CS 4530
Fundamentals of Software Engineering

Module 16: Refactoring and Technical Debt

Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

Learning Goals

By the end of this lesson, you should be able to…

- Define *refactoring, technical debt,* and give examples.
- Explain how refactoring fits into agile process and help reduce technical debt
- Suggest when it may be appropriate to accrue technical debt and when it may be appropriate to retire it

Refactoring

Refactoring is the process of applying transformations, *refactorings*, to a program and the *internal structure* of the system is improved

Goals:

- keep program readable, understandable, and maintainable
- by eliminating small problems soon, you can avoid big troubles later

Characteristics:

- **behavior-preserving**, i.e. do not change what the program does
- **incremental**, i.e. proceeds in small steps with tests at each stage

Why Refactor?

Altered design will make testing easier

Altered design will improve maintainability

Fix sloppiness by programmers

New or anticipated requirements require a different design

Retire or avoid technical debt

When to refactor?

Refactoring is incremental redesign

Acknowledge that it is *difficult to get design right the first time*

When?

- adding new functionality,
- fixing a bug,
- doing code review, or
- any time

A key part of TDD!

Refactoring evolves design in increments

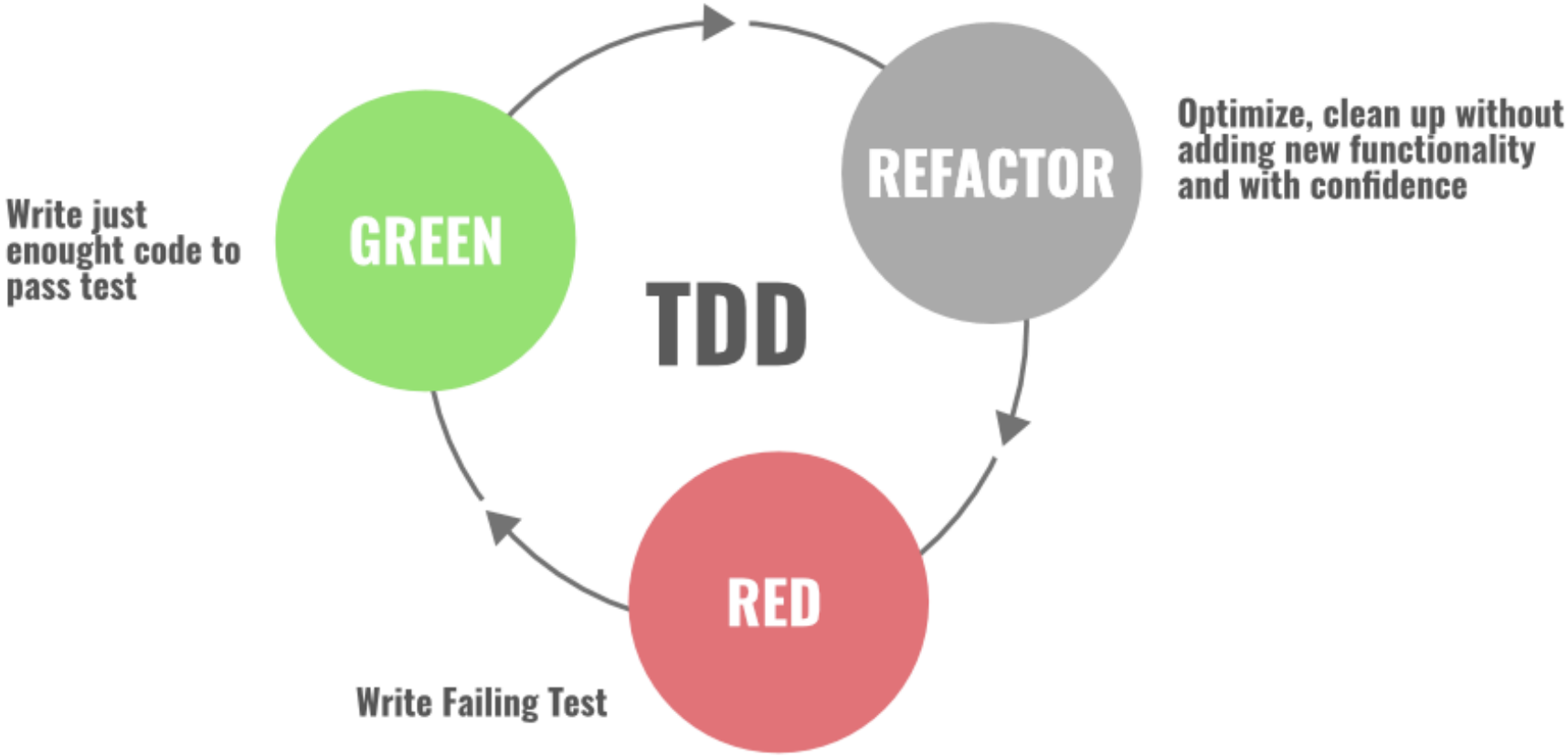
Refactoring reduces technical debt

Refactoring with TDD

Red: start writing failing “red-test”. Stop and check what needs to be written

Green: next, write simplest code that gets tests to “green”

Refactor: finally, focus on improving & enhancing code while keeping test green



Example

Original Code

```
function greeter (firstName : String, lastName : String) {
        return "Hello, " + firstName + " " + lastName;
}
document.body.innerHTML = greeter(“Jane","Doe");
```

Refactored Code # 1

```
function greeter (firstName : String, lastName : String, greeting = "Hello, ") {
        return greeting + firstName + " " + lastName;
}
document.body.innerHTML = greeter(“Jane”,"Doe");
```

Refactored Code # 2

```
function greeter (firstName : String, lastName : String, greeting : String) {
        return greeting + firstName + " " + lastName;
}
document.body.innerHTML = greeter(“Jane","Doe“,"Hello, ");
```

Dad

Martin Fowler is the “father” of refactoring

“Any fool can write code that a computer can understand

Good programmers write code that humans can understand”

