension( width: 5,  height: 5)));

JButton addQueryButton = new JButton( text: "Add New Query");
  c.insets = new Insets(10,0,0,0);  //top padding
c.gridx = GridBagConstraints.RELATIVE;       //aligned with button 2
c.gridwidth = 2;   //2 columns wide
c.gridy = GridBagConstraints.RELATIVE;       //third row
viewer.queryPanel.add(addQueryButton, c);

viewer.queryPanel.setBorder(
        BorderFactory.createCompoundBorder(
                BorderFactory.createTitledBorder("New Query"),
                BorderFactory.createEmptyBorder( top: 5, left: 5, bottom: 5, right: 5)));

addQueryButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (newQuery.getText().equals("")) {
            return; // Only parse query if text is present.
        }
        // Parse only the first word of the entered input, use lowercase.
        String[] keywords = newQuery.getText().split( regex: " ");
        for (String keyword : keywords) {
            addQuery(keyword.toLowerCase());
        }
          addQuery(newQuery.getText().split(" ")[0].toLowerCase());
        newQuery.setText("");
    }
});

// TODO Figure out how to get "Enter" key to submit button.
this.getRootPane().setDefaultButton(addQueryButton);

colorSetter.addMouseListener((MouseAdapter) mouseClicked(e) -> {
        if (e.getButton() == MouseEvent.BUTTON1) {
            Color newColor = JColorChooser.showDialog(
                    component: Application.this,
                    title: "Choose Background Color",
                    colorSetter.getBackground());
            // newColor is "null" if user clicks "Cancel" button on JColorChooser popup.
            if (newColor != null) {
                colorSetter.setBackground(newColor);
            }
        }
});
// ***************************************************************
```

Length and maintainability



New method

An Empirical Study on Maintainable Method Size in Java

Shaiful Alam Chowdhury
University of British Columbia
Vancouver, BC, Canada
shaifulc@cs.ubc.ca

Gias Uddin
University of Calgary
Calgary, AB, Canada
gias.uddin@ucalgary.ca

Reid Holmes
University of British Columbia
Vancouver, BC, Canada
rtholmes@cs.ubc.ca

Long Method in Practice

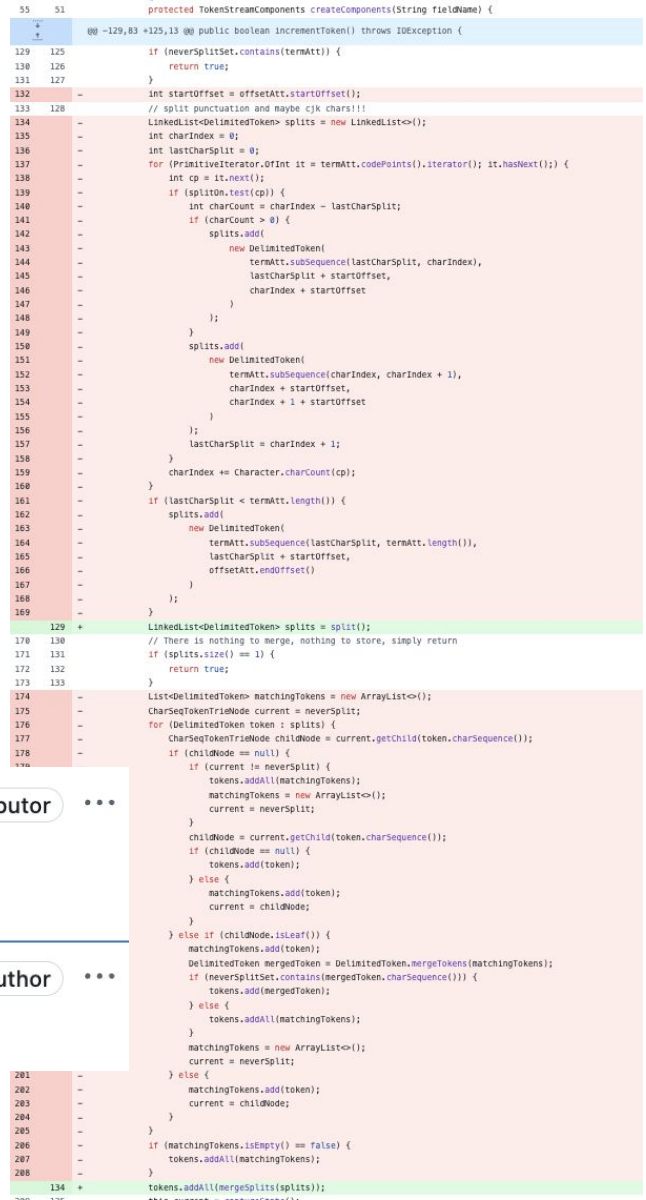
- Original change: reviewer comment
- Fix: fix

dimitris-athanasiou on Feb 3, 2022 Contributor

This method is too long. Could we extract this bit all the way to line 169 into a `splitPunctuation` sort of method?

Long Method in Practice

- Original change: reviewer comment
- Fix: fix



dimitris-athanasiou on Feb 3, 2022 — Contributor

This method is too long. Could we extract this bit all the way to line 169 into a `splitPunctuation` sort of method?

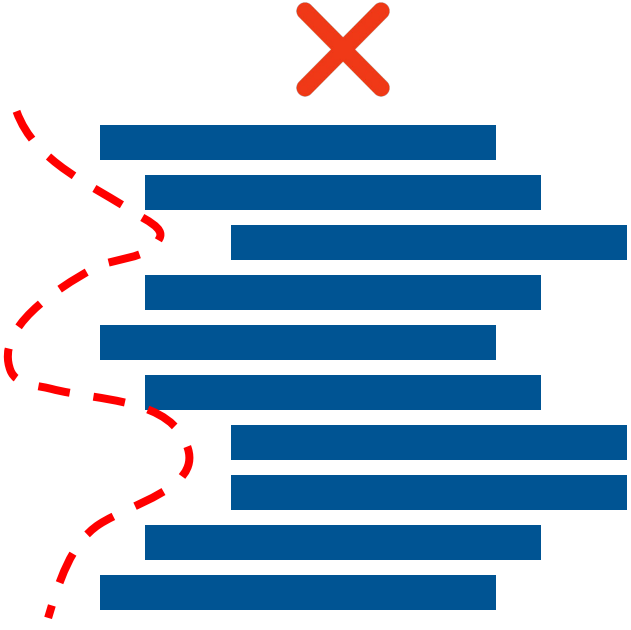
benwtrent on Feb 3, 2022 — Member, Author

for sure

Method Design Aesthetics

All the same level of abstraction.

Differing levels of abstraction.



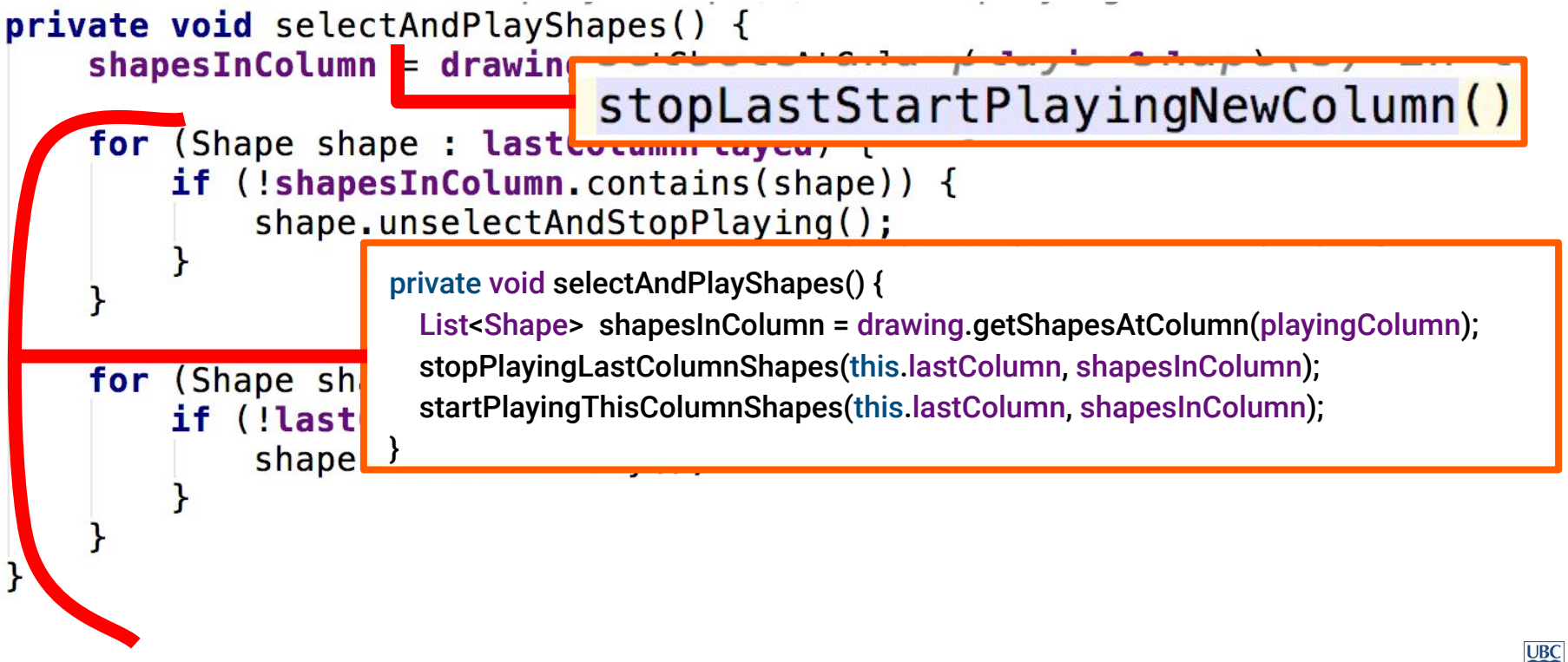
Happens to shorter methods too!

Methods should do just one thing… at the *right* abstraction level.

```
private void selectAndPlayShapes() {
    shapesInColumn = drawing.getShapesAtColumn(playingColumn);

    for (Shape shape : lastColumnPlayed) {
        if (!shapesInColumn.contains(shape)) {
            shape.unselectAndStopPlaying();
        }
    }

    for (Shape shape : shapesInColumn) {
        if (!lastColumnPlayed.contains(shape)) {
            shape.selectAndPlay();
        }
    }
}
```

Happens to shorter methods too!

Methods should do just one thing… at the *right* abstraction level.



Long parameter list

Hard to understand, annoying to use.

```
public pushCommit(
    author: string,
    commitId: string,
    files: File[],
    sha: string,
    branch: string,
    parentSHAs: [],
    ts: Date) : Promise<boolean> {
        // ...
}
```

```
public pushCommit(commit: Commit) : Promise<boolean> {
    // ...
}
```

CREATE PARAMETER OBJECT

- Replace the parameters with an object.

Comment misuse

Good: Comment *technology*.

- Comment technology is *necessary* for specification and documentation (e.g. requires, modifies, effects).

Bad: Comments that *explain* code.

- Comments needed to explain code do not refer to specification or documentation. These are normally anti-patterns.
- Special exception for comments that describe _why_.

Needing explanatory comments

> "... comments often are used as a *deodorant*. It's surprising how often you look at thickly commented code and notice that the *comments are there because the code is bad*."
>
> [Fowler & Beck, Refactoring]

"A good time to use a comment is when you don't know what to do. In addition to describing what is going on, comments can indicate **areas in which you aren't sure**. A comment is a good place to **say why** you did something. This kind of information helps future modifiers, **especially forgetful ones**."

[Fowler]

Needing explanatory comments

```
// convert to cents
a = x * 100;
// avg cents per customer
avg = a / n;
// add to list
if (avgs < avg) { t.push(1) }
```

```
cents = cost * 100;
centsPerCust = cents / totalCust;
if (avgs < avg) { t.push(1) }
```

These comments are actually okay! They are “TODO”s, not explanatory.

```
public Application() {
    super( title: "JMapViewer Demo");
    singleton = this;
    setSize( width: 300,  height: 300);
    viewer = new LayerViewer( name: "Keywords");
    initialize();

    bing = new BingAerialTileSource();

    // Listen to the map viewer for user operations so
    // receive events and update
    map().addJMVListener(this);

    setLayout(new BorderLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setExtendedState(JFrame.MAXIMIZED_BOTH);


    // TODO Move this into separate function for initializing newQuery section of UI.
    // ********************************************************************************
    final JTextField newQuery = new JTextField( columns: 5);
    JLabel queryLabel = new JLabel( text: "Enter Query: ");
    queryLabel.setLabelFor(newQuery);

    GridBagConstraints c = new GridBagConstraints();
    c.gridwidth = GridBagConstraints.RELATIVE;
    c.fill = GridBagConstraints.NONE;
    c.gridy = 0;
    c.gridx = 0;
    viewer.queryPanel.add(queryLabel, c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill = GridBagConstraints.HORIZONTAL;
    newQuery.setMaximumSize(new Dimension( width: 200,  height: 20));
    c.gridx = 1;
    viewer.queryPanel.add(newQuery, c);

    viewer.queryPanel.add(Box.createRigidArea(new Dimension( width: 5,  height: 5)));

    JLabel colorLabel = new JLabel( text: "Select Color: ");
    colorSetter.setBackground(getRandomColor());

    c.gridwidth = GridBagConstraints.RELATIVE;
    c.fill = GridBagConstraints.NONE;
    c.gridy = 1;
    c.gridx = 0;
    viewer.queryPanel.add(colorLabel, c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.fill = GridBagConstraints.BOTH;
    c.gridx = 1;
    colorSetter.setMaximumSize(new Dimension( width: 200,  height: 20));
    viewer.queryPanel.add(colorSetter, c);

    viewer.queryPanel.add(Box.createRigidArea(new Dimension( width: 5,  height: 5)));

    JButton addQueryButton = new JButton( text: "Add New Query");
      c.insets = new Insets(10,0,0,0);  //top padding
    c.gridx = GridBagConstraints.RELATIVE;       //aligned with button 2
    c.gridwidth = 2;   //2 columns wide
```

```
viewer.queryPanel.add(Box.createRigidArea(new Dimension( width: 5,  height: 5)));

JButton addQueryButton = new JButton( text: "Add New Query");
  c.insets = new Insets(10,0,0,0);  //top padding
c.gridx = GridBagConstraints.RELATIVE;       //aligned with button 2
c.gridwidth = 2;   //2 columns wide
c.gridy = GridBagConstraints.RELATIVE;       //third row
viewer.queryPanel.add(addQueryButton, c);

viewer.queryPanel.setBorder(
        BorderFactory.createCompoundBorder(
                BorderFactory.createTitledBorder("New Query"),
                BorderFactory.createEmptyBorder( top: 5, left: 5, bottom: 5, right: 5)));

addQueryButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (newQuery.getText().equals("")) {
            return; // Only parse query if text is present.
        }
        // Parse only the first word of the entered input, use lowercase.
        String[] keywords = newQuery.getText().split( regex: " ");
        for (String keyword : keywords) {
            addQuery(keyword.toLowerCase());
        }
          addQuery(newQuery.getText().split(" ")[0].toLowerCase());
        newQuery.setText("");
    }
});

// TODO Figure out how to get "Enter" key to submit button.
this.getRootPane().setDefaultButton(addQueryButton);

colorSetter.addMouseListener((MouseAdapter) mouseClicked(e) → {
        if (e.getButton() == MouseEvent.BUTTON1) {
            Color newColor = JColorChooser.showDialog(
                    component: Application.this,
                    title: "Choose Background Color",
                    colorSetter.getBackground());
            // newColor is "null" if user clicks "Cancel" button on JColorChooser popup.
            if (newColor != null) {
                colorSetter.setBackground(newColor);
            }
        }
});
// ********************************************************************
```

Why are explanatory comments bad?

- People don't respect explanatory comments!
- You can change code, and not change the explanatory comments - this leads to documentation drift.
- Explanatory comments don't necessarily move with code!
- You could accidentally:
  - Copy and paste code but forget an explanatory comment.
  - Add new code between the explanatory comment and the implementation it refers to.
  - Make a change that makes the comment misleading.

Comments often signal clusters of behaviour - and indicate a method is doing more than it should.

Fixing explanatory comments

- Extract method and give that method a useful name.
- Named elements ***become*** the documentation.
- People respect names: people generally will ***not willingly*** change the contract of a class/method/field.
- Documentation “envelopes” move with code: when you instantiate a class, reference a field, or call a method you implicitly access the documentation through its name.

Video Store comments

```
public statement(): string {
  // generate header
  let result = "Rental Record for " + this.getName() + ":\n";
  for (const rental of this.rentals) {
     let rentalAmount = 0;
     // determine amounts for rental movie rented
     switch (rental.getMovie().getPriceCode()) {
        case Movie.REGULAR:
           rentalAmount += 2;
           if (rental.getDaysRented() > 2) {
              rentalAmount += (rental.getDaysRented() - 2) * 1.5;
           }
           break;
        case Movie.NEW_RELEASE:
           rentalAmount += rental.getDaysRented() * 3;
           break;
     }
     // add frequent renter points
     frequentRenterPoints++;
     // add bonus for a multi-day new release rental
     if ((rental.getMovie().getPriceCode() === Movie.NEW_RELEASE) && rental.getDaysRented() > 1) {
        frequentRenterPoints++;
     }
     // show figures for this rental
     result += "\t" + rental.getMovie().getTitle() + "\t" + rentalAmount.toFixed(2) + "\n";
     totalAmount += rentalAmount;
  }
  // generate footer
  result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
  result += "You earned " + frequentRenterPoints.toFixed(0) + " frequent renter points.";
  console.log(result);

  return result;
}
```

Guide extraction with code comments

```
public statement(): string {
  // generate header
  let result = "Rental Record for " + this.getName() + ":\n";
  for (const rental of this.rentals) {
    let rentalAmount = 0;
    // determine amounts for rental movie rented
    switch (rental.getMovie().getPriceCode()) {
      case Movie.REGULAR:
        rentalAmount += 2;
        if (rental.getDaysRented() > 2) {
          rentalAmount += (rental.getDaysRented() - 2) * 1.5;
        }
        break;
      case Movie.NEW_RELEASE:
        rentalAmount += rental.getDaysRented() * 3;
        break;
    }
    // add frequent renter points
    frequentRenterPoints++;
    // add bonus for a multi-day new release rental
    if ((rental.getMovie().getPriceCode() === Movie.NEW_RELEASE) && rental.getDaysRented() > 1) {
      frequentRenterPoints++;
    }
    // show figures for this rental
    result += "\t" + rental.getMovie().getTitle() + "\t" + rentalAmount.toFixed(2) + "\n";
    totalAmount += rentalAmount;
  }
  // generate footer
  result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
  result += "You earned " + frequentRenterPoints.toFixed(0) + " frequent renter points.";

  return result;
}
```

```
private simplifiedStatement(): string {
  let totalAmount = 0;
  let statement = this.getHeader();
  for (const rental of this.rentals) {
    const rentalAmount = this.getRentalAmount(rental);
    this.addRenterPoints(rental);
    this.addBonuses(rental);
    statement +=
      this.generateStatementLine(rental, rentalAmount);
    totalAmount += rentalAmount;
  }
  statement += this.getFooter(totalAmount);
  return statement;
}
```

Refactoring is usually iterative

