Principles of Software Construction: Objects, Design, and Concurrency

Part 4: et cetera

Toward SE in practice: Empirical methods, DevOps

Josh Bloch **Charlie Garrod**

Administrivia

- Homework 6 available
  - Checkpoint deadline this Thursday
  - Due Wednesday, April 29th

Key concepts from last Thursday

- Parallel streams conclusion, performance trolling
- SE as a sociotechnical system

Major topics in 17-313 (Foundations of SE)

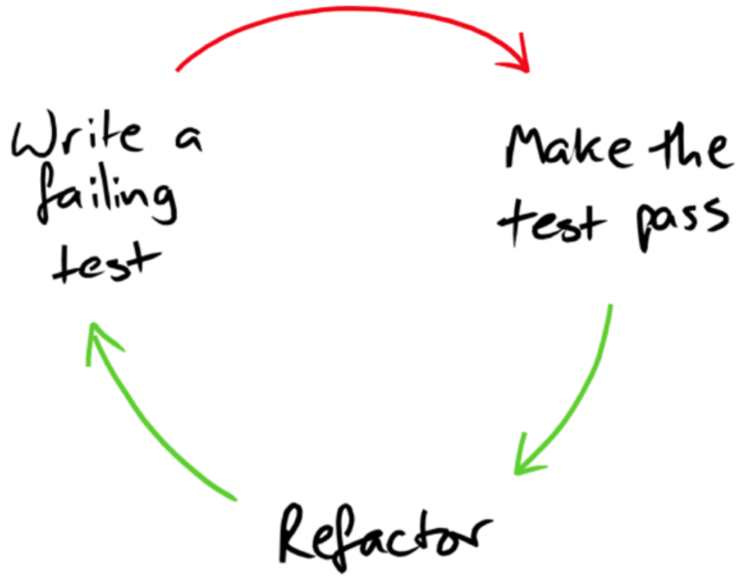
- Process considerations for software development
- Requirements elicitation, documentation, and evaluation
- Design for quality attributes
- Strategies for quality assurance
- Empirical methods in software engineering
- Time and team management
- Economics of software development

Today: Software engineering in practice

- SE empirical methods: Test-driven development case study
- Version and release management
  - Introduction to DevOps

Test-driven development (TDD), informally

Test-driven development (TDD), informally



From Growing Object-Oriented Software by Nat Pryce and Steve Freeman
http://www.growing-object-oriented-software.com/figures.html

@sebrose http://cucumber.io

Formal test-driven development rules

1. You may only write production code to make a failing test pass
2. You may only write a minimally failing unit test
3. You may only write minimal code to pass the failing test

Test-driven development as a design process

"The act of writing a unit test is more an act of design and documentation than of verification. It closes a remarkable number of feedback loops, the least of which pertains to verification."

Advantages of test-driven development

- Clear place to start
- Iterative, agile design process
- Less wasted effort?
- Robust test suite, including regression tests

A test-driven development demo: Diamond Kata

- Given a letter, generate a diamond starting at ‘A’, with the given letter at the widest point.
  - e.g., `diamond('C')` would generate:

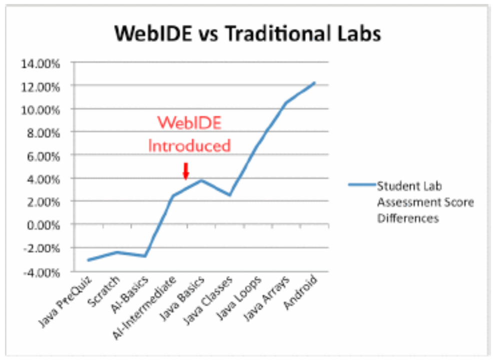
```
  A
 B B
C   C
 B B
  A
```

Formal test-driven development: Your impressions?