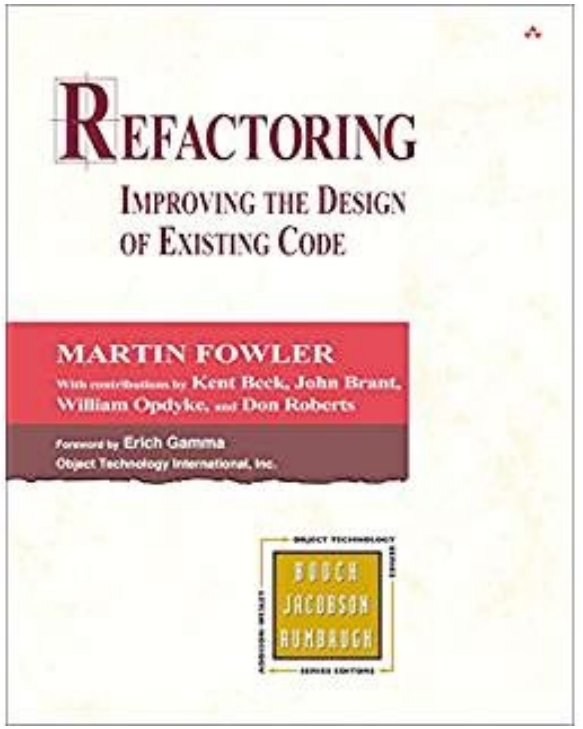
The Book

A catalogue of refactorings, similar to the design patterns in the GoF book

- Names each transformation
- Helpful for team communication
- Names “bad smells” (triggers for refactorings)
- Discusses when and how to apply refactorings

Many refactorings are the inverse of another refactoring

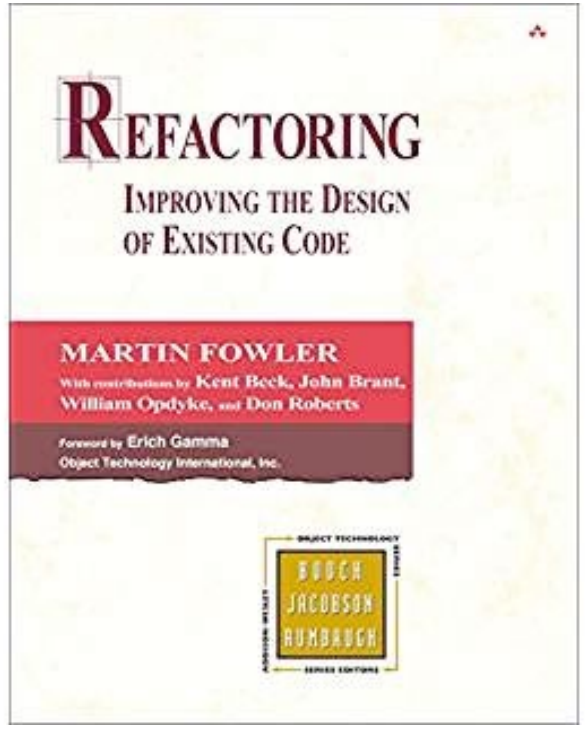
- often there is not a unique “best” solution
- discussion of the tradeoffs



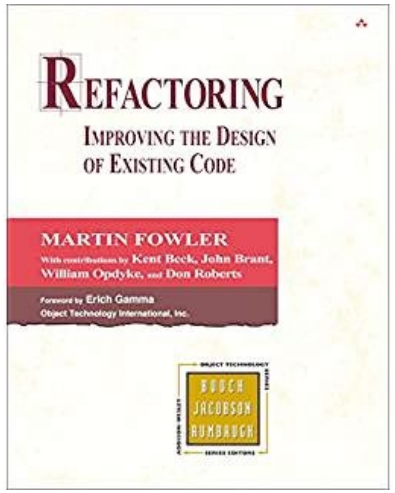
The List

Fowler gave colorful names to his “code smells”

Mysterious Name
Duplicated Code
Long Function
Long Parameter List
Global Data
Mutable Data
Divergent Change
Shotgun Surgery
Feature Envy
Data Clumps
Primitive Obsession
Repeated Switches
Loops
Lazy Element
Speculative Generality
Temporary Field
Message Chains
Middle Man
Insider Trading
Large Class
Alternative Classes with Different Interfaces
Data Class
Refused Bequest



Renaming



Is the most common…

- Rename Function (124)
- Rename Variable (137)
- Rename Field (244)

We are often afraid to rename things, thinking it's not worth the trouble, but a good name can save hours of future puzzled incomprehension

Renaming is not just an exercise

When you are not happy with a name, it's often a sign of a deeper design malaise.

Puzzling over a tricky name leads to significant improvements to your code

Remember:
Use Good Names!

Renaming

Luckily, VSC automates this and many other common transformations

Local Refactorings

https://refactoring.guru/

Rename	provides better intuition for renamed thing's purpose
Extract Method	enables reuse; avoids cut-and-paste; improves readability
Inline Method	replace a method call with method's body; often intermediate step
Extract Local	introduce a new local variable for an expression
Inline Local	replace a local variable with the expression that defines its value
Change Method Signature	reorder a method's parameters
Encapsulate Field	introduce getter/setter methods
Convert Local Variable to Field	sometimes useful to enable application of Extract Method

Typescript-specific Refactoring https://www.jetbrains.com/help/webstorm/specific-typescript-refactorings.html

Type-Related Refactorings

aka ***Refactoring by Abstraction***

Bad abstraction is worst than duplication

- (pieces of code that look the same, still represent different concepts).

Generalize Declared Type	replace type of a declaration with more general type
Extract Interface	create new interface, and update code to use it where possible
Pull Up Members	move methods and fields to a superclass
Infer Generic Type Arguments	infer type arguments for “raw” uses of generic types

Use “Rule of Three” – Three strikes and you refactor

https://understandlegacycode.com/blog/refactoring-rule-of-three/

Typescript-specific Refactoring https://www.jetbrains.com/help/webstorm/specific-typescript-refactorings.html

Refactoring Benefits

Small incremental steps that preserve program behavior

- …simplify regression testing

Aiming for simple steps

- …allows for automation

Refactoring needs not proceed in a straight line

- …sometimes, you want to undo a step you did earlier
- …when you have insights for a better design
- Having a name for what you did makes undos easier

Refactoring Risks

Developer time is valuable: is this the best use of your time today?