```
public statement(): string {
  // generate header
  let result = "Rental Record for " + this.getName() + ":\n";
  for (const rental of this.rentals) {
    let rentalAmount = 0;
    // determine amounts for rental movie rented
    switch (rental.getMovie().getPriceCode()) {
      case Movie.REGULAR:
        rentalAmount += 2;
        if (rental.getDaysRented() > 2) {
          rentalAmount += (rental.getDaysRented() - 2) * 1.5;
        }
        break;
      case Movie.NEW_RELEASE:
        rentalAmount += rental.getDaysRented() * 3;
        break;
    }
    // add frequent renter points
    frequentRenterPoints++;
    // add bonus for a multi-day new release rental
    if ((rental.getMovie().getPriceCode() === Movie.NEW_RELEASE) && rental.getDaysRented() > 1) {
      frequentRenterPoints++;
    }
    // show figures for this rental
    result += "\t" + rental.getMovie().getTitle() + "\t" + rentalAmount.toFixed(2) + "\n";
    totalAmount += rentalAmount;
  }
  // generate footer
  result += "Amount owed is " + totalAmount.toFixed(2) + ".\n";
  result += "You earned " + frequentRenterPoints.toFixed(0) + " frequent renter points.";

  return result;
}
```

```
private simplifiedStatement(): string {
  let totalAmount = 0;
  let statement = this.getHeader();
  for (const rental of this.rentals) {
    const rentalAmount = this.getRentalAmount(rental);
    this.addRenterPoints(rental);
    this.addBonuses(rental);
    statement +=
      this.generateStatementLine(rental, rentalAmount);
    totalAmount += rentalAmount;
  }
  statement += this.getFooter(totalAmount);
  return statement;
}
```

```
private simplifiedStatement(): string {
  let statement = this.getHeader();
  statement += this.getStatementRows();
  const totalAmount = this.getTotalAmount();
  statement += this.getFooter(totalAmount);
  return statement;
}
```

Switch on type

A conditional varying behaviour based on an object's type:

Class Bird:

```
public getSpeed(obj:BirdContainer ): number {
   switch (obj.kind) {
      case Bird.EUROPEAN:
         return this.getBaseSpeed();
      case Bird.AFRICAN:
         return this.getBaseSpeed() - this.getLoad() * this.coconutCount;
      case Bird.NORWEGIAN_BLUE:
         return (this.isNailed) ? 0 : this.getBaseSpeed();