Empirical methods in software engineering

- How do we study the effectiveness of test-driven development compared to other methodologies?
  - Note: Mix of social and technical issues

Research on test-driven development (1/2)



- Hilton et al.: Students learn better when forced to write tests first
- Bhat et al.: At Microsoft, projects using TDD had greater than two times code quality, but 15% more upfront setup time
- George et al.: TDD passed 18% more test cases, but took 16% more time
- Scanniello et al.: Perceptions of TDD include: novices believe TDD improves productivity at the expense of internal quality

Research on test-driven development (2/2)

- Fucci et al.: Results: The Kruskal-Wallis tests did not show any significant difference between TDD and TLD in terms of testing effort (p-value = .27), external code quality (p-value = .82), and developers' productivity (p-value = .83).

- Fucci et al.: Conclusion: The claimed benefits of TDD may not be due to its distinctive test-first dynamic, but rather due to the fact that TDD-like processes encourage fine-grained, steady steps that improve focus and flow.

Today: Software engineering in practice

- SE empirical methods: Test-driven development case study
- Version and release management
  - Introduction to DevOps

Real-world software development challenges

- Imagine: You discover a bug in version 8.2.4 of your software
  - You want to discover, fix, and deploy updates to old versions
  - You want to fix the bug for new versions in ongoing development

Configuration management (CM)

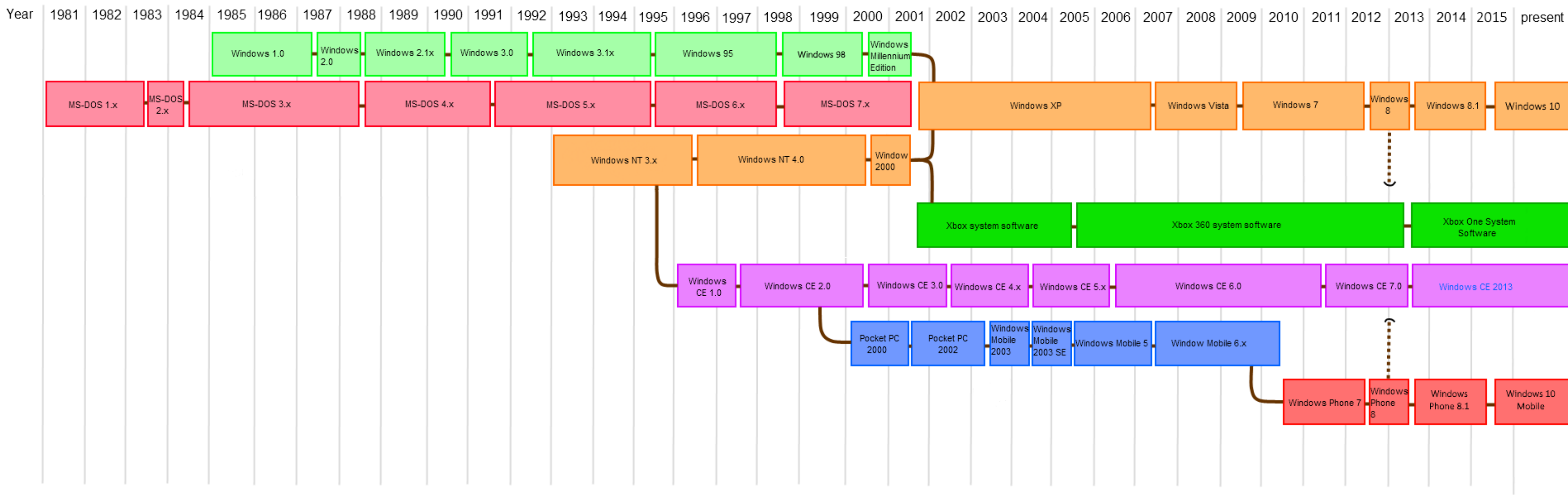
- Definition (Pressman): *Configuration management "is a set of tracking and control activities that are initiated when a software engineering projects begins and terminates when software is taken out of operation."*

Reasons for configuration management

- Software evolution
- Separate development
- Audits (legal, regulatory)
- Product lines
- Market variation (e.g., U.S., Europe, Asia)
- Platform variation (e.g., Android, iOS)