Despite best intentions, may not be safe

Potential for version control conflicts

It brings us to Technical Debt

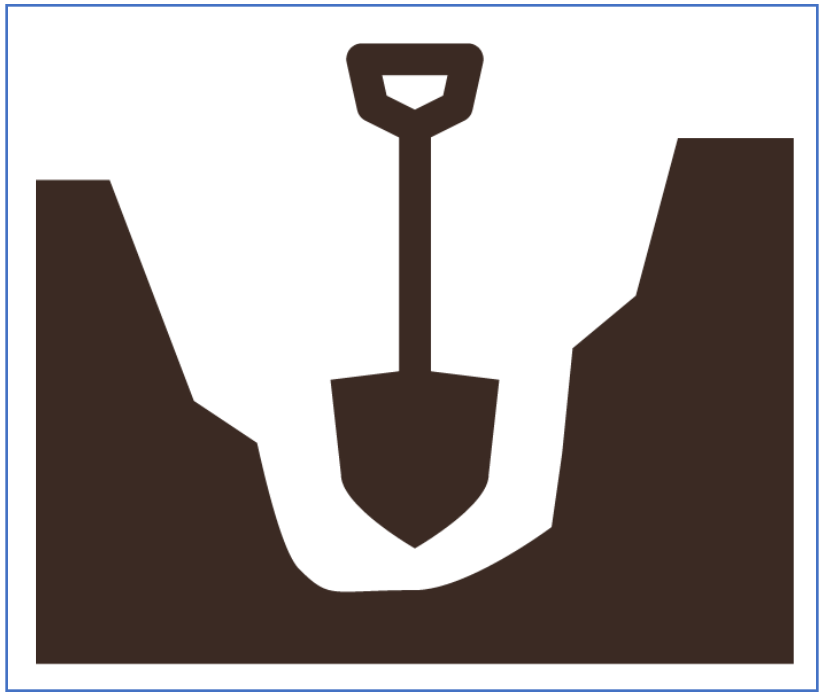
Technical Debt

… is the accumulation of internal problems in a code base

Internal because they don't show as user-visible failures

Examples:

- Code Smells
- Missing tests
- Missing documentation
- Dependency on old versions of third-party systems
- Inefficient algorithms



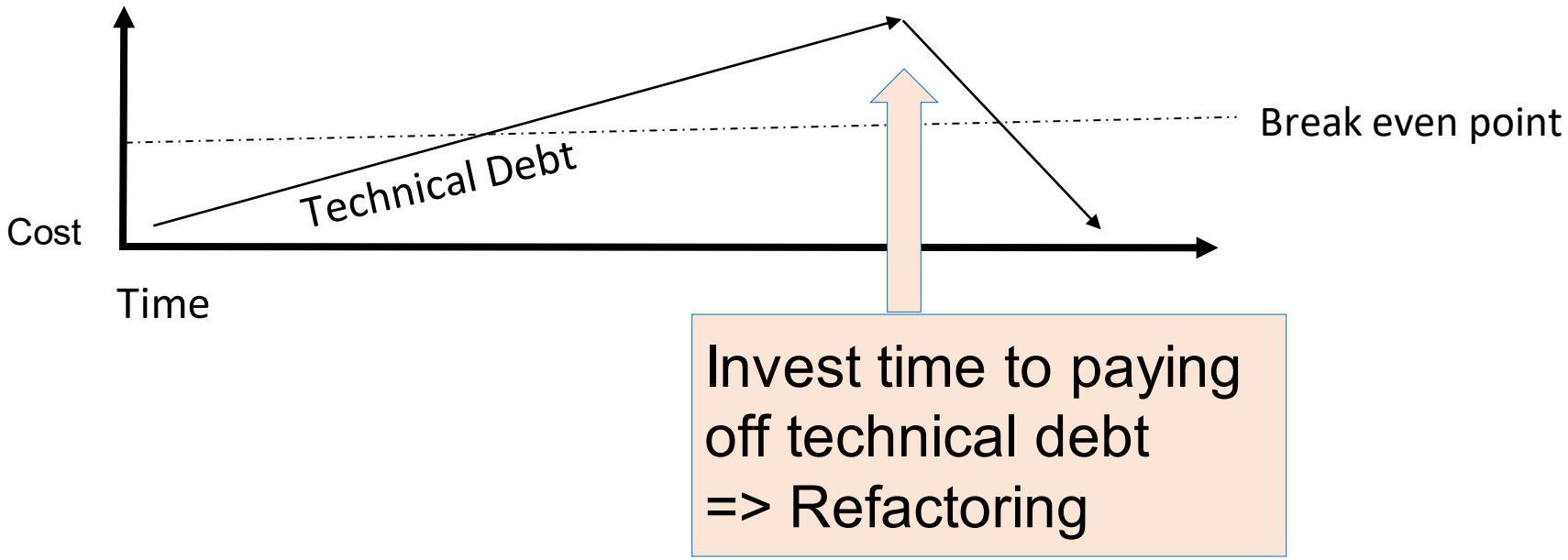
Not just code!