}
throw new Error("Unknown kind of bird");
```

Refactoring: Replace conditional with polymorphism

https://refactoring.guru/replace-conditional-with-polymorphism

Bird class

```
public getSpeed(obj:any ): number {
   switch (obj.kind) {
      case Bird.EUROPEAN:
         return this.getBaseSpeed();
      case Bird.AFRICAN:
         return this.getBaseSpeed() - this.getLoad() * this.coconutCount;
      case Bird.NORWEGIAN_BLUE:
         return (this.isNailed) ? 0 : this.getBaseSpeed();
}
throw new Error("Unknown kind of bird");
```

Instantiate the subtype

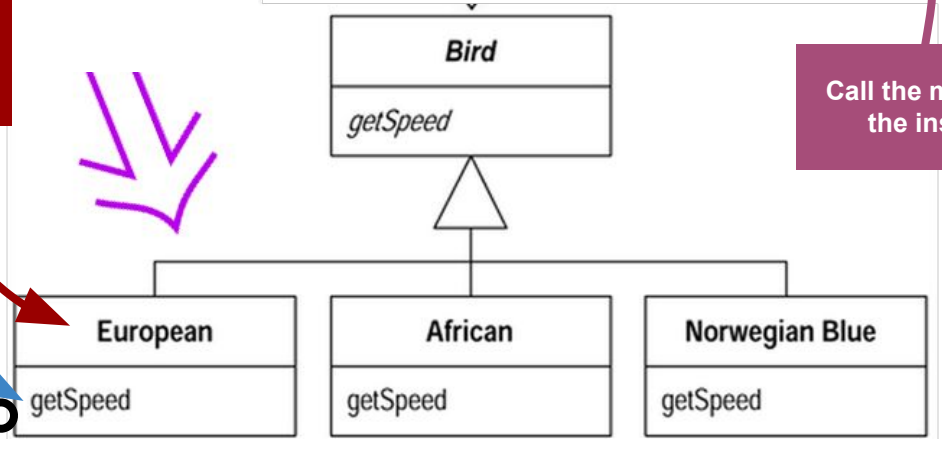
```
private calcProperties(): void {
   const bird: Bird = new African();
   const speed = bird.getSpeed();
   // … use speed
}
```

Each case gets its own class

Each class gets its own version of the method

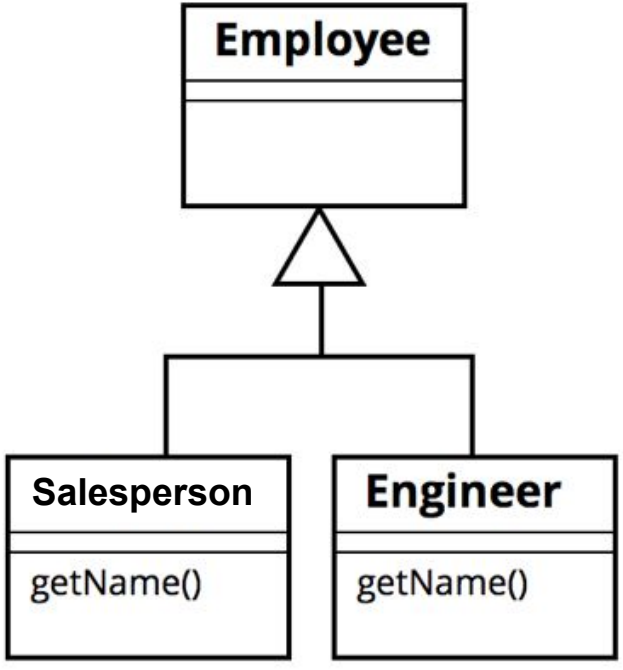
Each method gets the contents of the case body

Call the method on the instance




```
return getBaseSpeed();
```

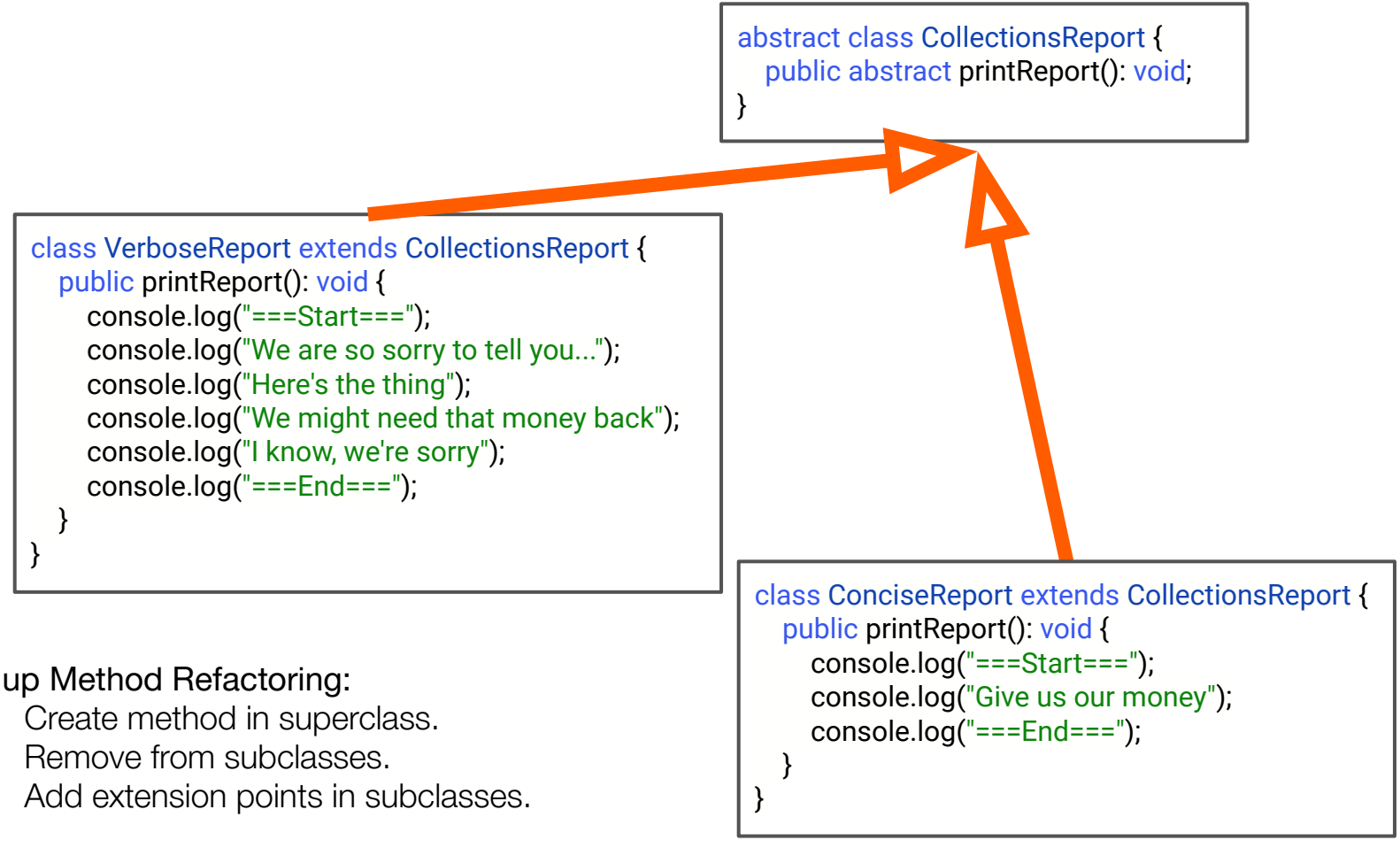
Identical methods in two classes



Pull up Method Refactoring:

1) Create method in superclass.
2) Remove from subclasses.

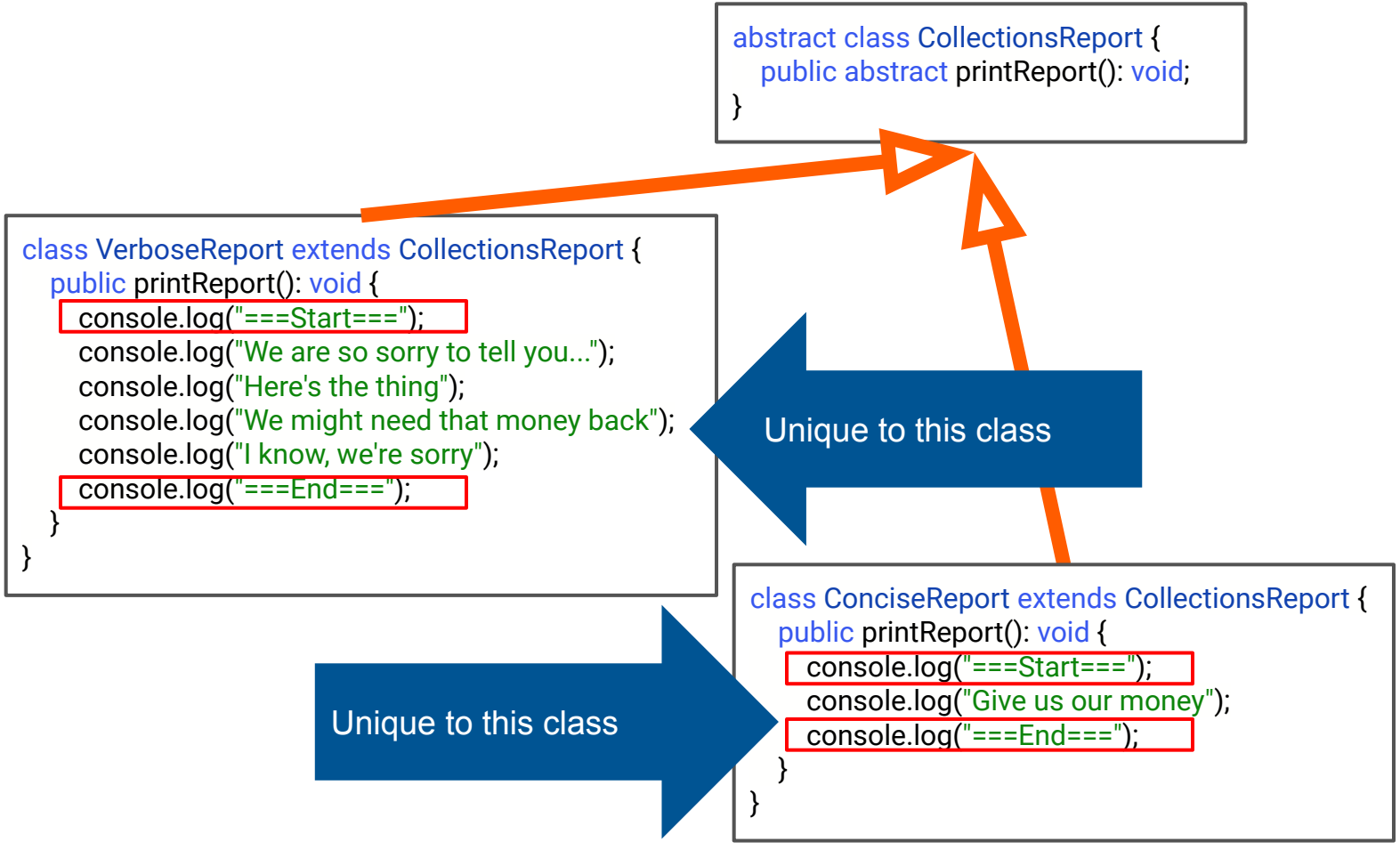
Almost identical methods in two classes



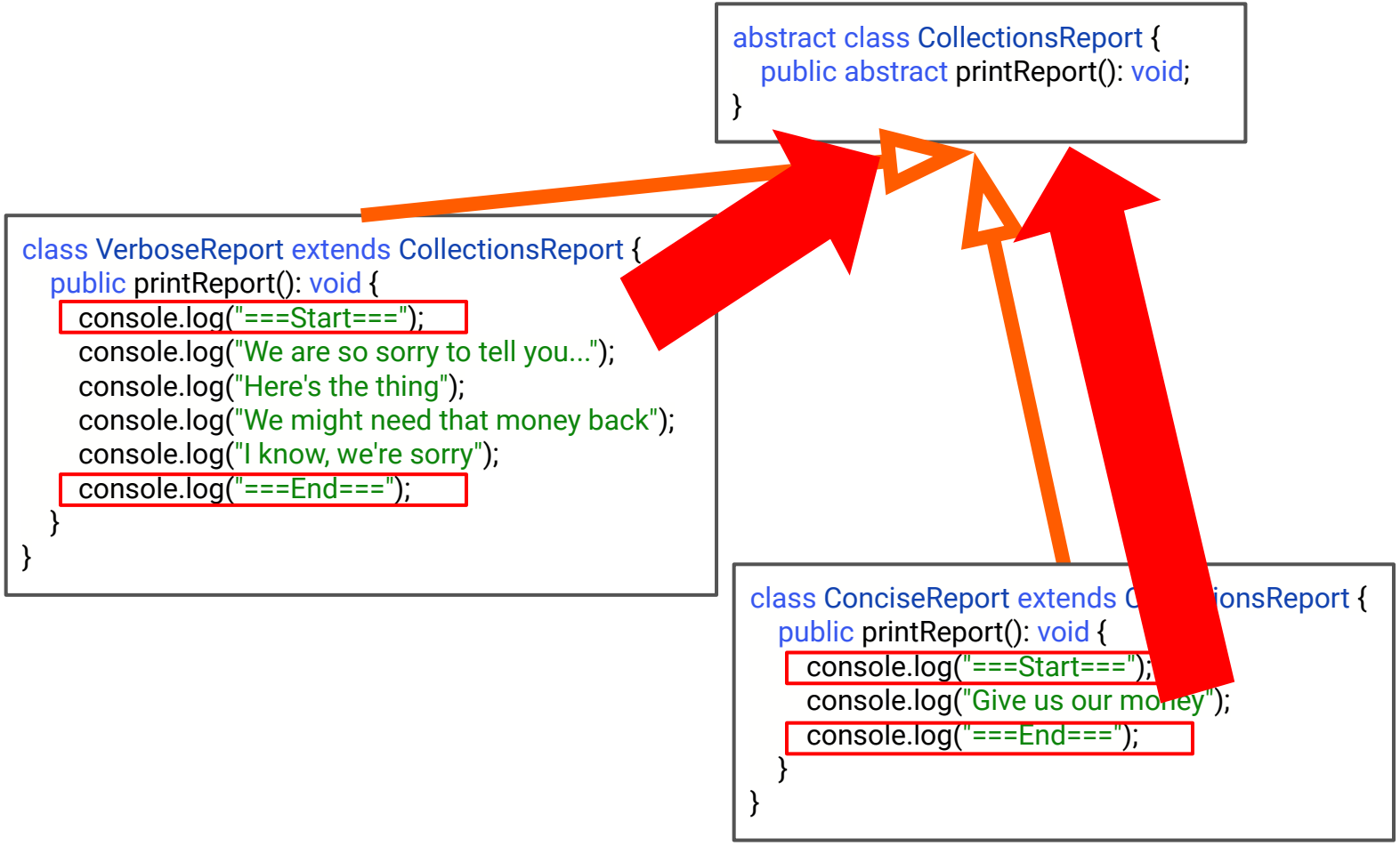
Pull up Method Refactoring:

1) Create method in superclass.
2) Remove from subclasses.
3) Add extension points in subclasses.

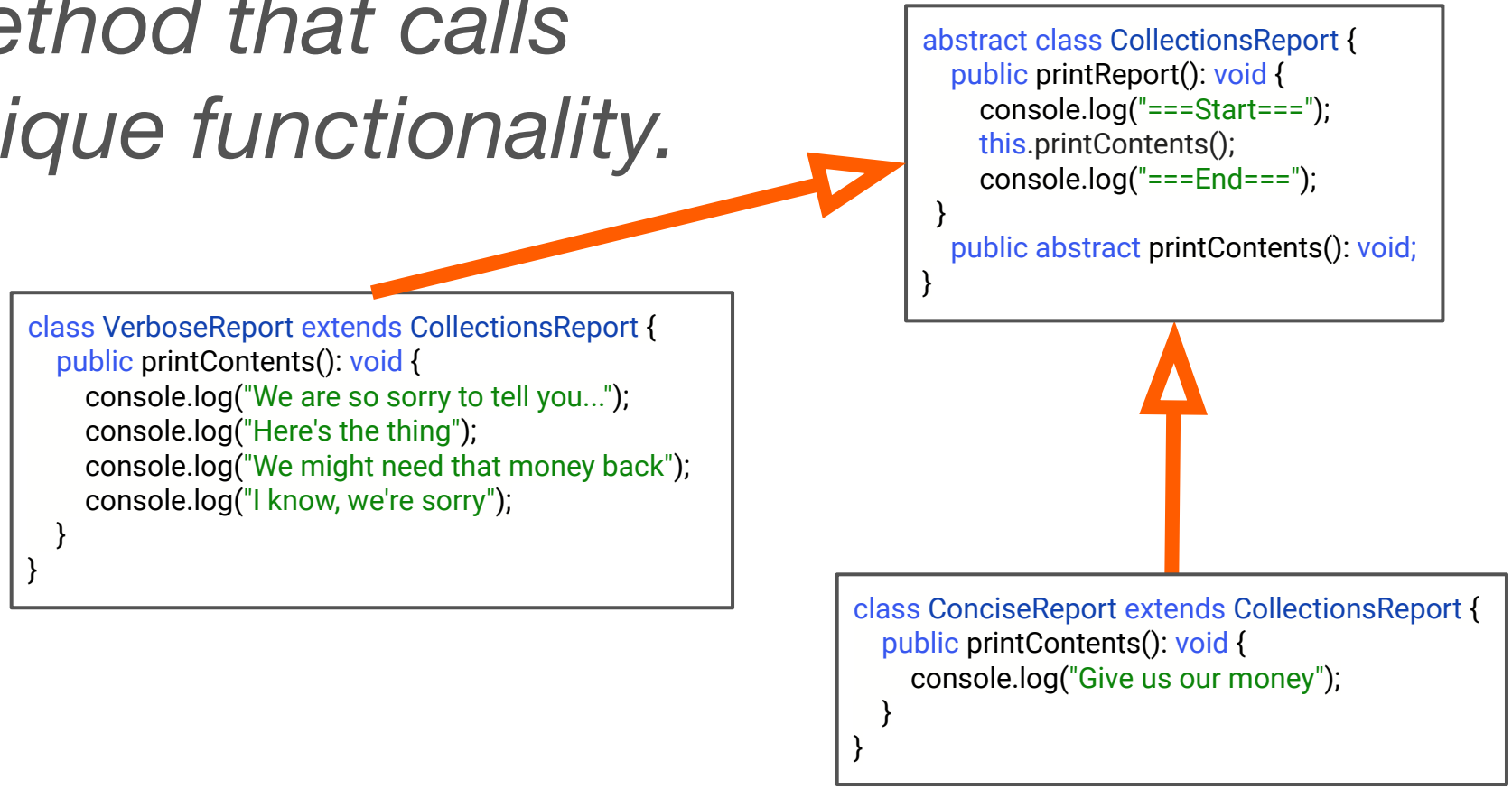
1) Identify what is common and unique



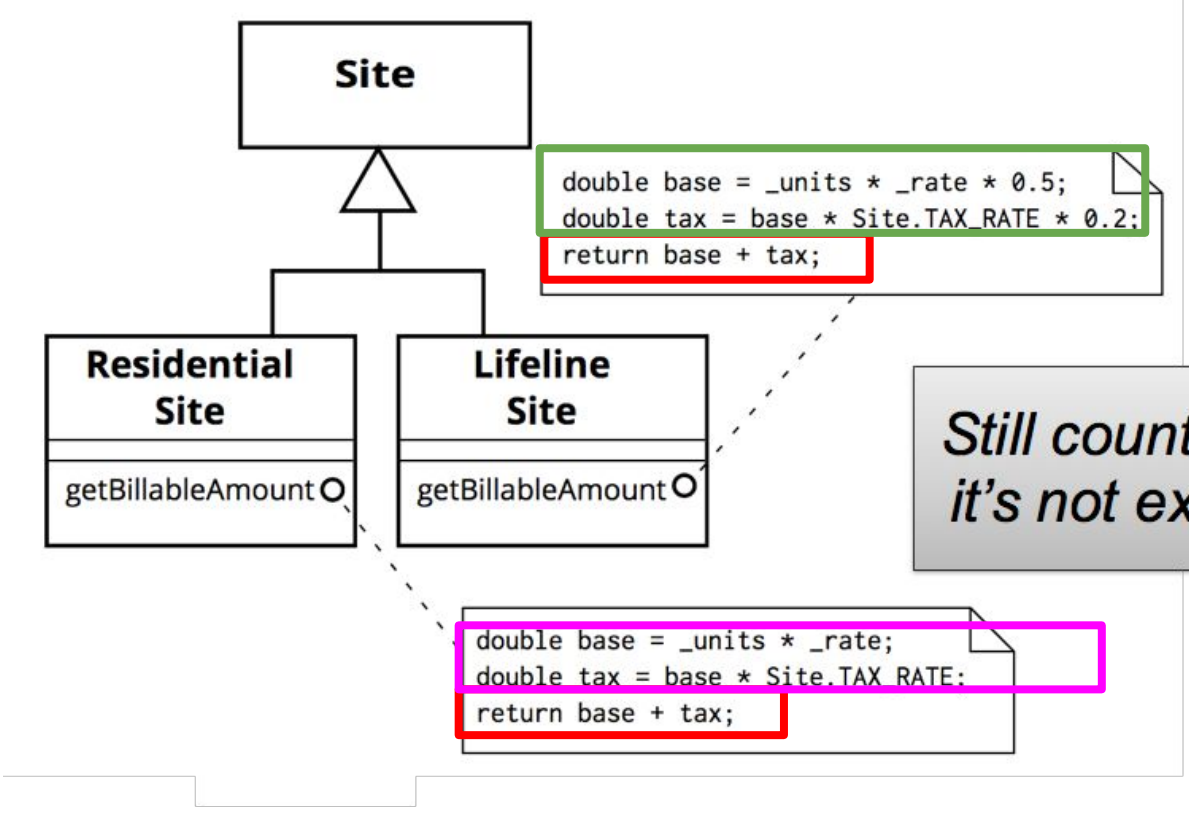
2) Pull up duplicate behaviour



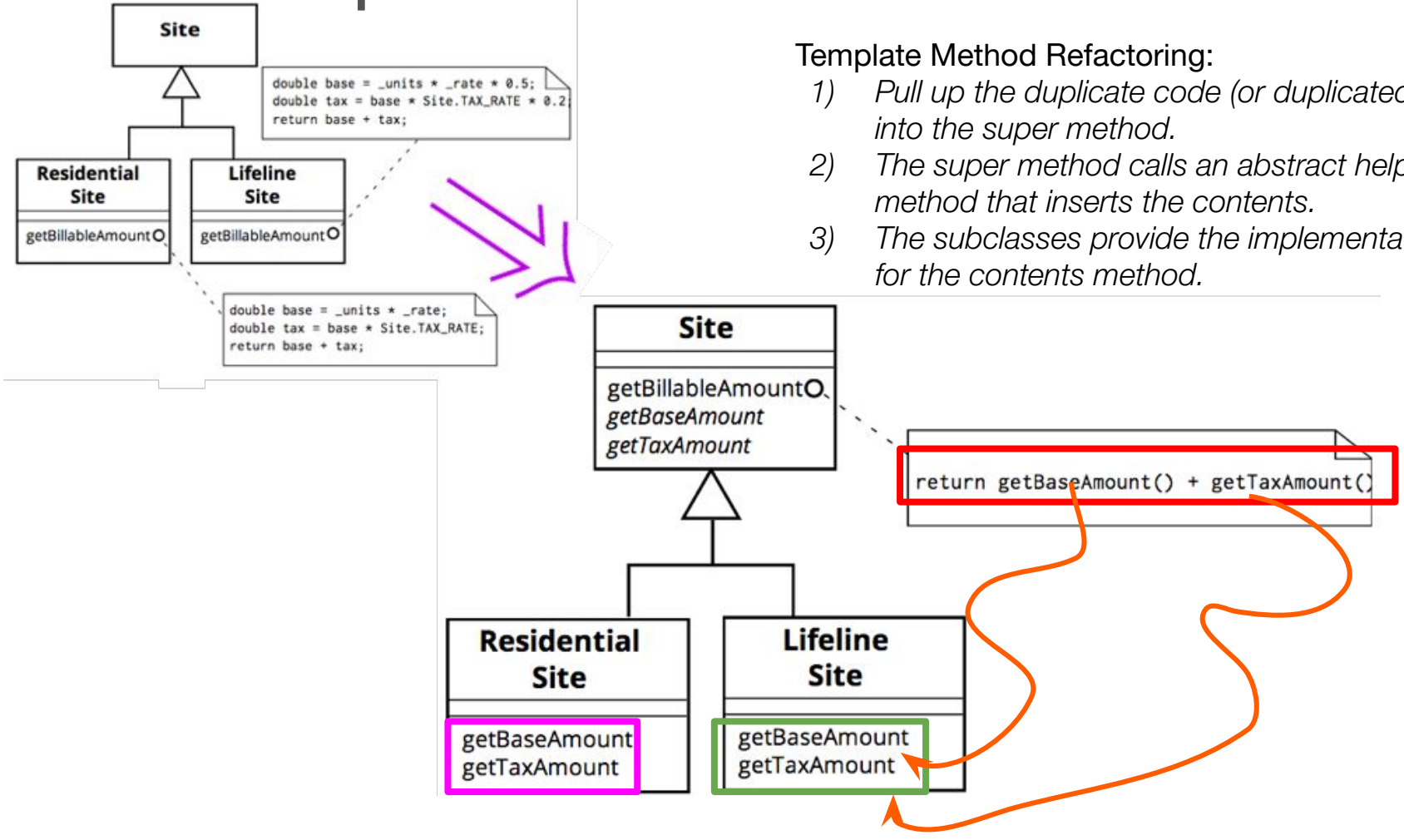
3) Make an abstract method that calls unique functionality.



Almost Duplicate Code (Again)



Create Template Method



Template Method Refactoring:

1) *Pull up the duplicate code (or duplicated logic) into the super method.*
2) *The super method calls an abstract helper method that inserts the contents.*
3) *The subclasses provide the implementations for the contents method.*

Feature Envy (Behaviour in Wrong Place)

When you find yourself wishing you had access to lots of fields from another class, the method you're implementing should probably live in that class.

Law of Demeter: Too many dereferences mean the method is probably in the wrong spot.