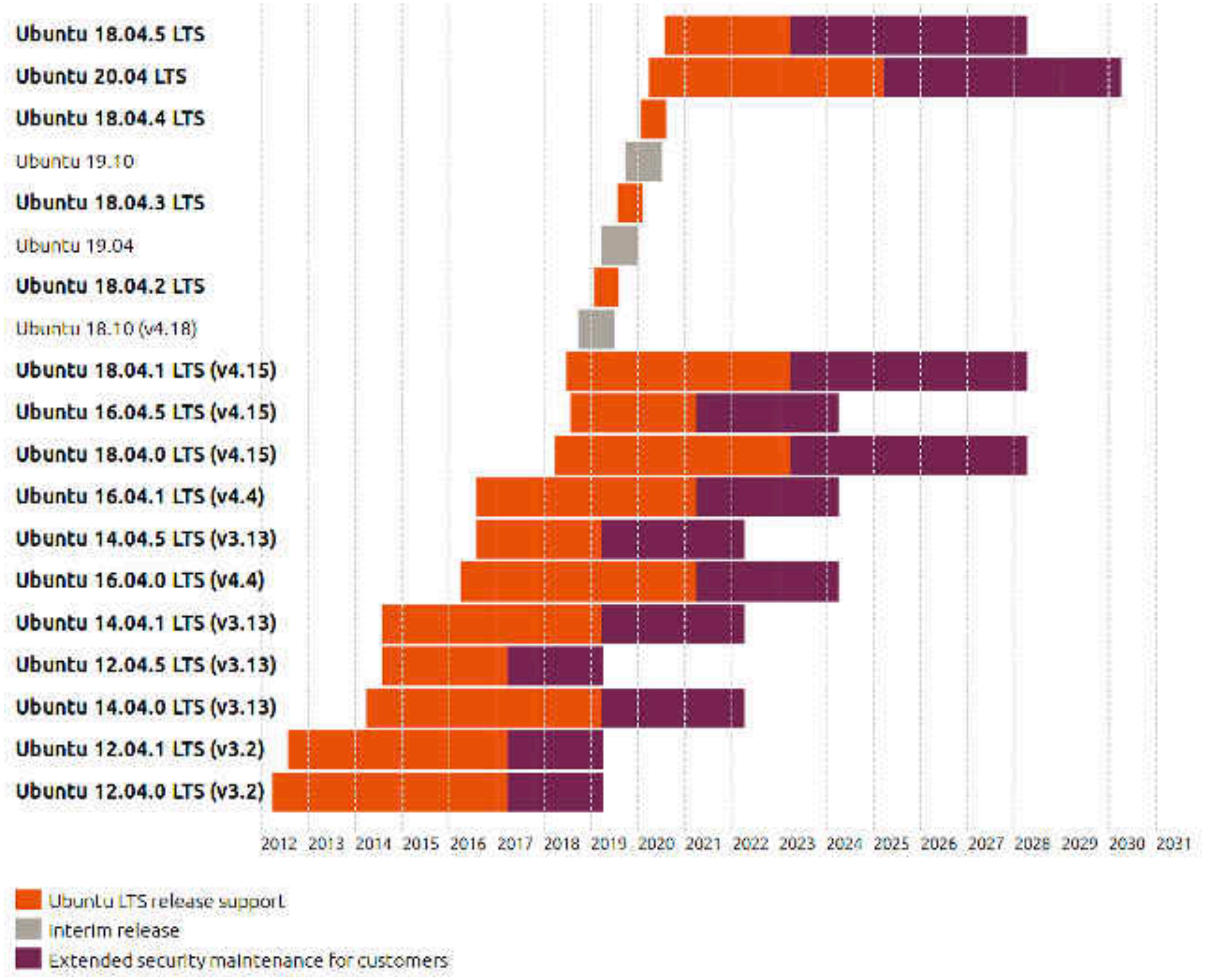
Consider: timelines of traditional software development

e.g., the Microsoft* OS development history



Source: By Paulire - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=46634740