Technical debt

…has costs, i.e. interest on the debt

Examples of Debt	Examples of Cost
Code Smells	“Smelly” code is less flexible
Missing tests	need to revert breaking change
Missing documentation	can’t figure out how to use
Dependency on versions of third-party	may have to take over maintenance of old system
Inefficient/non-scalable algorithms	lose potential customers

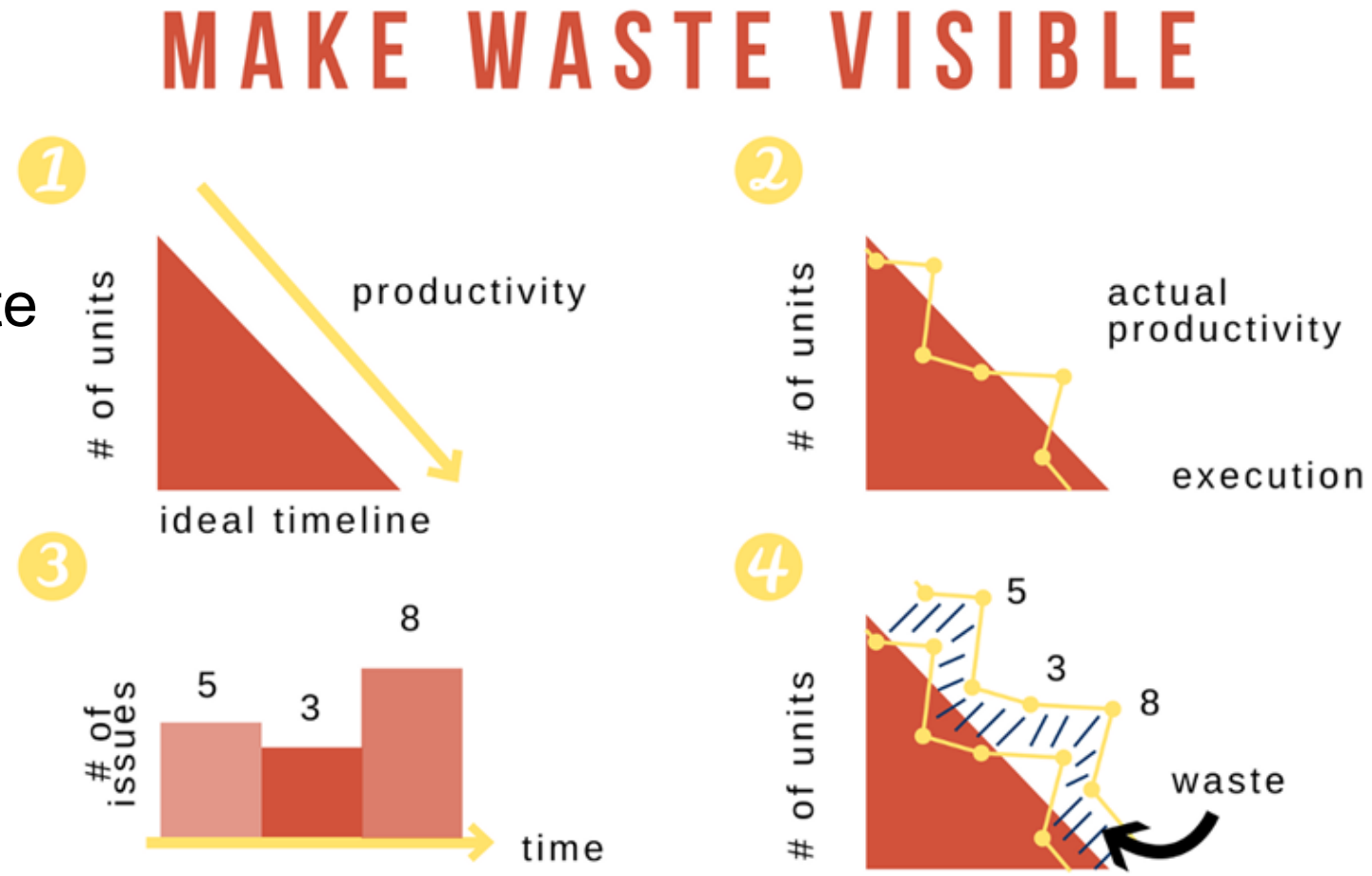
Interest accrues over time



Make Technical Debt Visible

Here are the steps:

- Plan the ideal
- Track your actual
- Track what you spend on waste
- Put it all together



Help stakeholders visualize data (like progress, effect of debt, refactoring)

https://www.scrum.org/resources/blog/making-tech-debt-visible

Reasons to go into Debt

Prototyping

- If code will be discarded, or rewritten, don't waste time perfecting it

Getting a product out the door

- Time is often crucial in a competitive environment

Fixing a critical failure

- People are waiting

Maybe a simple algorithm is good enough

- "Premature optimization is the root of all evil" — Tony Hoare, Donald Knuth

Architectural debt is costliest

Total cost of ownership generally higher than implementation-level issues; harder to get out of choices of:

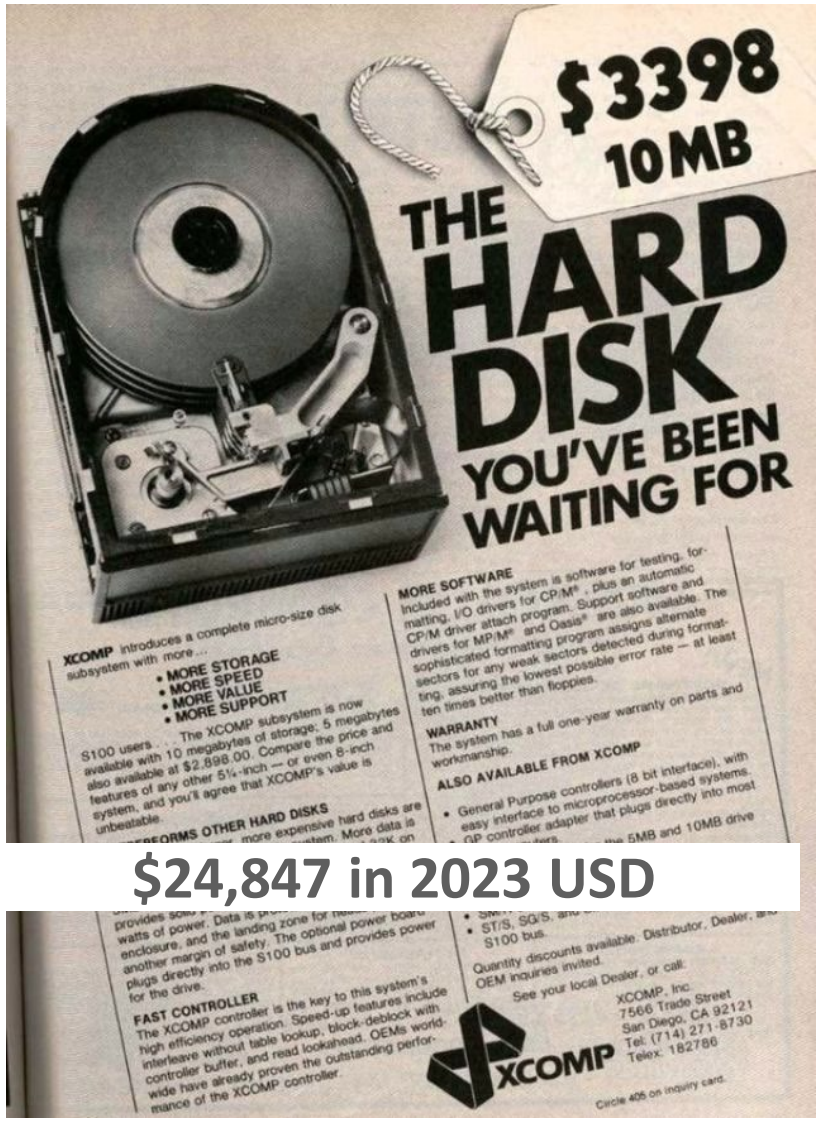
- Language
- Middleware frameworks
- Deployment pipeline