```
class Cat {
    private fido = new Dog();
    public chaseDog() {
        this.fido.tail.wag();
        this.fido.mouth.smile();
        // ... etc
    }
}
```

Why is **Cat** telling **Dog** how to behave? Objects should define their own behaviour.

```
this.db.students[sNum].courses['cs310'].getC2().push();
```

Experiential Code Smell Symptoms (Issues you notice while changing code)

The issues that we have seen up to this point are all *static*, they can be readily identified just by looking at the code.

In reality, code smells are identified *experientially*: as you try to *change* or understand code.

Two experiential symptoms that point to more elusive (and problematic) code smells are divergent changes and shotgun surgery.

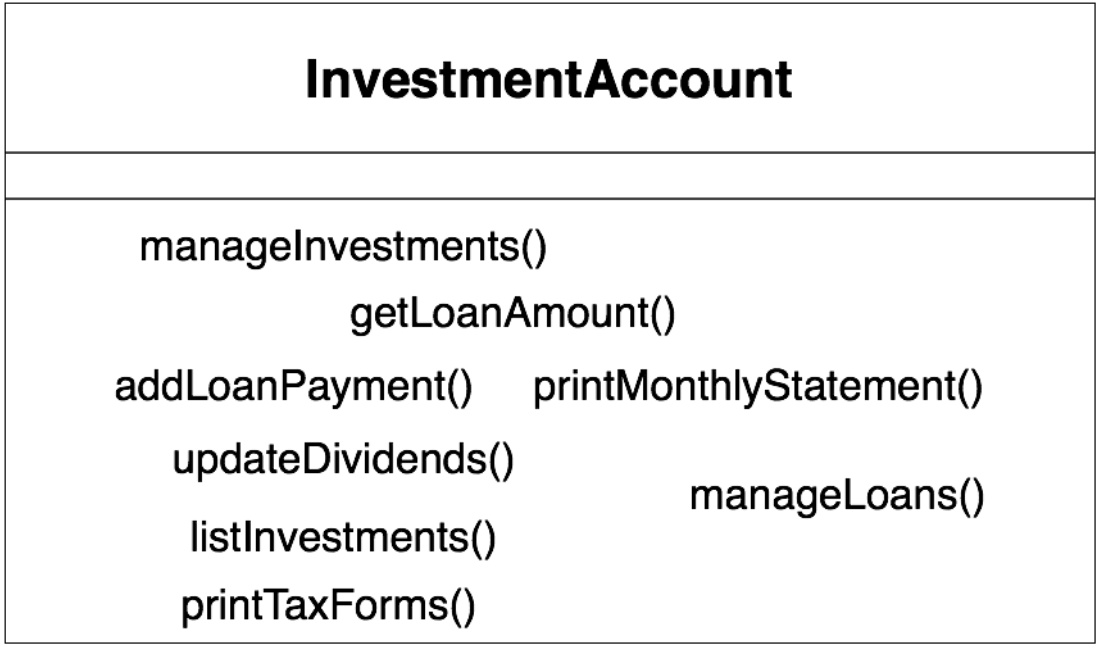
These smells often highlight the need for a new class to abstract the tangled or scattered design elements.

a) You have to make a change in more than one location.
b) *Changes (you can tell) will result in development collisions.*
c) *The code is too obfuscated to be clearly understood.*

Divergent Changes

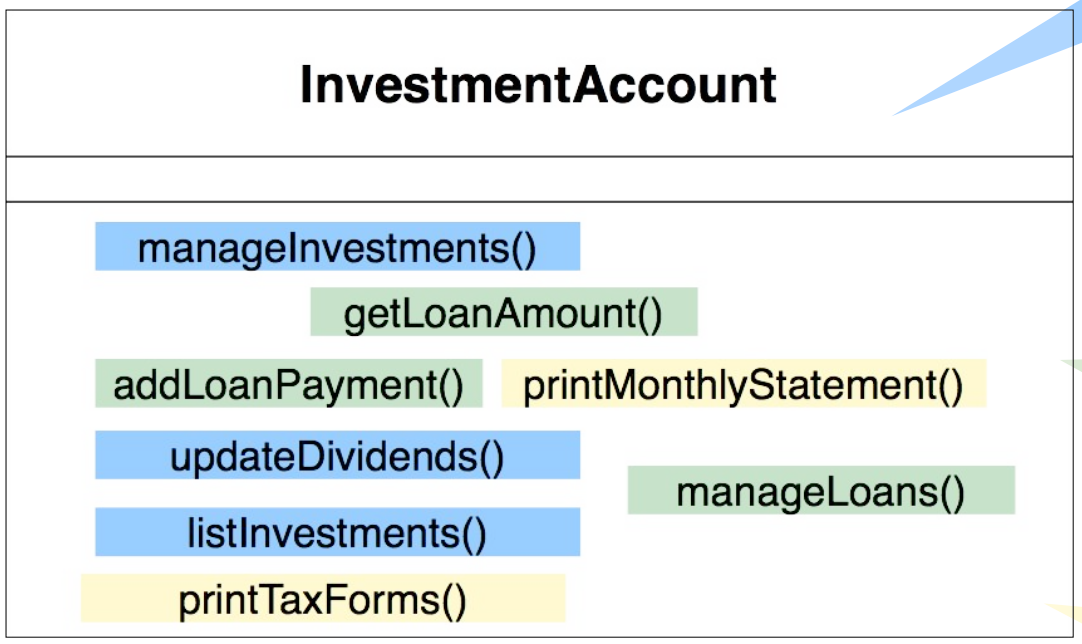
Divergent changes occur when one class is commonly changed in different ways for different reasons.

Any change to handle a variation should change only a single class, and that class should capture the unique variation.



If you look at a class and say, "Well, I will have to **change these three methods every time** I get a new database connection; I have to change **these four methods every time** there is a new financial instrument," you likely have a situation in which **two classes are better than one**. That way **each object is changed only as a result of one kind of change**. Of course, you often discover this only after you've added a few databases or financial instruments.

Divergent Changes



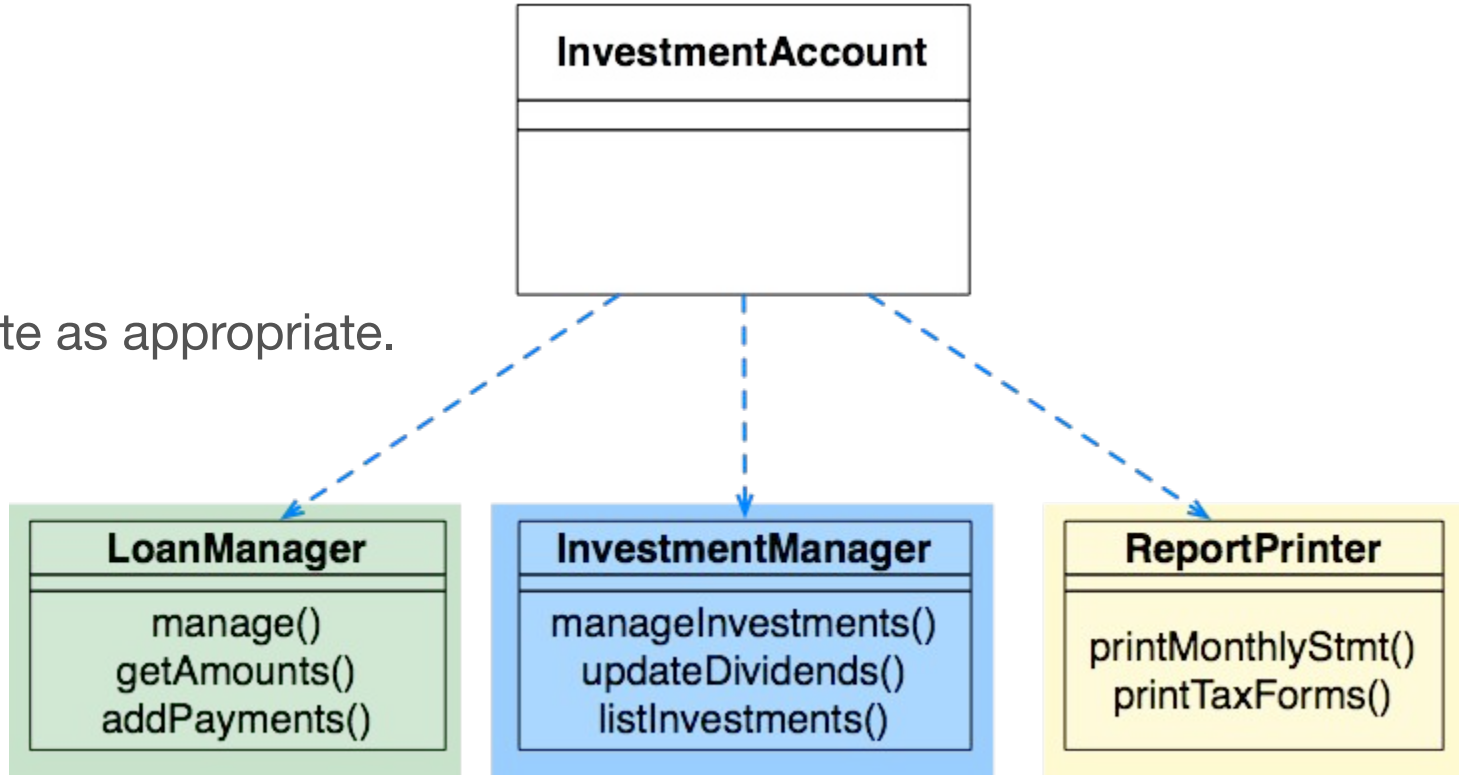
Will need to change whenever the investments implementation is changed.

Will need to change whenever the loans implementation is changed.

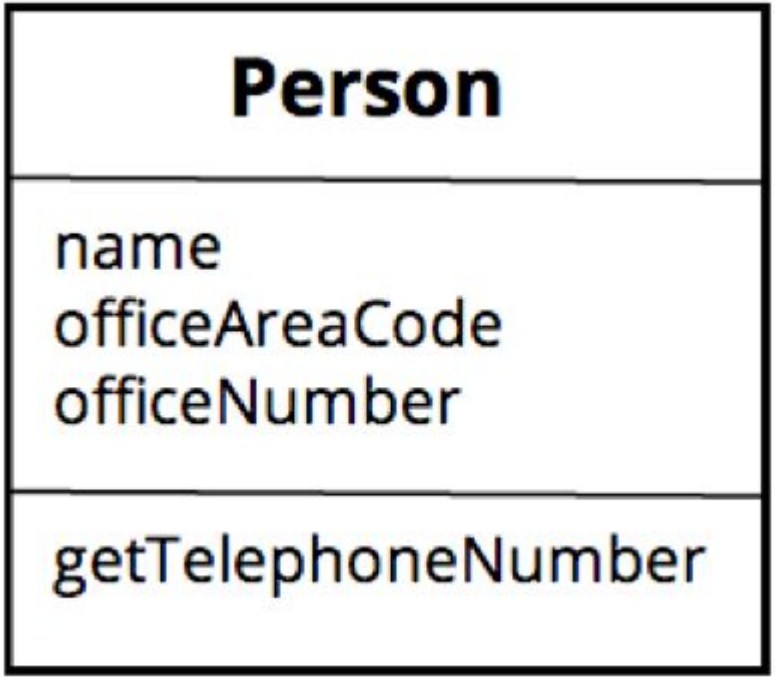
Will need to change every time the printing implementation is changed.

Extract Behaviours Into Their Own Classes

Delegate as appropriate.

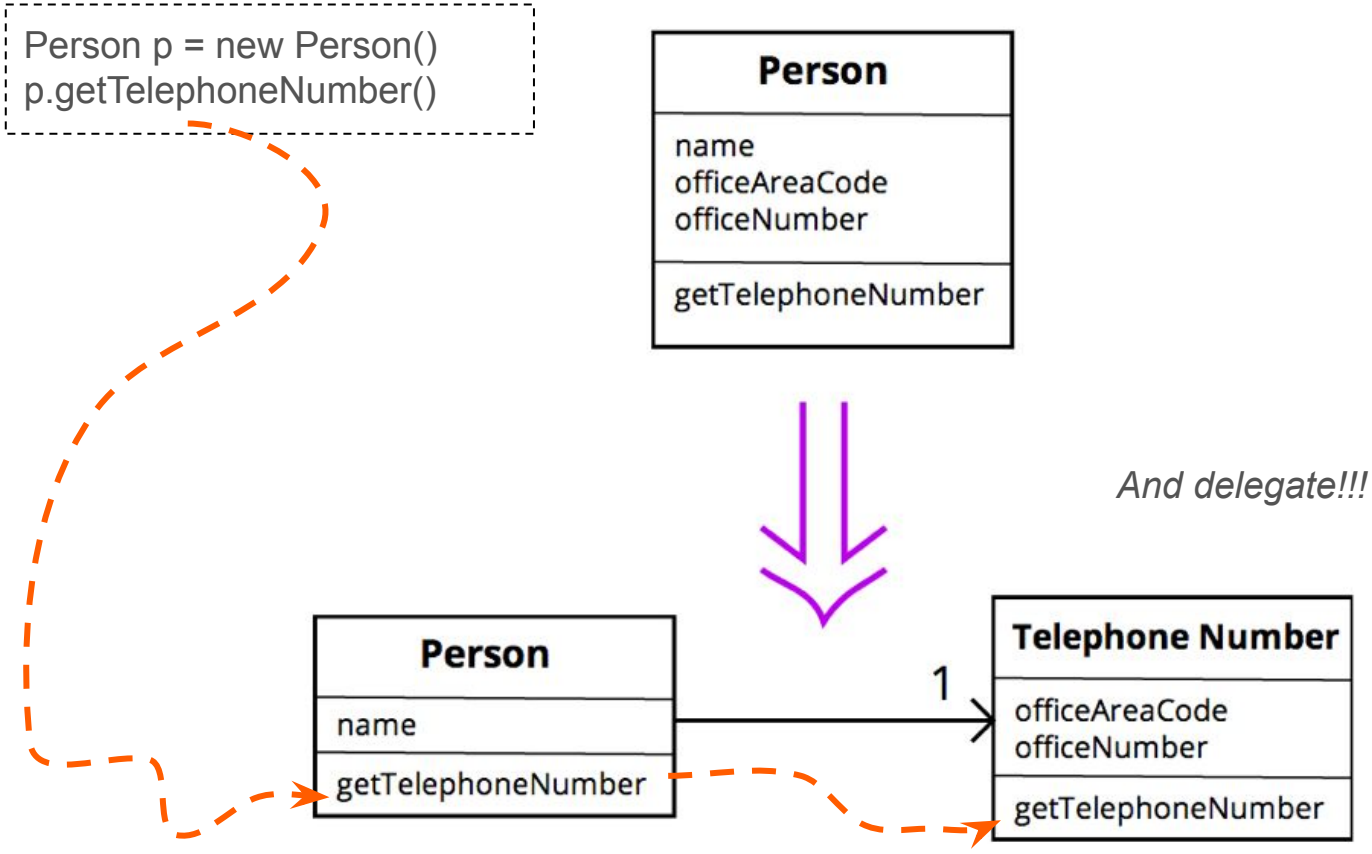


Symptom: **Divergent Changes**
Code smell: **One Class is Actually Two**



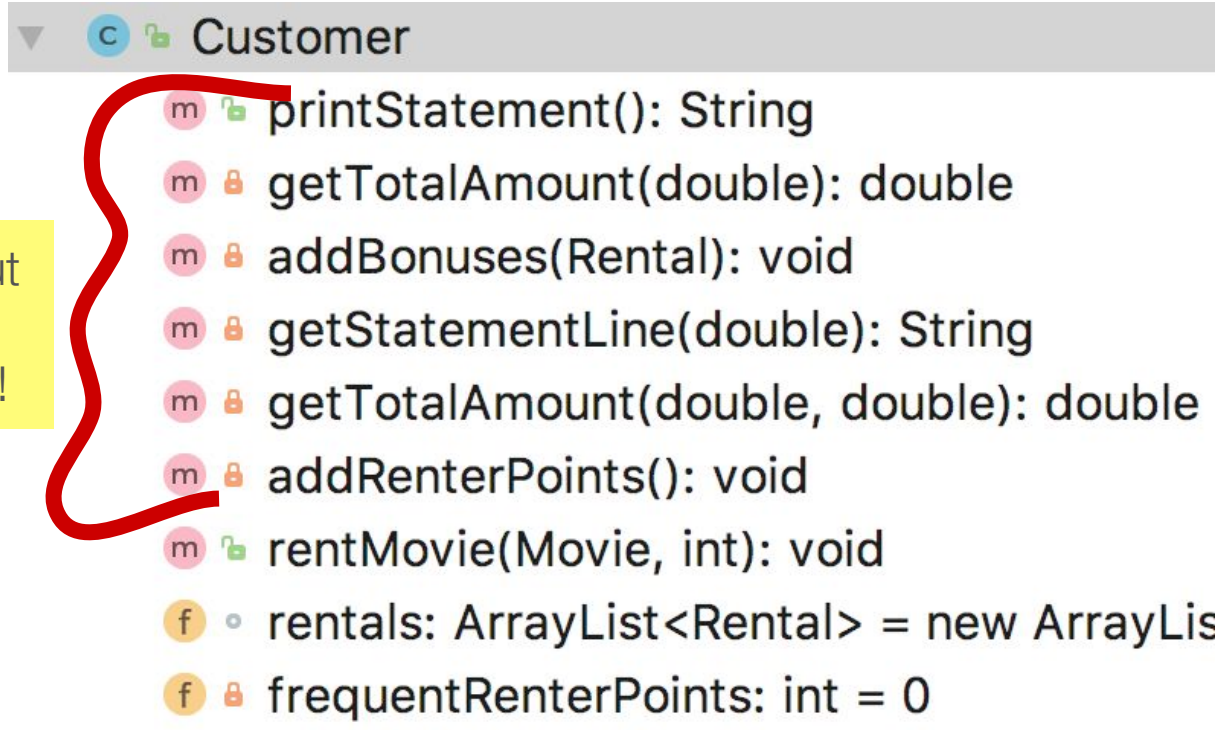
There is more than one field related to the telephone number, which gives it more weight. There might be additional methods supporting the phone number.

Code Smell: **One Class is Actually Two**
Refactoring: **Extract Class**

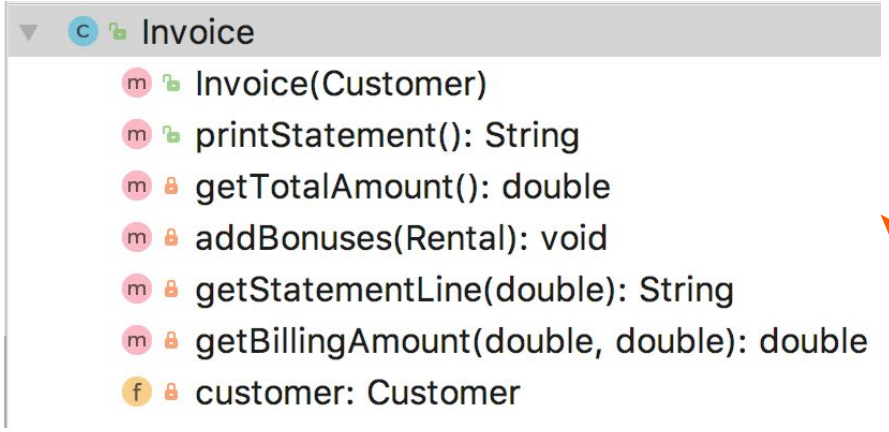


Code Smell: **One Class is Actually Two**
Refactoring: **Extract Class**

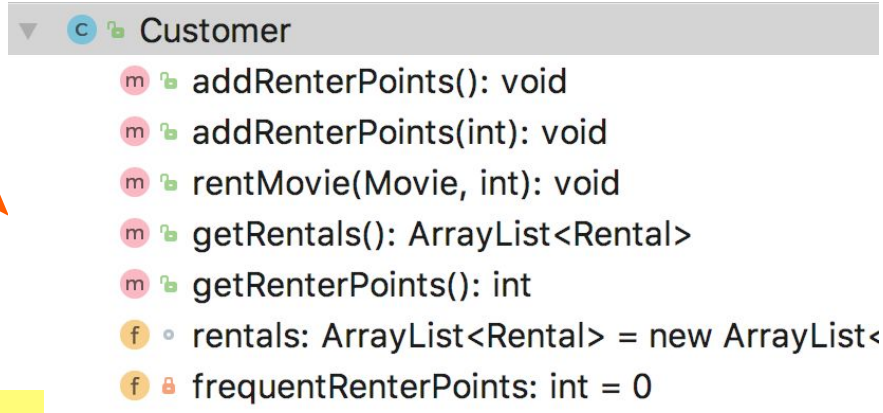
Seems to be about billing, not about being a customer!



Code Smell: **One Class is Actually Two**
Refactoring: **Extract Class**

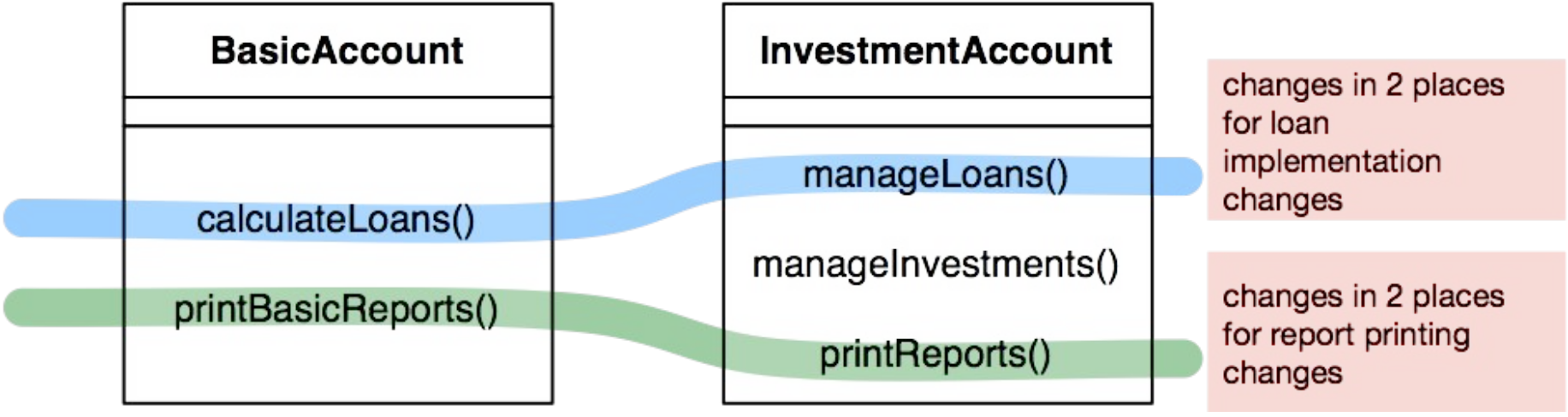



```
// customer.printStatment(); →
const invoice = new Invoice(customer);
invoice.printStatement();
```



The approach here is typically to make the class, then copy the methods over, then get them to compile, then alter the invocation so the new methods are being called, and run tests. Only then do you delete the old methods.

Shotgun Surgery



You whiff this when every time you make a kind of change you have to make lots of little changes in lots of different places. When changes are scattered they're hard to find and easy to miss.

Catchy synonym for **Delocalized** or **Scattered** change.

Divergent Changes and Shotgun Surgery

Divergent change is one class that suffers many kinds of changes, and **shotgun surgery** is one change that alters many classes. Either way you want to arrange things so that, ideally, there is a one-to-one link between common changes and classes.

[Fields, Harvie, & Fowler]

Underpinning Principle of Design

Makes an important statement on OO design:

That classes not *just* be considered in terms of their apparent responsibility, but must also be centered around their empirically observed relevance!

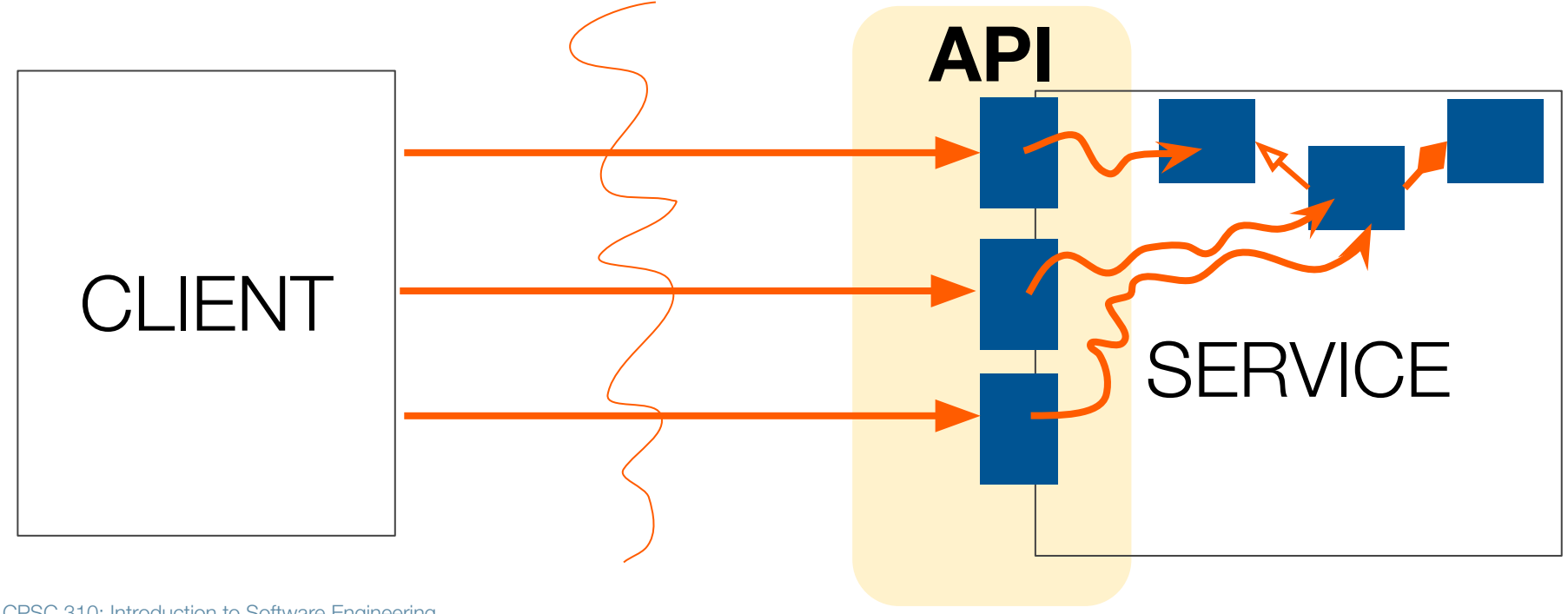
Systems should be decomposed to support evolution.

REST

Examinable Skills

- Spot non-RESTful qualities in a service: endpoints and semantics (using client side context to compute a “next” link, for instance).
- Structure a RESTful service given a description of the service
- Provide appropriate responses for each of the URIs in a RESTful service.
- Given a client-side user story, describe what the client side pseudo code would look like (what request it would send, would it be in a loop or an individual request, etc.).
- Relate RESTful qualities to the other design principles: the concept of localisation of reasoning and change.
- Discuss how RESTful services can use versioning to evolve their APIs.

Revisiting APIs



RESTful Design Principles

We were taking a journey through Object Oriented design, but we're going to briefly zoom out to inter-component design!

Underpinning Principles:

1) Localisation of reasoning.

2) Reduction in the propagation of change.

Why has REST become so pervasive?

- In *No Silver Bullet*, Fred Brooks states that: “The best way to address the complexity of software is to not build it at all.”
  - Several long-sought NSB solutions favour reuse:
    - As a means to reduce complexity.
    - As a means to improve productivity.
    - As a means to increase reliability.
    - As a means to encourage / enable reuse.

Reuse in practice

- New features are added by “dropping in” components:
  - Accounts for 53% of reuse at NASA. [Selby 2005]
- Domain-specific component markets:
  - Populated by carefully created reusable components.
  - This is harder than it sounds.
- Three main impediments:
  - High up-front cost. [Gaffney 1992, ICSE]
  - Architectural mismatch. [Garlan *et. al.* 1995, IEEE Software]
  - Library scaling problem. [Biggerstaff 1994, ICSR]

Up front costs

- Budgets are drawn up annually
  - Heavy emphasis on the current quarter
- Reusable software is:
  - ~Twice as expensive [Gaffney 1992, ICSE]
  - ~Three times as expensive [Brooks 1975]
- Requires careful forethought to determine what software *will* be reused and whether any savings outweigh extra costs
- What is the benefit to the customer?

Architectural Mismatch

- Even reusable code makes some assumptions about *how* it should be reused; these assumptions are often implicit
  - Explicit assumptions are often easy to identify:
    - Programming language
    - Libraries & frameworks
  - Implicit assumptions are harder to spot:
    - Topology assumptions
    - Protocols of use
- Implicit assumptions are often not documented because the original developer may not have considered them constraints.

Library Scaling

- Two extremes:
    - Large, feature-laden, components.
    - Small, simple, components.
- Large components are hard to adapt.
- Small components have complex cost/benefit implications.

Let's Look at an Example

CanaFind:

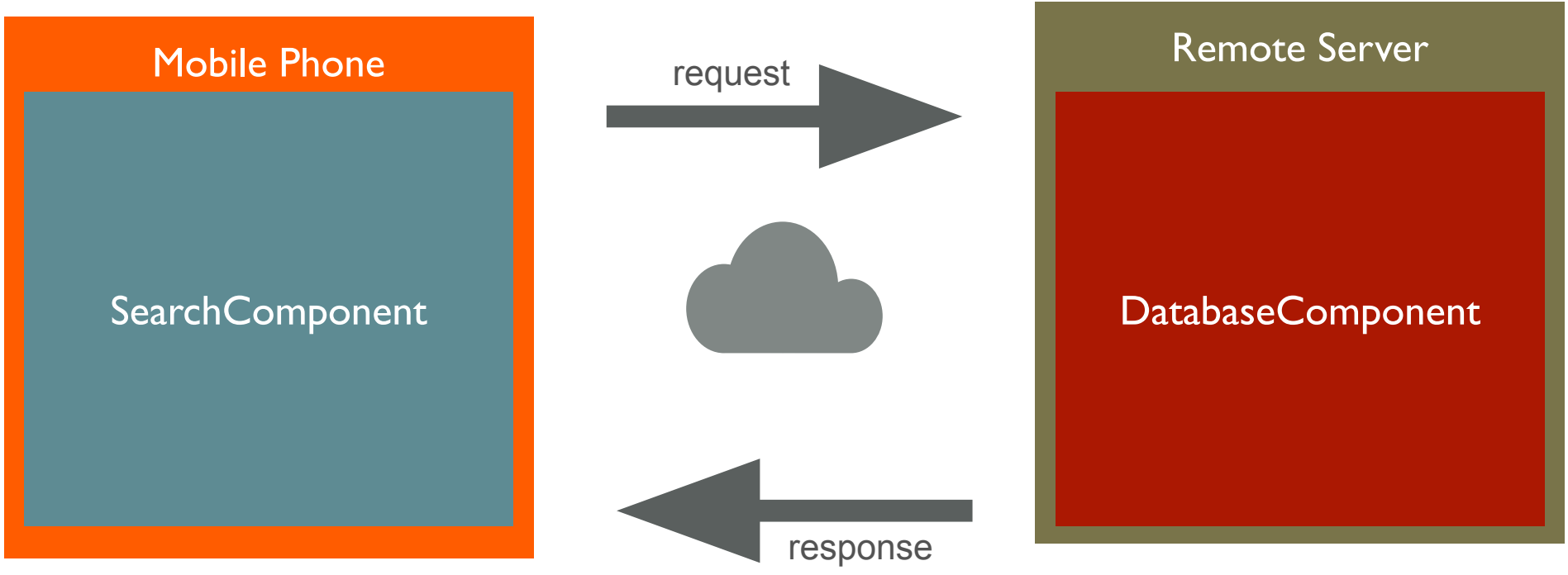
- **Finders:** As someone who found something in a public place, I would like to be able to upload a photo of that thing, so their owner can learn that it has been found.
- **Searcher:** As someone who lost something in a public space, I would like to be able to look at a location on a map and see whether anything was found there, so I can see if someone has found what I have lost.

(Yes, this assumes no one will move anything.)

(It's not a perfect system.)

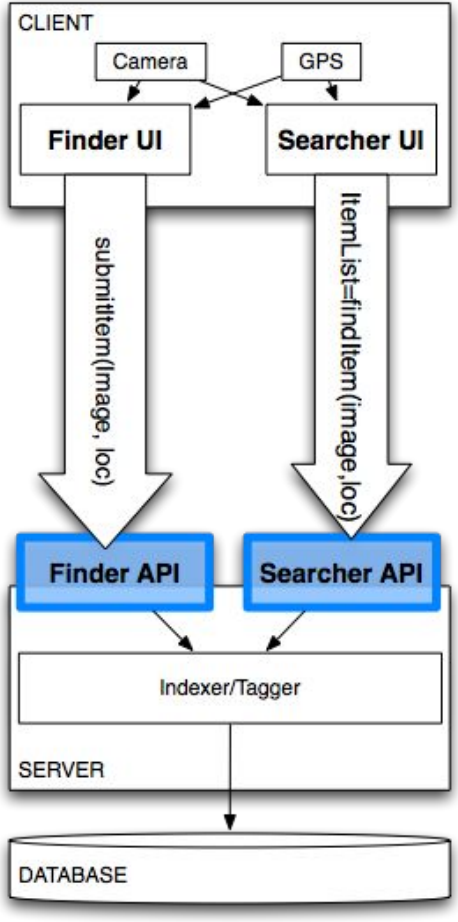
What is This All About?

RPC


Mobile Phone
SearchComponent
request
response
Remote Server
DatabaseComponent

An “RPC” Approach

Delocalization of changes!!!!
(Server changes propagate to client!)



- This approach involves **Remote Procedure Calls**, and a lot of integration between client and server.
- Note that the **calls** are specialised ‘**verbs**’ (**submitItem(...), findItem(...)**).
- **Pro**: Simple to program, behaves like local calls.
- **Con**: The protocol must be pre-defined. The client and the server must agree on methods/data types.
- This is the approach used in the Simple Object Access Protocol (SOAP) ...which we will not cover.
- RPC is covered in much more detail in CPSC 416.

Another Way: REST

- **RE**presentational
- **S**tate
- **T**ransfer

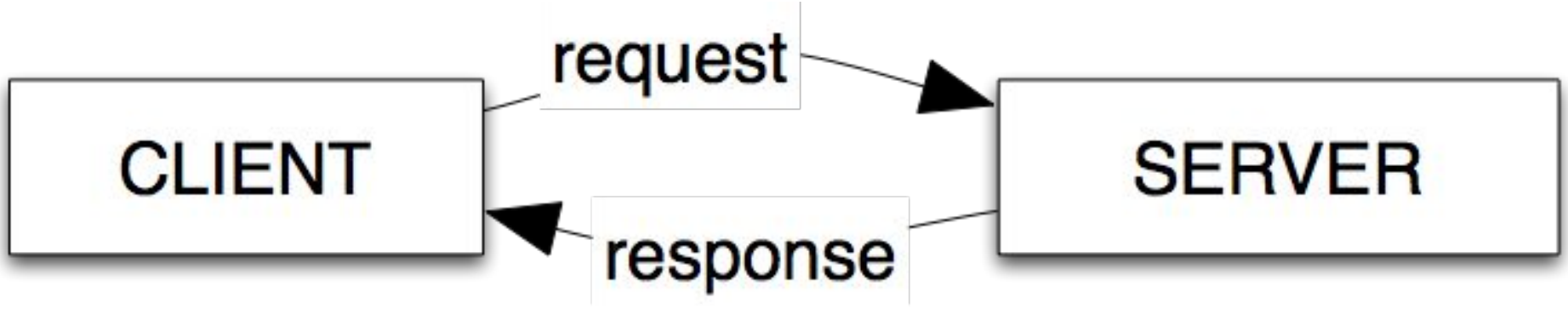
RESTful Interfaces

"… the web is an existence proof of a massively scalable **distributed system** that works really well, and we can take ideas from that to **build integrated systems** more easily." – Martin Fowler

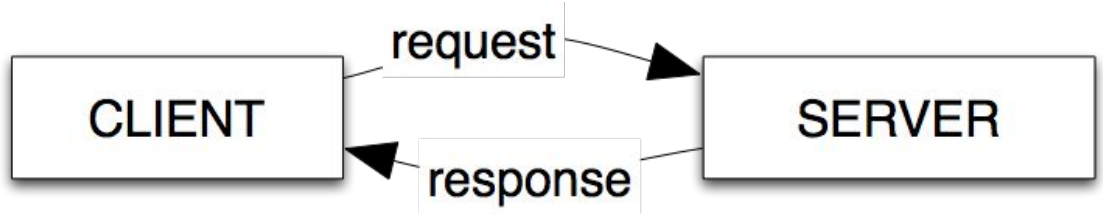
"Representational State Transfer is intended to evoke an image of how a **well-designed** Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (**state transitions**), resulting in the next page (representing **the next state** of the application) being transferred to the user and rendered for their use." - Roy Fielding

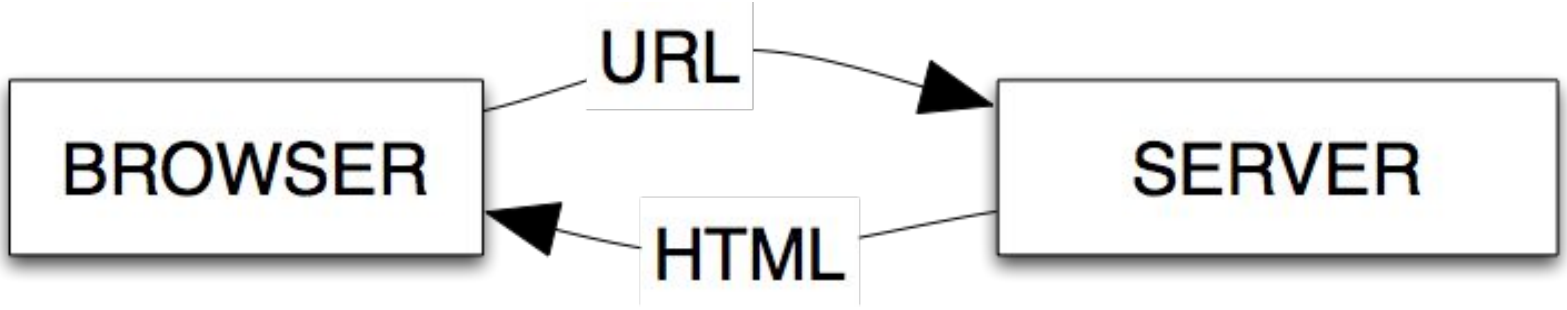
[http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven]
Roy Fielding interview: [https://www.youtube.com/watch?v=w5j2KwzzB-0]

REST: Basics

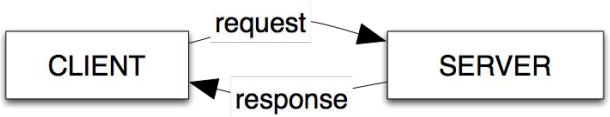


If Your Client is a Browser



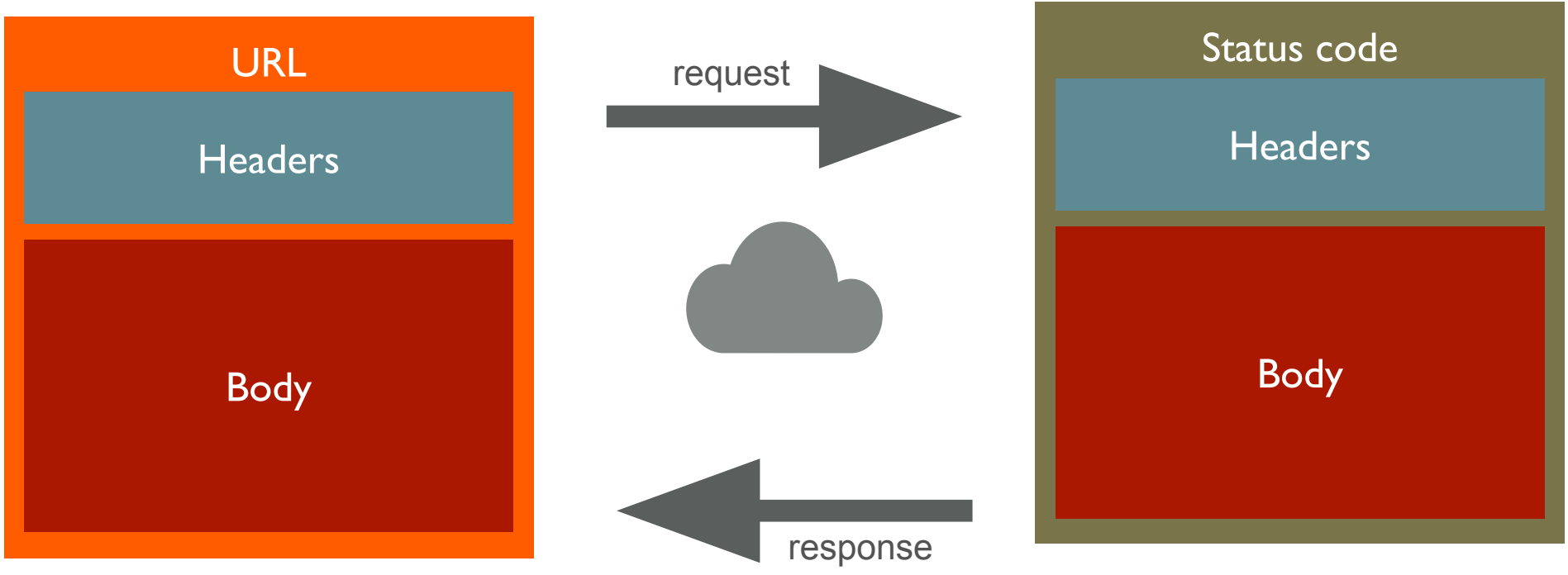


If Your Client is an Application



GET, PUT, POST, DELETE

HTTP


URL
Headers
Body
request
response
Status code
Headers
Body

REST: Nouns & Verbs

- **URIs** are the "nouns" of REST APIs:
  - Provide abstraction of resources and data.
  - e.g., what can be accessed:
    - cs310.students.cs.ubc.ca/student/grades/
    - cs310.students.cs.ubc.ca/admin/grades/
- **Verbs** are the connection "methods":
  - GET, PUT, POST, DEL (and PATCH, OPTIONS)
  - GET (retrieve), POST (create), PUT (update), DELETE (remove)
  - **Idempotency**: Will performing a VERB multiple times have the same result for server resources as performing it once?
    - Important in distributed systems.
    - **POST**: Is not idempotent; always makes new resources.
    - **Other Verbs**: Idempotent. Resources are consistent.

Nouns: Resources

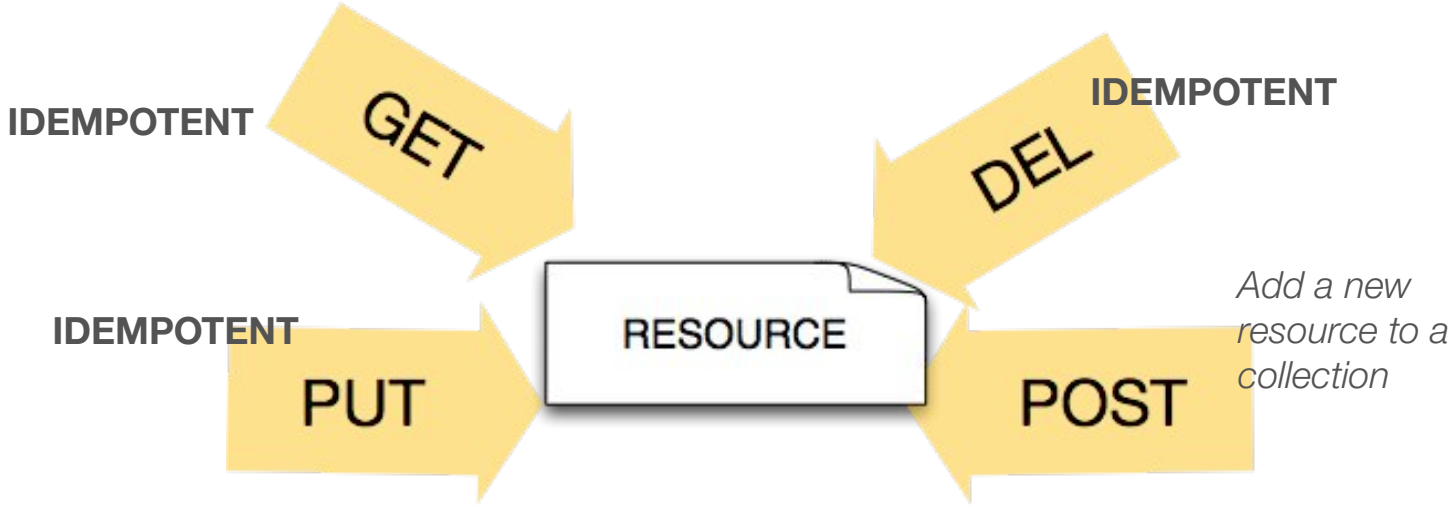
2 kinds of things:
- Items (individual)
- Collections (always plural)

www.piazza.com/courses/310/notes/145 << THIS IS THE URI.
www.piazza.com/getCourses/ <<NO NO NO (verb in URI)

- Resources are identified by “Uniform Resource Identifiers” (URIs)
- Examples:
  - Twitter resources include tweets, users, followers
  - Facebook resources include users, friends, posts
  - An email system’s resources might be contacts, and messages
- **URIs:** example:
  - http://twitter.com/users/{id}/tweets/{id}/
  - http://twitter.com/hashtags/{id}/tweets
  - http://twitter.com/hashtags/{id}/users/{id}/tweets
  - http://canafind.ca/locations/4/items/3

CanaFind resources might be:
- **locations**
- **items**

Verbs: GET/PUT/POST/DELETE



- Every resource must support (to some elegant extent) all four verbs.
- Examples:
  - A twitter client can get tweets, put/post tweets, delete tweets, etc.
  - An email client can get messages, get lists of users, post messages, delete messages, etc.

REST Design Constraints

- Apply principled design process to commonly occurring challenges facing online applications.
- Needed to support networked sets of loosely coupled services.
- This led to several constraints, addressing provided properties:
  - Constraint: Replication needed to support caching.
  - Constraint: Uniform interface allowed simple mechanisms.
  - Constraint: Statelessness required for reliability.
  - Constraint: Connectedness supports discoverability.

REST Design Principles

- **Resource-orientation**: Orienting designs around resources emphasis a separation of concerns between a resource and the actions a system might perform on the action. This commonly drives information hiding decisions in REST systems.
- **Uniform interface**: Having a consistent, predictable, and simple mechanism for marshalling requests and responses allows for clients and servers to independently evolve (for instance using different programming langs).
- **Statelessness**: All client requests need to contain all of the information needed to process a response. This is a fundamental decision that allows server resources to scale.
- **Self-descriptive messages**: Messages, especially responses, should contain information to help clients reason about how to process a response. This is most commonly used to enable follow-up actions for a resource. This often enables connectedness, where clients do not need to guess about an API.

POST `/repos/{owner}/{repo}/issues/{issue_number}/comments`

Input

Name	Type	Description
`body`	`string`	**Required**. The contents of the comment.