Consider:

- What are the quality attributes that our software needs to ultimately satisfy?
- How do these architectural decisions reflect those attributes?

Y2K bug as example of architectural debt

How many digits does it take to store a year?



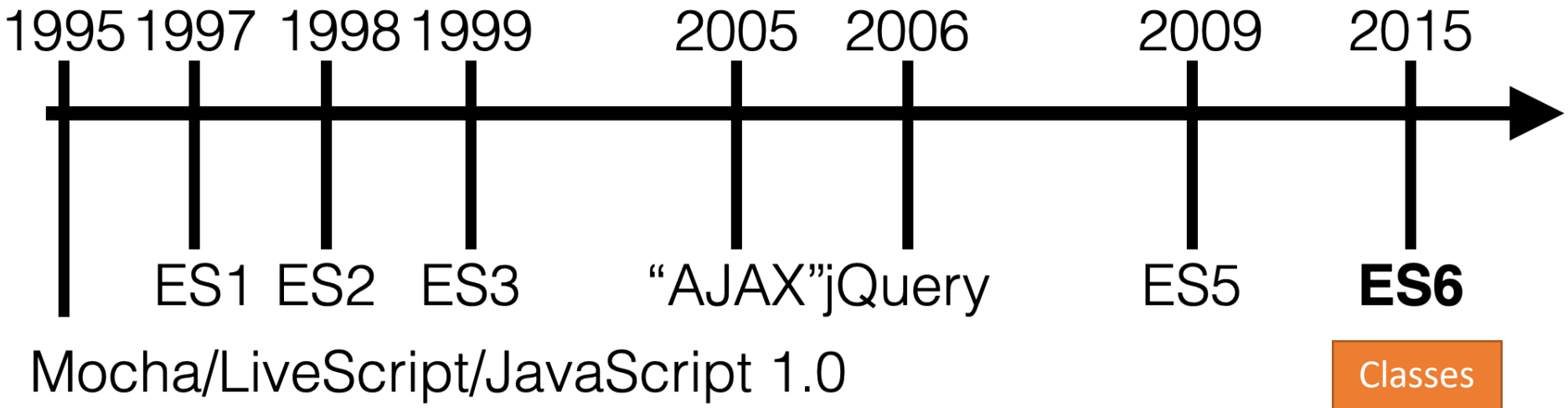
$24,847 in 2023 USD

"I just never imagined anyone would be using these systems 10 years later, let alone 20."

Evolving languages make debt

Choice of language can cause technical debt, particularly if that language is rapidly evolving.

Consider JavaScript



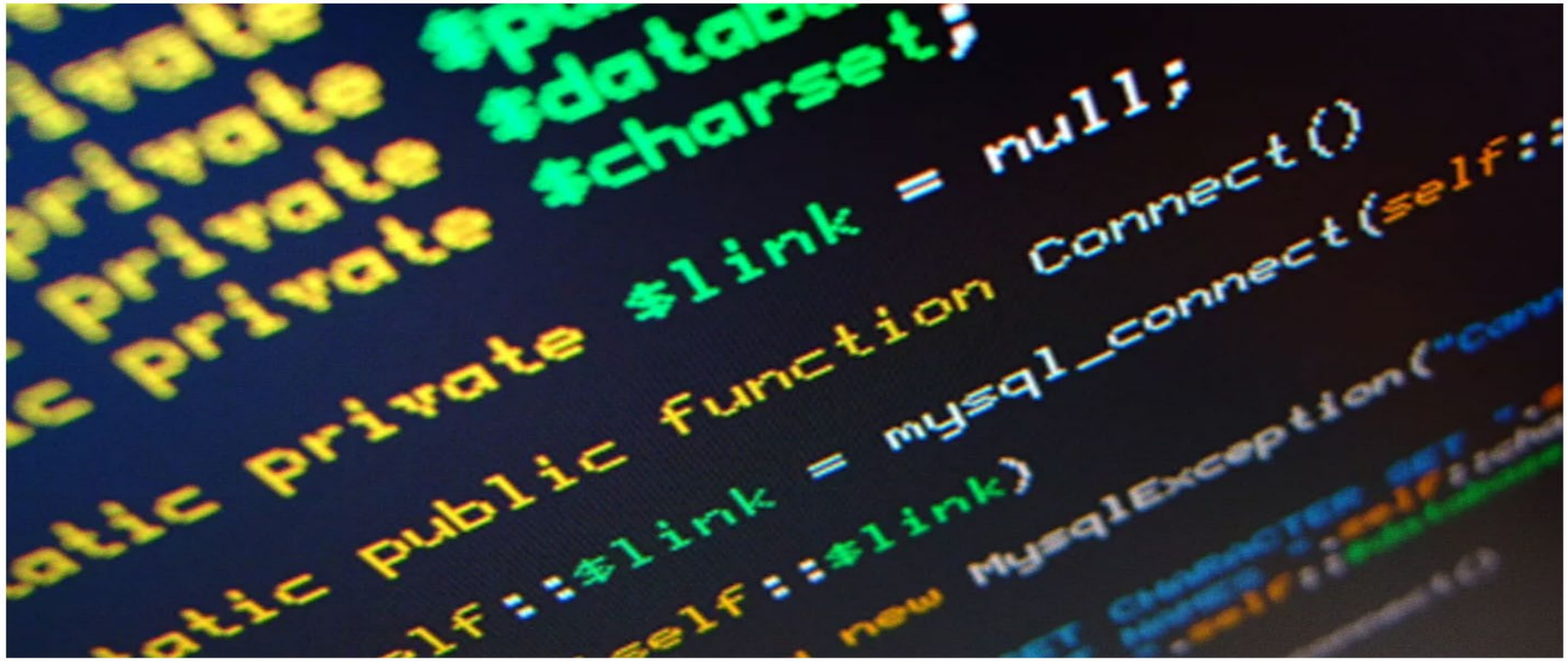
PLUS:
2016: ES7 (Array.includes)
2017: ES8 (Async/Await)
2018: ES9 (rest/spread operator, async iterators)