```
{
  "body": "Me too"
}
```

POST /repos/{owner}/{repo}/issues/{issue_number}/comments

Response

```
Status: 201 Created
Location: https://api.github.com/repos/octocat/Hello-World/issues/comments/1

{
```

POST /repos/{owner}/{repo}/issues/{issue_number}/comments

Response

```
Status: 201 Created
Location: https://api.github.com/repos/octocat/Hello-World/issues/comments/1
```


```
{
  "id": 1,
  "node_id": "MDEyOklzc3VlQ29tbWVudDE=",
  "url": "https://api.github.com/repos/octocat/Hello-World/issues/comments/1",
  "html_url": "https://github.com/octocat/Hello-World/issues/1347#issuecomment-1",
  "body": "Me too",
  "user": {
    "login": "octocat",
    "id": 1,
    "node_id": "MDQ6VXNlcjE=",
    "avatar_url": "https://github.com/images/error/octocat_happy.gif",
    "gravatar_id": "",
    "url": "https://api.github.com/users/octocat",
    "html_url": "https://github.com/octocat",
    "followers_url": "https://api.github.com/users/octocat/followers",
    "following_url": "https://api.github.com/users/octocat/following{/other_user}",
    "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
```

DEL /repos/{owner}/{repo}/pulls/{number}/reviews/{review_id}


```
    "repos_url": "https://api.github.com/users/octocat/repos",
    "events_url": "https://api.github.com/users/octocat/events{/privacy}",
    "received_events_url": "https://api.github.com/users/octocat/received_events",
    "type": "User",
    "site_admin": false
  },
  "body": "Here is the body for the review.",
  "commit_id": "ecdd80bb57125d7ba9641ffaa4d7d2c19d3f3091",
  "state": "PENDING",
  "html_url": "https://github.com/octocat/Hello-World/pull/12#pullrequestreview-80",
  "pull_request_url": "https://api.github.com/repos/octocat/Hello-World/pulls/12",
  "_links": {
    "html": {
      "href": "https://github.com/octocat/Hello-World/pull/12#pullrequestreview-80"
    },
    "pull_request": {
      "href": "https://api.github.com/repos/octocat/Hello-World/pulls/12"
```

Design Flexibility

- URIs define resources, but there are many ways to pass additional data to the server:
  - PUT: /segments/{id}/starred
    - What *user* is applying the star?
  - DEL: /repos/../{repo}/pulls/../{review_id}
    - Is the *user* performing the action permitted to delete this?
  - Potential options:
    - Headers, explicit parameters, request bodies.

Developer Documentation

PUT `/segments/{id}/starred`

Parameters

id required Long, in path	The identifier of the segment to star.
starred required Boolean, in form	If true, star the segment; if false, unstar the segment.

List push subscriptions

This request is used to retrieve the summary representations of the push subscriptions in place for the current application.

Parameters

client_id:	**integer** *required* application's ID, obtained during registration
client_secret:	**string** *required* application's secret, obtained during registration

Attributes

id:	**integer**
object_type:	**string** object for which to get events, eg. 'activity'
aspect_type:	**string** aspect for which to get events, eg. 'create'
callback_url:	**string**
created_at:	**time string**
updated_at:	**time string**

DEFINITION

```
GET https://api.strava.com/api/v3/push_subscriptions
```

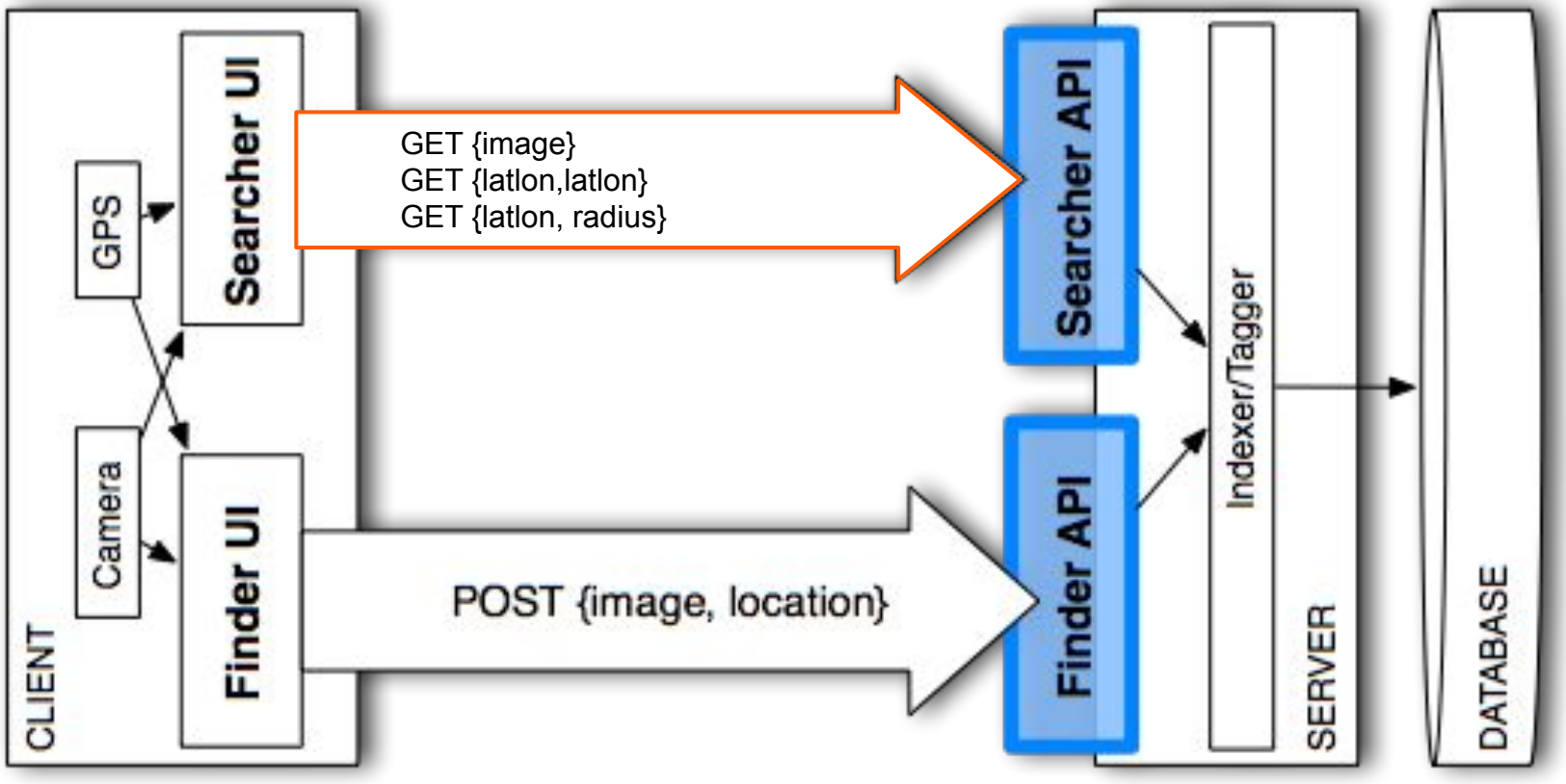
EXAMPLE REQUEST

```
$ curl -G https://api.strava.com/api/v3/push_subscriptions \
    -d client_id=5 \
    -d client_secret=7b2946535949ae70f015d696d8ac602830ece412
```

EXAMPLE RESPONSES

```
[
  {
    "id": 1,
    "object_type": "activity",
    "aspect_type": "create",
    "callback_url": "http://you.com/callback/",
    "created_at": "2015-04-29T18:11:09.400558047-07:00",
    "updated_at": "2015-04-29T18:11:09.400558047-07:00"
  }
]
```

RESTful CanaFind



Enabling API Evolutionary Robustness

- Imagine the following response object:
  - {name: ‘cpsc autobot’}
- But in the future you could imagine that you need to differentiate first and last names; one possible schema:
  - {first: ‘cpsc’, last: ‘autobot’, name: ‘cpsc autobot’}
- REST resources as much as possible strive for backwards compatibility since frontend and backends are loosely coupled and may not be simultaneously updated.

Versioning

- APIs are both assets and liabilities.
- Why would we want to version APIs?

Versioning

- Path: `GET /2017-01-01/customers/{id}/history`
  - (Encode date of API version)
- Path: `GET /2/customers/{id}/history`
  - (Encode version in request)
- Path: `GET /v2/customers/{id}/history`
  - (Encode version in request)
- Query: `GET /customers/{id}/history/?v=2`
  - (Encode version in param)
- Specify the version in the header: `APIVersion: 2`

RESTful Takeaways

- Resources are **nouns**, not verbs.
- All resources must elegantly handle:
    - GET/PUT/POST/DEL
    - Make sure GET/PUT/DEL are idempotent.
- All behavioural **state must remain client-side**.
- Clients should not have to remember the internal server structure — **send back links** in your responses to help clients **discover** the other resource location.

DevOps

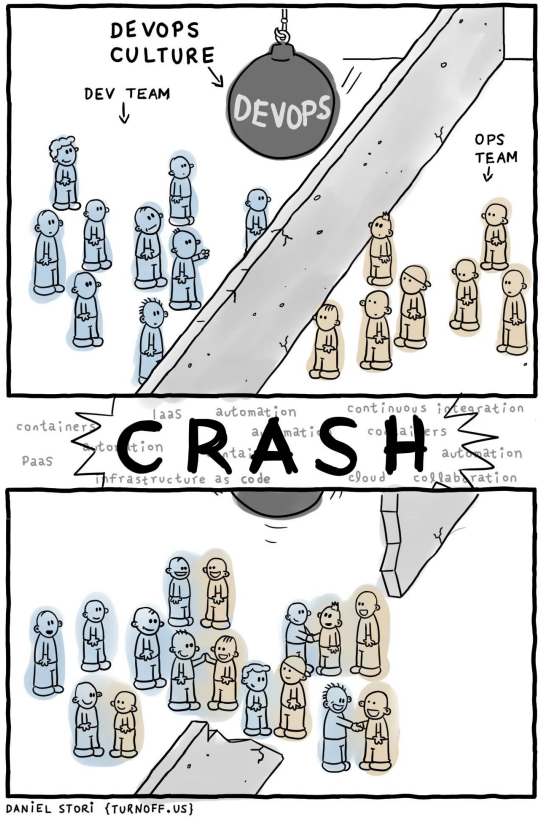
Examinable skills

- Explain the rationale behind the DevOps movement.
- Identify key ideas and common practices in DevOps.
- Explain the value behind testing in production.
- Explain how A/B testing and canary releases work.
- Explain the role of automation and tools in DevOps.
- Understand the purpose Docker and Kubernetes fulfill in the DevOps environment.


WORKED FINE IN DEV
OPS PROBLEM NOW
memegenerator.net

How did we get here

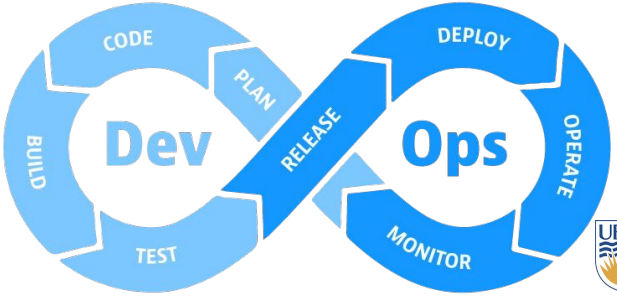
- Historically, the people who *develop the software*, and the people who *operate software* were not on the same teams.
- This divide has numerous downsides:
  - Poor communication.
  - Slow resolution of production issues.
  - Lengthened feedback loop.
  - Higher deployment cost/time.
- **DevOps** attempts to unify these teams.
- **Automation** crucial to enable almost all aspects of the modern DevOps toolbox.



DevOps Key Goals

- Better coordinate between developers and operations (*collaborative*).
- Key goal: Reduce friction in bringing changes from development into production (*efficiency*).
- Consider the entire tool chain into production (*holistic*).
- Documentation and versioning of all dependencies and configurations ("*configuration as code*").
- Heavy *automation*, e.g., continuous delivery, monitoring.
- Small iterations, incremental and continuous releases.

Warning: Buzz word!



Common release problems

Examples:

- Missing dependencies.
- Different compiler versions or library versions.
- Different local utilities (e.g., unix grep vs mac grep).
- Database problems (config, auth, storage).
- OS differences.
- Too slow in real settings.
- Difficult to rollback changes.
- Source code from many different repositories.
- Obscure hardware? Cloud? Enough memory?

DevOps divide

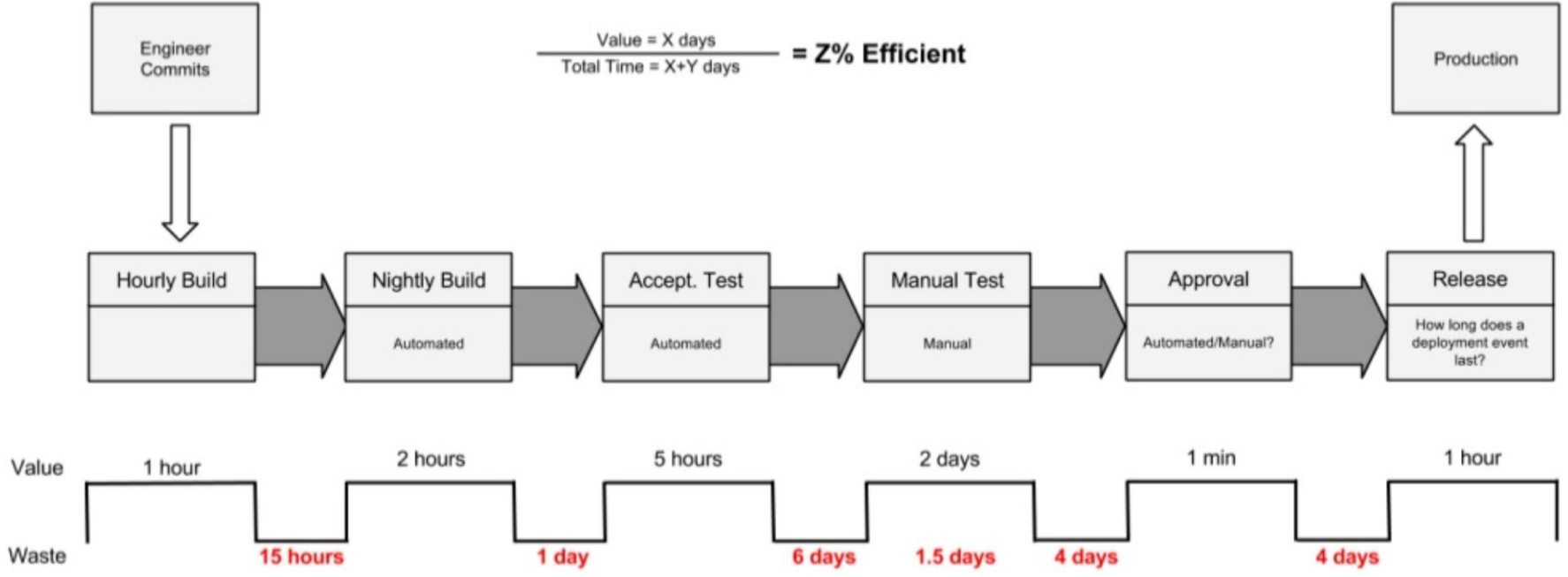
- Coding
- Testing
- Static analysis
- Code reviews
- Continuous integration
- Bug tracking
- Running local tests
- Local scalability experiments
- …

- Allocating hardware resources
- Managing OS updates
- Monitoring performance
- Monitoring crashes
- Managing load spikes, ...
- Tuning DB performance
- Running distributed at scale
- Rolling back releases
- …

QA does not stop in dev

- Ensuring product builds correctly (e.g., reproducible builds).
- Ensuring scalability under real-world loads.
- Supporting environment constraints from real systems (hardware, software, OS).
- Efficiency with given infrastructure.
- Monitoring (server, database, Dr. Watson, etc).
- Bottlenecks, crash-prone components, ... (possibly thousands of crash reports per day/minute).

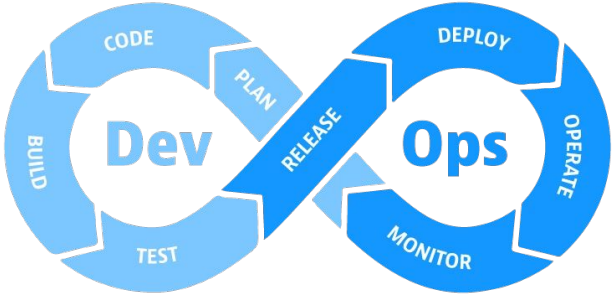
Efficiency of release pipeline



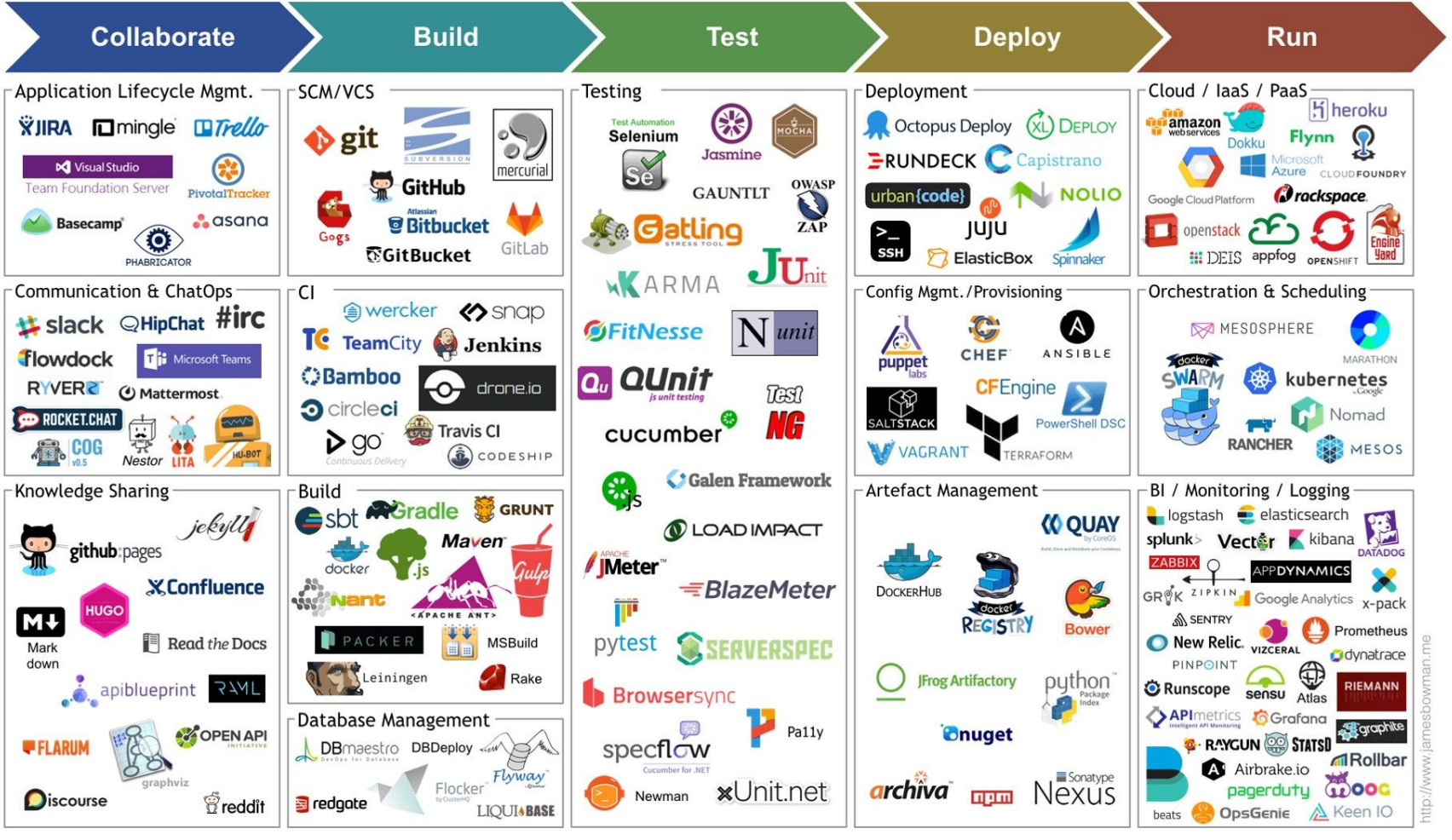
Netflix process: [https://www.slideshare.net/jmcgarr/continuous-delivery-at-netflix-and-beyond]

Common DevOps practices

- All configurations in version control.
- Test and deploy in containers.
- Automated testing, testing, testing, ...
- Monitoring, orchestration, and automated actions in practice.
- Microservice architectures.
- Frequent releases.



Heavy tooling/automation



http://www.jamesbowman.me

Tooling/automation, examples

- Infrastructure as code: Ansible, **Terraform**, Puppet, Chef, **CloudFormation**.
- CI/CD: **Jenkins**, TeamCity, GitLab, Shippable, Bamboo, Azure DevOps, Bitbucket Pipelines.
- Test automation: Selenium, Cucumber, Apache JMeter.
- Containerization: **Docker**, Rocket, Unik.
- Orchestration: **Kubernetes**, Swarm, Mesos.
- Software deployment: Elastic Beanstalk, Octopus, Vamp.
- Measurement/Monitoring: Datadog, DynaTrace, Kibana, NewRelic, ServiceNow, **Prometheus**, **Grafana**.

- Lightweight virtualization.
- Sub-second boot time.
- Shareable virtual images with full setup & configuration.
- Used in development and deployment.
- Separate docker images for separate services (web server, business logic, database, …).
- Weaker isolation than virtual machines: less secure.

Configuration management, Infrastructure as Code (IaC)

- Scripts to change system configurations (configuration files, install packages, versions, …); declarative vs imperative.
- Usually placed under version control.
- Cloud-native (e.g. AWS CloudFormation) vs. cloud-agnostic (e.g. Terraform).

(AWS CloudFormation)

```
EC2Instance:
  Type: AWS::EC2::Instance
  Properties:
    InstanceType:
      Ref: InstanceType
    SecurityGroups:
    - Ref: InstanceSecurityGroup
    KeyName:
      Ref: KeyName
    ImageId:
      Fn::FindInMap:
      - AWSRegionArch2AMI
      - Ref: AWS::Region
      - Fn::FindInMap:
        - AWSInstanceType2Arch
        - Ref: InstanceType
        - Arch
```

(Terraform)

```
provider "aws" {
    access_key = "${var.access_key}"
    secret_key = "${var.secret_key}"
    region = "us-east-1"
}

resource "aws_instance" "ec2_instance" {
    ami = "${var.ami_id}"
    count = "${var.number_of_instances}"
    subnet_id = "${var.subnet_id}"
    instance_type = "${var.instance_type}"
    key_name = "${var.ami_key_pair_name}"
}
```

Kubernetes container orchestration

- Manages which container to deploy to which machine.
- Launches and kills containers depending on load.
- Manage updates and routing.
- Automated restart, replacement, replication, scaling.
- Kubernetes master controls many nodes.

Monitoring

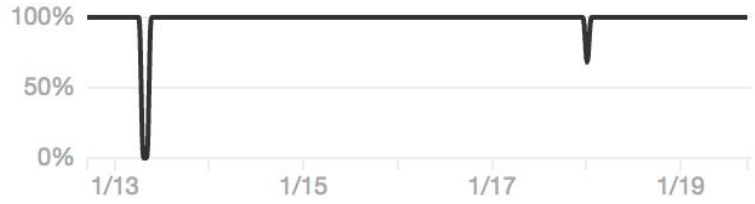
- Monitor server health.
- Monitor service health.
- Collect and analyze measures or log files.
- Dashboards and triggering automated decisions.
- Many tools, e.g., Grafana as dashboard, Prometheus for metrics, Loki + ElasticSearch for logs.
- Push and pull models.

DevOps Measurement / Analytics

- When you service 1,000s of request per second, understanding performance can be hard.
- **What would you measure?**

APP SERVER AVAILABILITY

98.6152%



MEAN WEB RESPONSE TIME

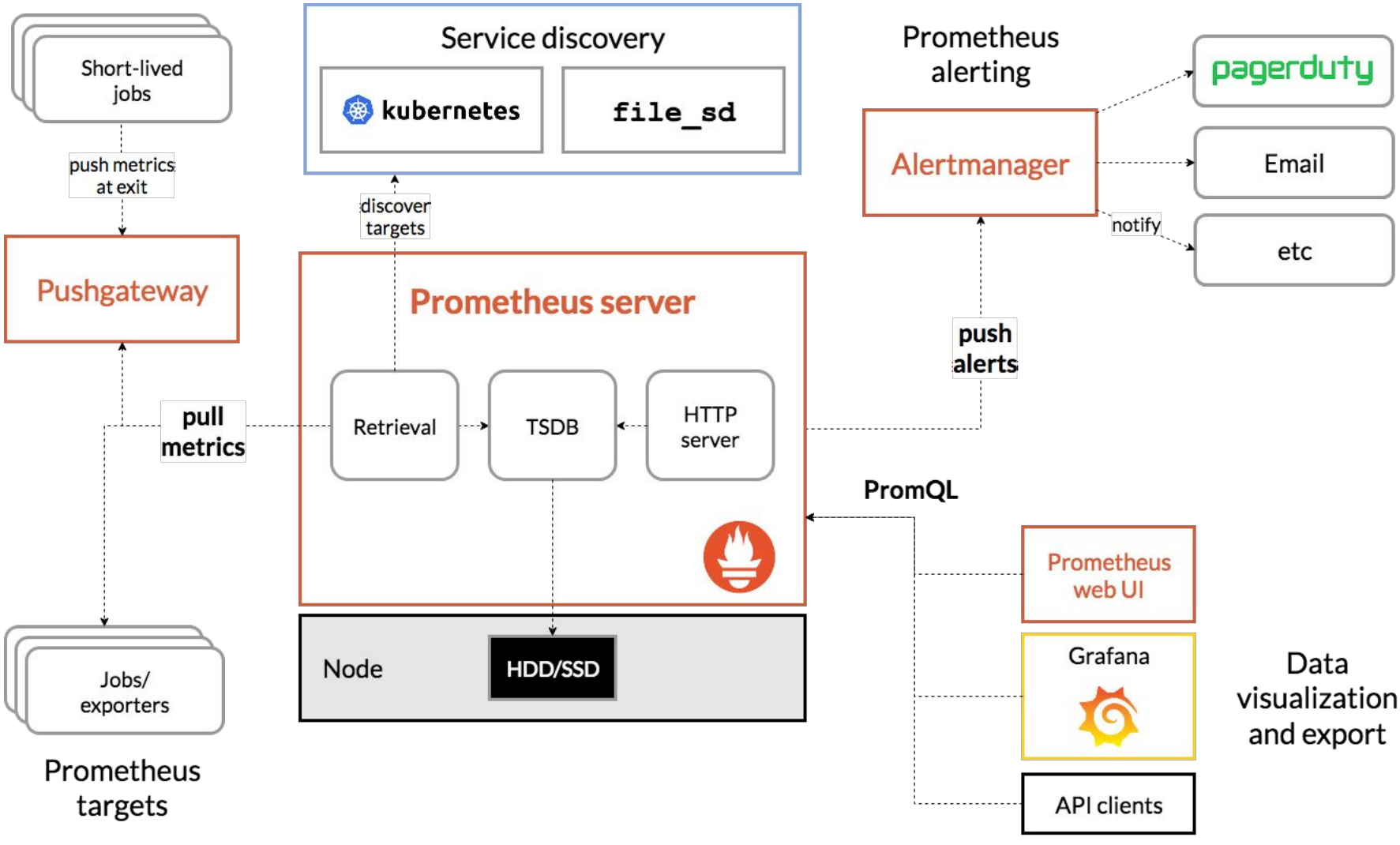
196ms

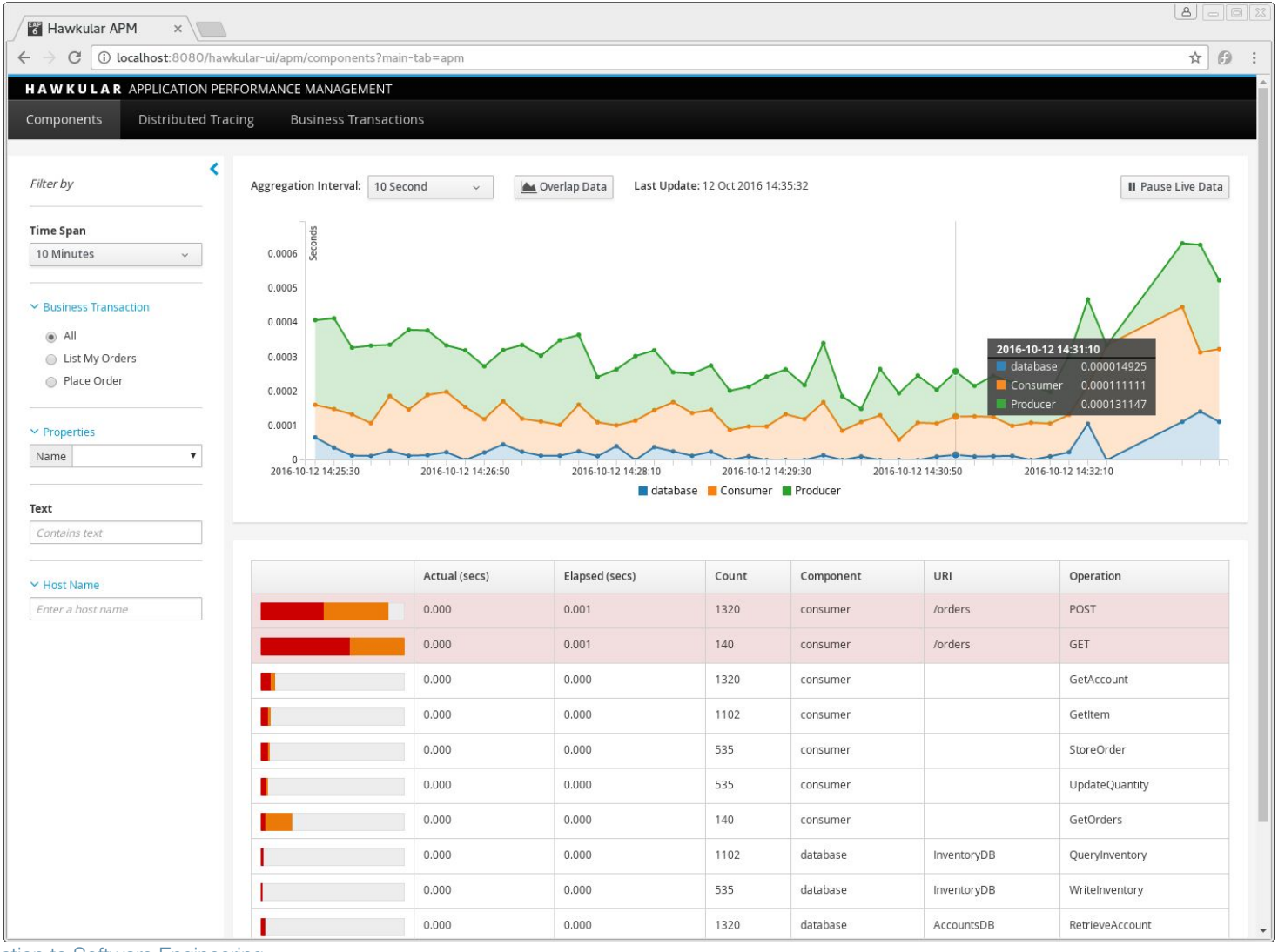
MEAN API RESPONSE TIME

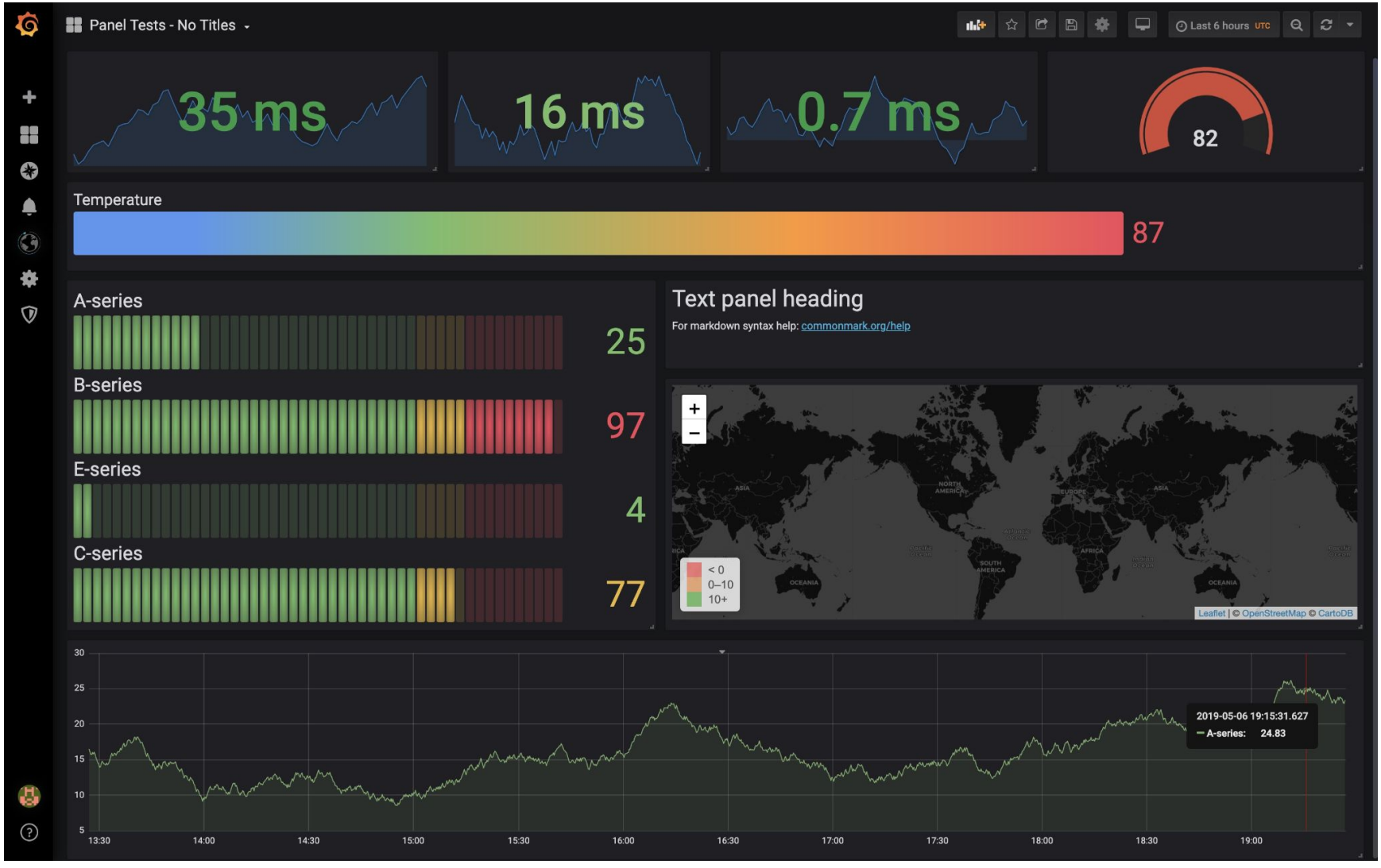
47ms

98TH PERC. WEB RESPONSE TIME

761ms


Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export


Hawkular APM
localhost:8080/hawkular-ui/apm/components?main-tab=apm
HAWKULAR APPLICATION PERFORMANCE MANAGEMENT
Components
Distributed Tracing
Business Transactions
Filter by
Time Span
10 Minutes
Business Transaction
All
List My Orders
Place Order
Properties
Name
Text
Contains text
Host Name
Enter a host name
Aggregation Interval: 10 Second
Overlap Data
Last Update: 12 Oct 2016 14:35:32
Pause Live Data
Seconds
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
2016-10-12 14:25:30
2016-10-12 14:26:50
2016-10-12 14:28:10
2016-10-12 14:29:30
2016-10-12 14:30:50
2016-10-12 14:32:10
2016-10-12 14:31:10
database 0.000014925
Consumer 0.000111111
Producer 0.000131147
database Consumer Producer
Actual (secs) Elapsed (secs) Count Component URI Operation
0.000 0.001 1320 consumer /orders POST
0.000 0.001 140 consumer /orders GET
0.000 0.000 1320 consumer GetAccount
0.000 0.000 1102 consumer GetItem
0.000 0.000 535 consumer StoreOrder
0.000 0.000 535 consumer UpdateQuantity
0.000 0.000 140 consumer GetOrders
0.000 0.000 1102 database InventoryDB QueryInventory
0.000 0.000 535 database InventoryDB WriteInventory
0.000 0.000 1320 database AccountsDB RetrieveAccount


Panel Tests - No Titles
Last 6 hours UTC
35 ms
16 ms
0.7 ms
82
Temperature
87
A-series
25
B-series
97
E-series
4
C-series
77
Text panel heading
For markdown syntax help: commonmark.org/help
< 0
0–10
10+
Leaflet | © OpenStreetMap © CartoDB
30
25
20
15
10
5
13:30
14:00
14:30
15:00
15:30
16:00
16:30
17:00
17:30
18:00
18:30
19:00
2019-05-06 19:15:31.627
A-series: 24.83

Testing in production??

Crash telemetry

What if...?

... we had plenty of subjects for experiments.

... we could randomly assign subjects to treatment and control group without them knowing.

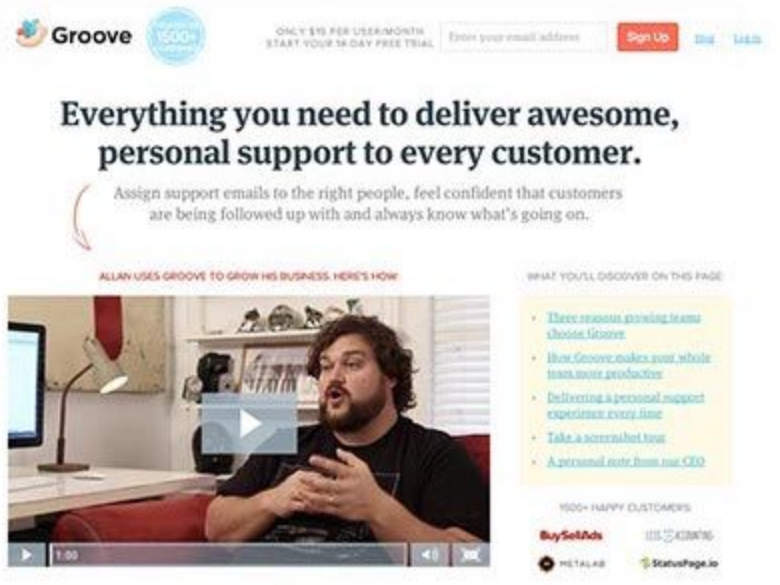
... we could analyze small individual changes and keep everything else constant.

→ Ideal conditions for controlled experiments.

A/B testing

Original: 2.3%

Long Form: 4.3%



Experiment size

- With enough subjects (users), we can run many many experiments.
- Even very small experiments become feasible.

A/B testing

- **Implement alternative versions** of the system:
  - Using feature flags (decisions in implementation)
  - Separate deployments (decision in router/load balancer).
- **Map users** to treatment group:
  - Randomly from distribution.
  - Static user - group mapping.
  - Online service (e.g., launchdarkly, split).
- **Monitor outcomes** per group:
  - Telemetry, sales, time on site, server load, crash rate.

Feature flags

- Boolean options.
- Good practices; features are:
  - Tracked explicitly and documented.
  - Localized and independent.
- External mapping of flags to customers:
  - Who should see what configuration.
    - e.g., the same 1% of users sees one_click_checkout, or 50% of beta-users and 90% of developers and 0.1% of all users.