Facebook’s debt

04-07-14

Why Facebook Invented A New PHP-Derived Language Called “Hack”

Instead of throwing out years of legacy code, Facebook built a new branch of the language that originally underpinned TheFacebook.com. Here’s the story behind a two-year labor of love.



[IMAGE: FLICKR USER BULL3T HUGHES]

https://www.fastcompany.com/3028778/why-facebook-invented-a-new-php-derived-language-called-hack

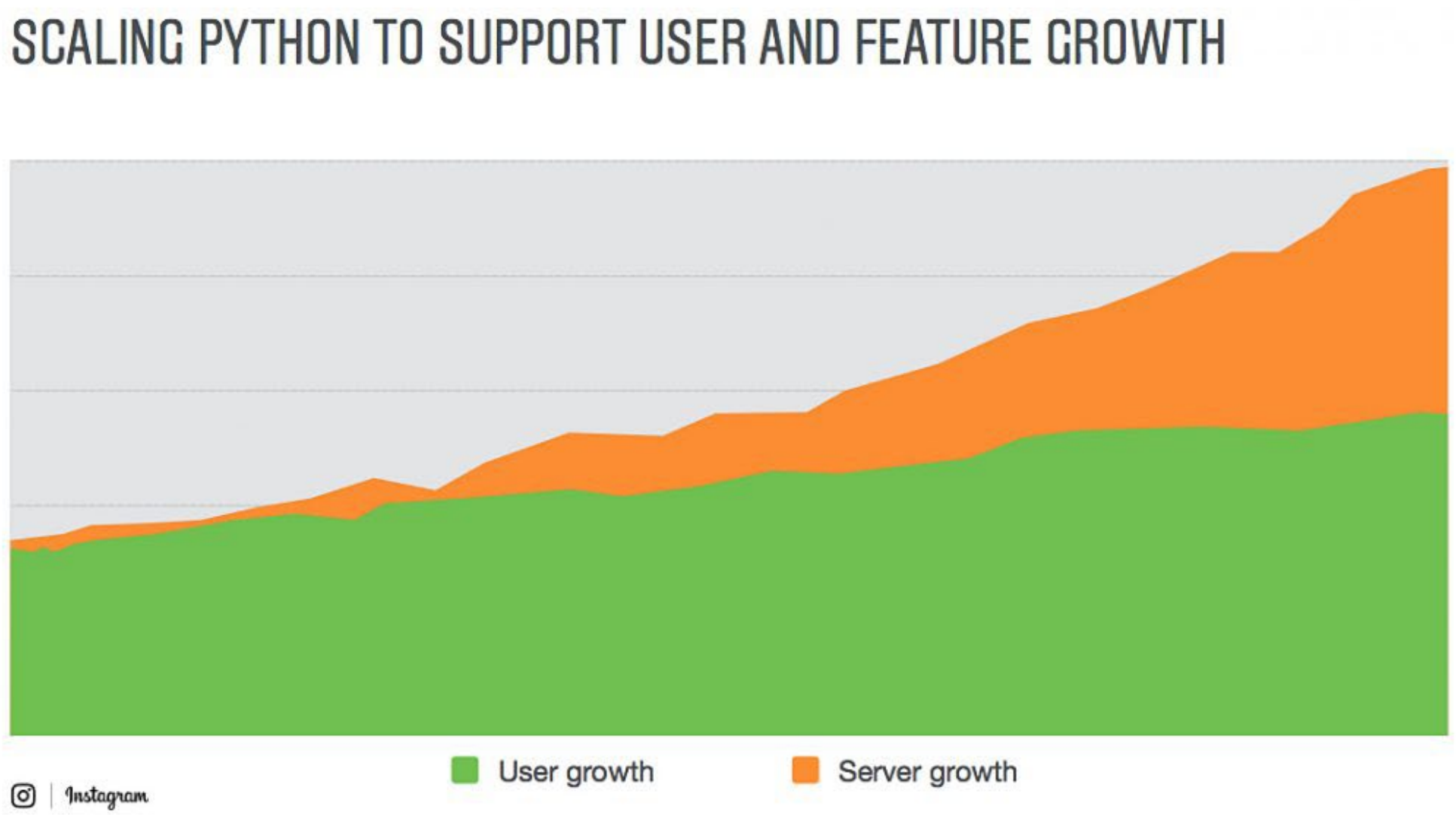
Facebook's debt

Hack added new safety features…

- …automatic type inference
- …lets you specify types of variables
- …issues an error if code is logically inconsistent
- When a file changed, two versions had to be compared to deduce what must be rechecked at a very fine-grained level
- "*Hack enables us to dynamically convert our code one file at a time*" - Facebook Technical Lead HipHop VM (HHVM)

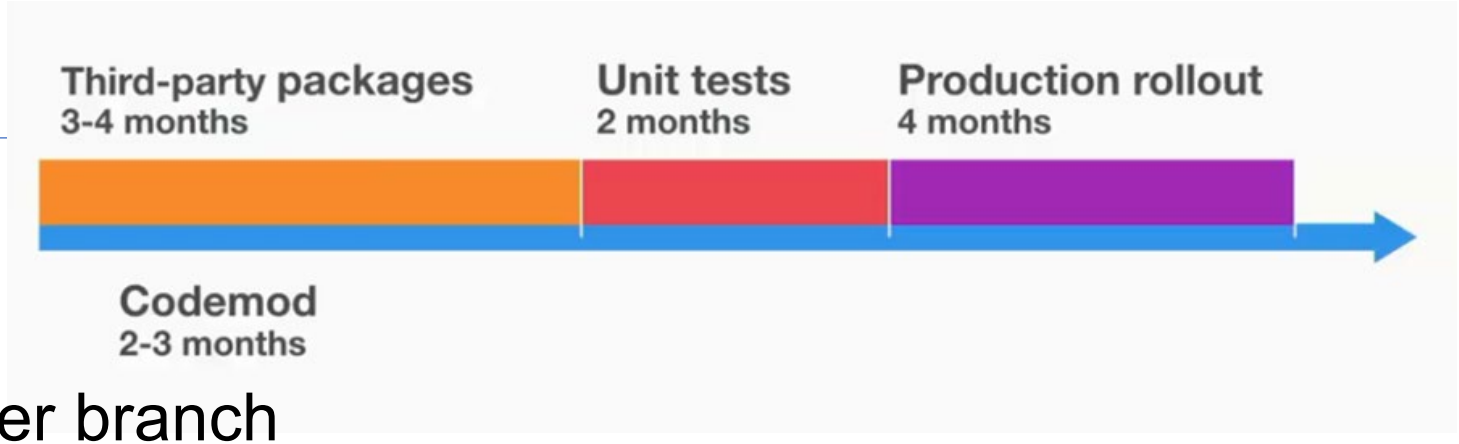
Facebook's Runtime Engine supports PHP and Hack. https://hhvm.com/

Instagram’s debt



https://thenewstack.io/instagram-makes-smooth-move-python-3/

Instagram's debt



From Python 2 to 3

- Migrated in 10 months
- All work done directly in Master branch
- Upgraded all packages (Working Rule: not in Py3 => not used)

Examples of refactorings:

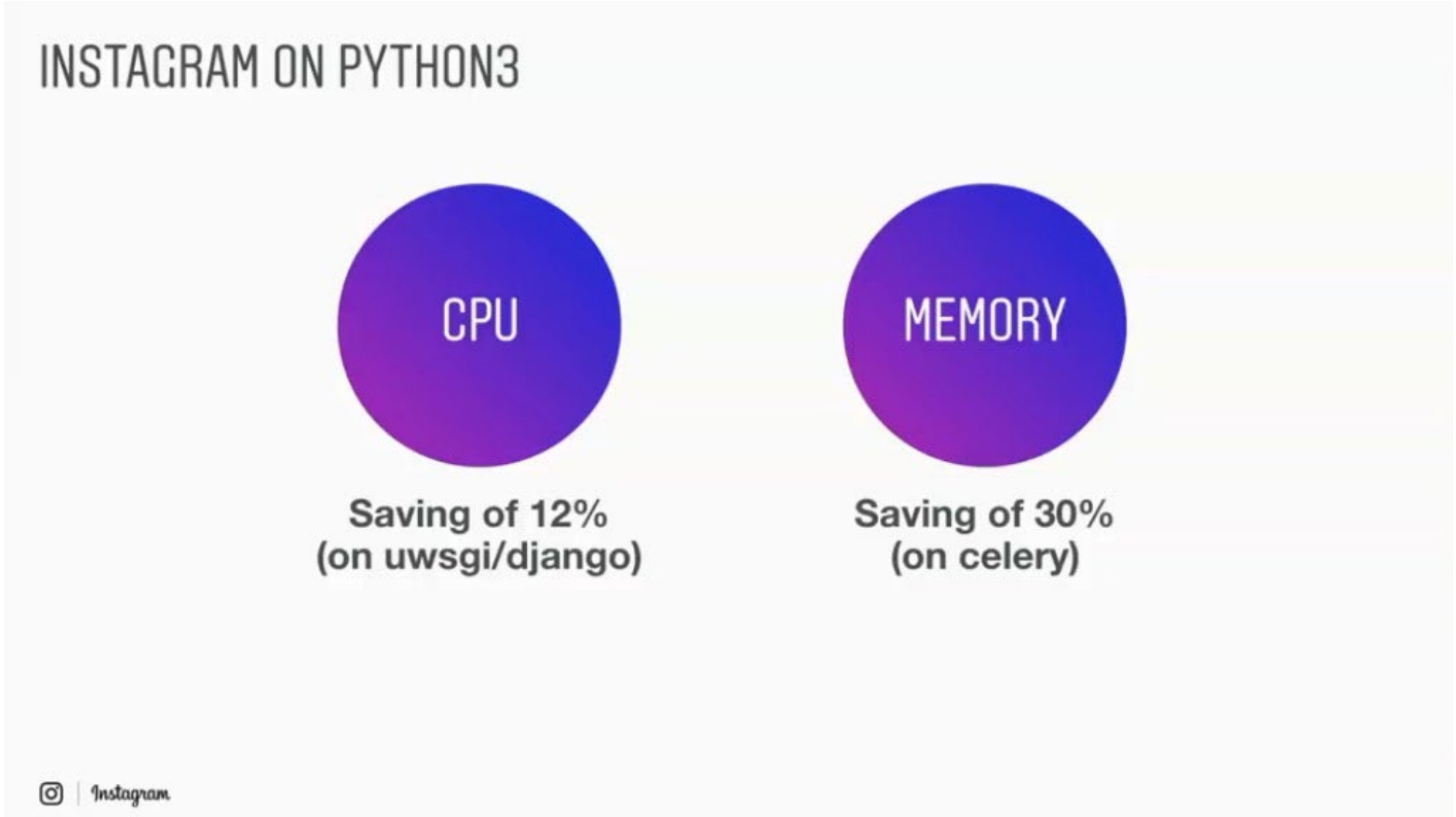
- Differences in unicode, str, bytes => add helper functions
- Differences in iterators, such as map => convert all maps to Py3 list
- Differences in dictionary order differences in the dumped JSON data => force sorted_keys in json.dump function