```
if (features.enabled(userId, "one_click_checkout")) {
     // new one click checkout function
} else {
     // old checkout functionality
}

def isEnabled(user): Boolean = (hash(user.id) % 100) < 10
```

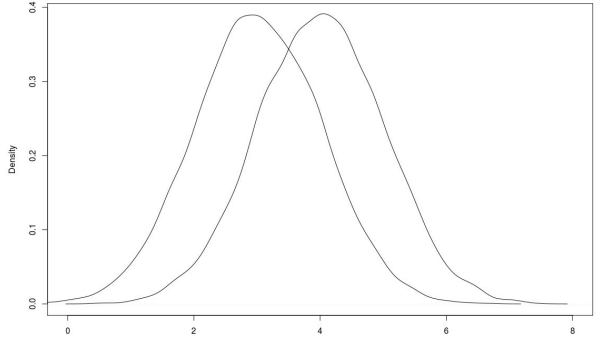
Comparing outcomes

Group A

- base game
- 2158 Users
- average 18:13 min time on site

Group B

- game with extra god cards
- 2020 Users
- average 20:24 min time on site



The Morality Of A/B Testing

Josh Constine @joshconstine / 11:50 PM EDT • June 29, 2014

Comment

We don't use the "real" Facebook. Or Twitter. Or Google, Yahoo, or LinkedIn. We are almost all part of experiments they quietly run to see if different versions with little changes make us use more, visit more, click more, or buy more. By signing up for these services, we technically give consent to be treated like guinea pigs.

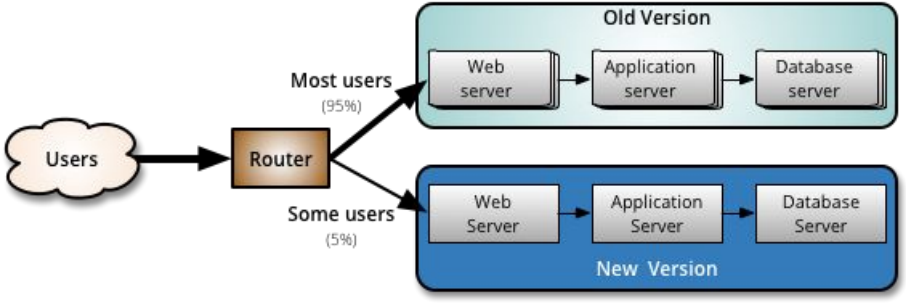
But this weekend, Facebook stirred up controversy because one of its data science researchers published the results of an experiment on 689,003 users to see if showing them more positive or negative sentiment posts in the News Feed would affect their happiness levels as deduced by what they posted. The impact of this experiment on manipulating emotions was tiny, but it

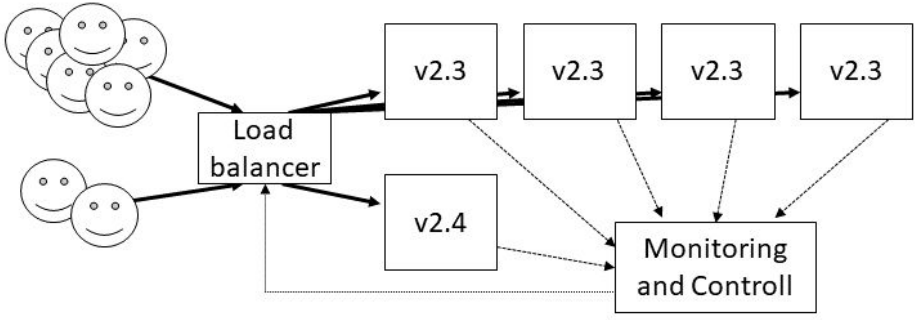
[https://techcrunch.com/2014/06/29/ethics-in-a-data-driven-world/]

Canary releases

Canary releases

- Testing releases in production.
- **Incrementally deploy** a new release to users, not all at once.
- **Monitor difference in outcomes** (e.g., crash rates, performance, user engagement).
- **Automatically roll back** bad releases.
- Simple version of A/B testing.
- Telemetry essential.



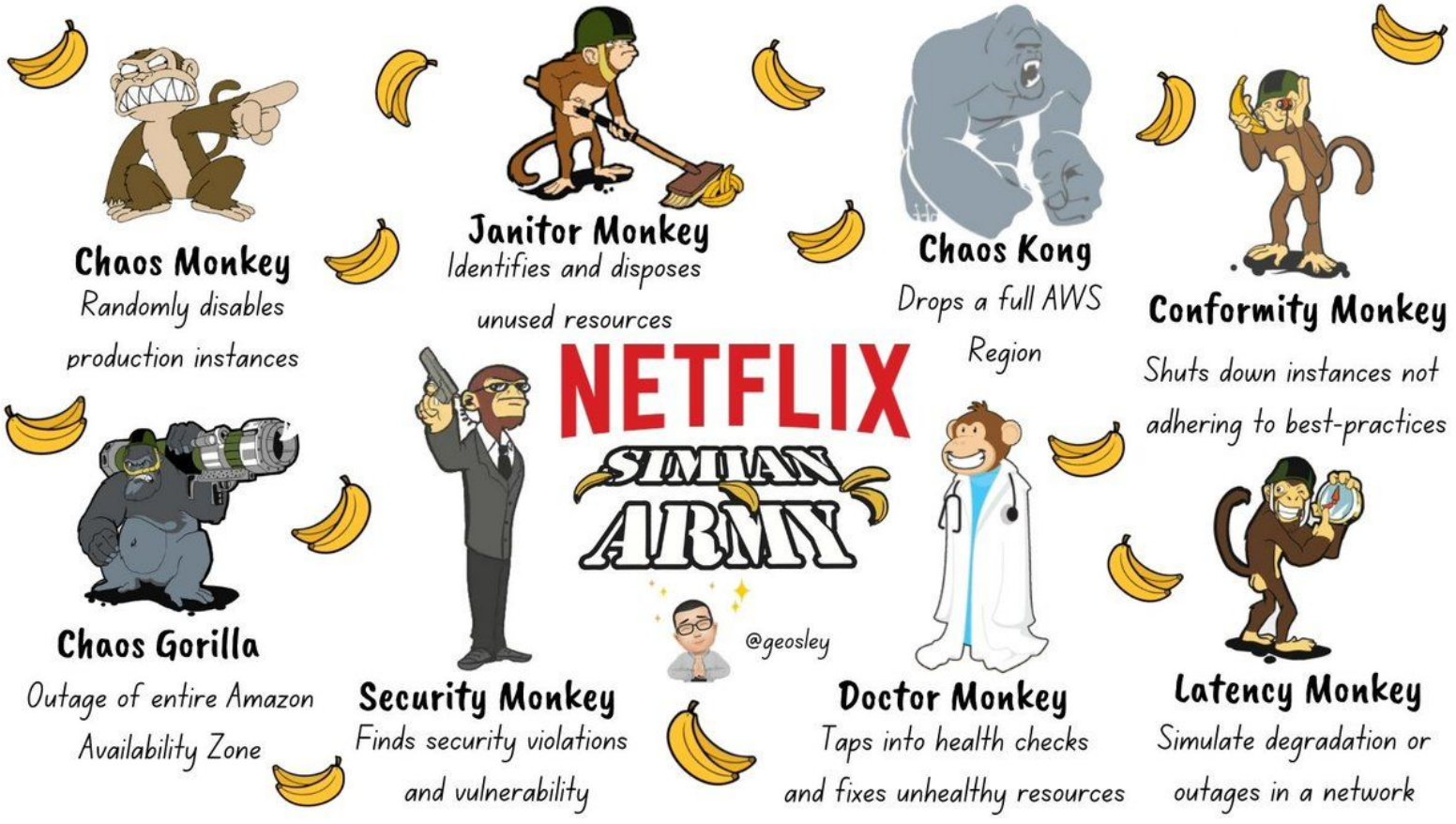


Chaos Experiments

"A chaos experiment in production allows you to measure a system's traffic, patterns of use, resource utilization, and practical application. This information **can help build resilience** within the system and improve cyber defenses."

[https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116]
[https://www.cisecurity.org/insights/blog/the-chaos-experiment]

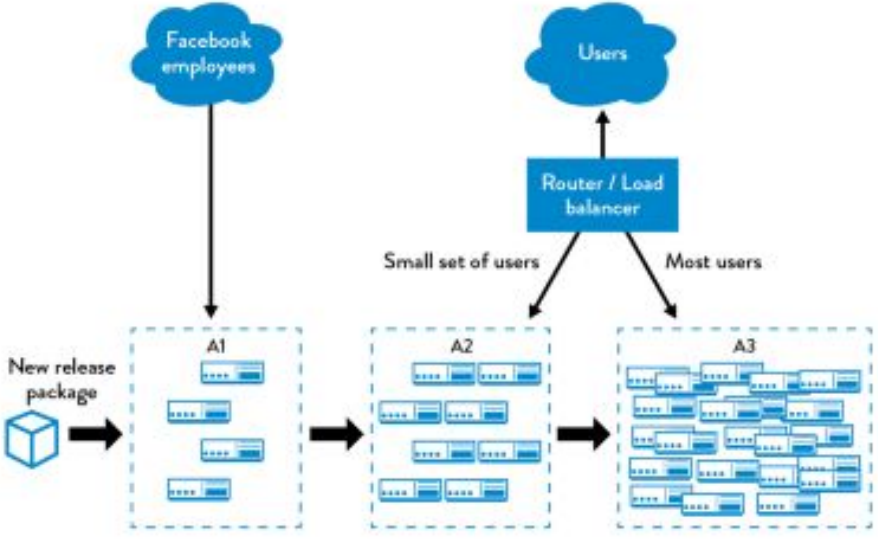
Chaos Experiments

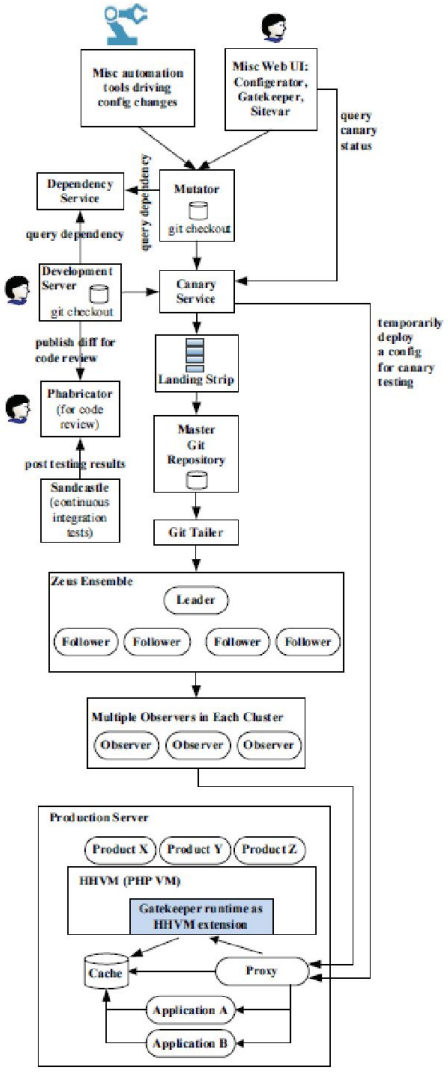


[https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116]
[https://www.cisecurity.org/insights/blog/the-chaos-experiment]

The complexity of real DevOps pipelines

- Incremental rollout, re-configuring routers.
- Canary testing.
- Automatic change rollback.





Further readings: Chunqiang et al. *Holistic configuration management at Facebook.* In Proceedings of the 25th Symposium on Operating Systems Principles. 2015.

DevOps summary

- Code only has value if it can be used. Most modern systems are deployed rather than physically shipped.
- Increasing automation of tests and deployments.
- Containers and configuration management tools help with automation, deployment, and rollbacks.
- Monitoring becomes crucial.
- Enables many opportunities for testing in production: A/B testing, canary releases, feature flags as mechanism.