PyCon 2017 Keynote Talk: https://www.youtube.com/watch?v=66XoCk79kjM

http://euccas.github.io/blog/20170616/how-instagram-moved-to-python-3.html

Instagram’s debt

Dropped Python 2 in Feb 2017



Siri’s debt

Voice assistants are “dumb as a rock,” Satya Nadella (Microsoft’s chief executive)

- Clunky Code: Weeks to update code
- One big snowball!
- 6 weeks to build db for adding 1 word

How Siri, Alexa and Google Assistant Lost the A.I. Race
https://www.nytimes.com/2023/03/15/technology/siri-alexa-google-assistant-artificial-intelligence.html

Retire Technical Debt at Leisure

Set aside time to pay off technical debt:

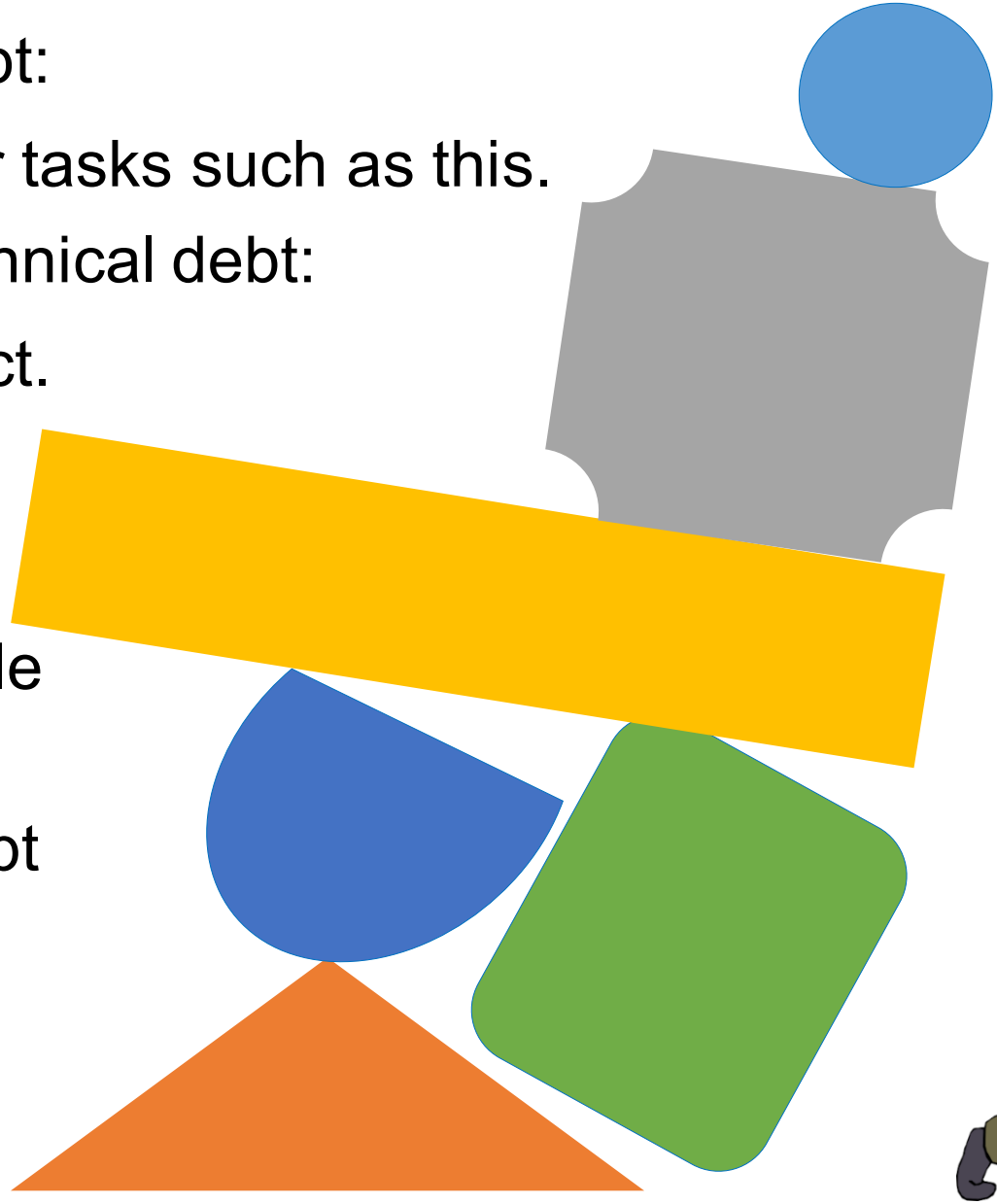
- Google has (had?) “20%-time” for tasks such as this.

A new initiative can take on some technical debt:

- Refactoring at the start of a project.

Don’t keep on putting off!

- When a crisis hits, it’s too late
- Hasty fixes to unmaintainable code multiplies problems
- Eventually mounting technical debt can bury the team



Now back to you…

Suggested Activity

- Individually, Identify 5 candidates for Refactoring in your project code.
- Then come together as a group to share and discuss. Keep that list with you to work on during the project.

Learning Goals

You should now be able to…

- Define *refactoring, technical debt,* and give examples.
- Explain how refactoring fits into agile process and help reduce technical debt
- Suggest when it may be appropriate to accrue technical debt and when it may be appropriate to retire it