Compare to the Ubuntu release cycle

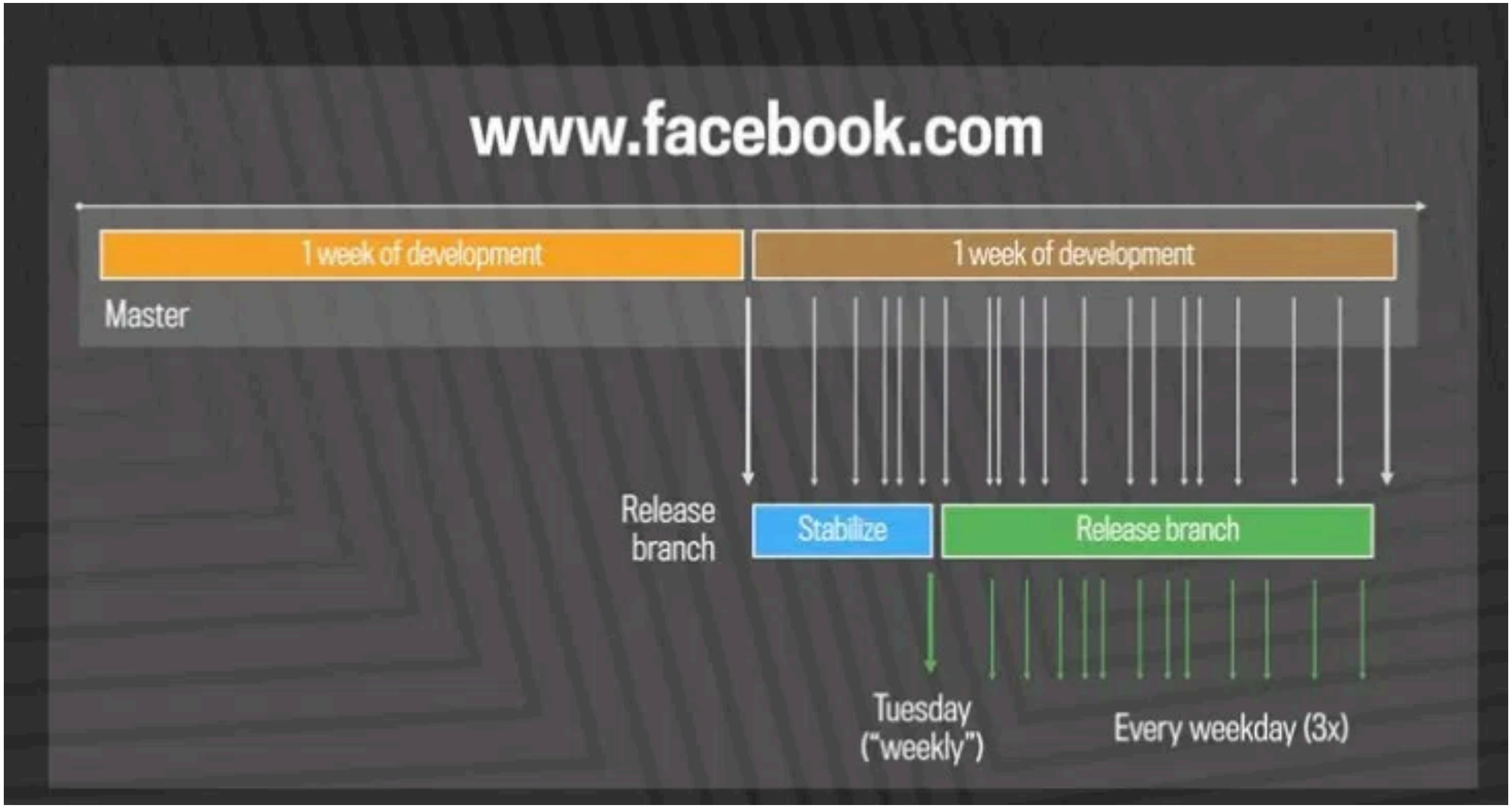


Aside: Semantic versioning for releases

- Given a version number MAJOR.MINOR.PATCH, increment the:
  - MAJOR version when you make incompatible API changes,
  - MINOR version when you add functionality in a backwards-compatible manner, and
  - PATCH version when you make backwards-compatible bug fixes.
- Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

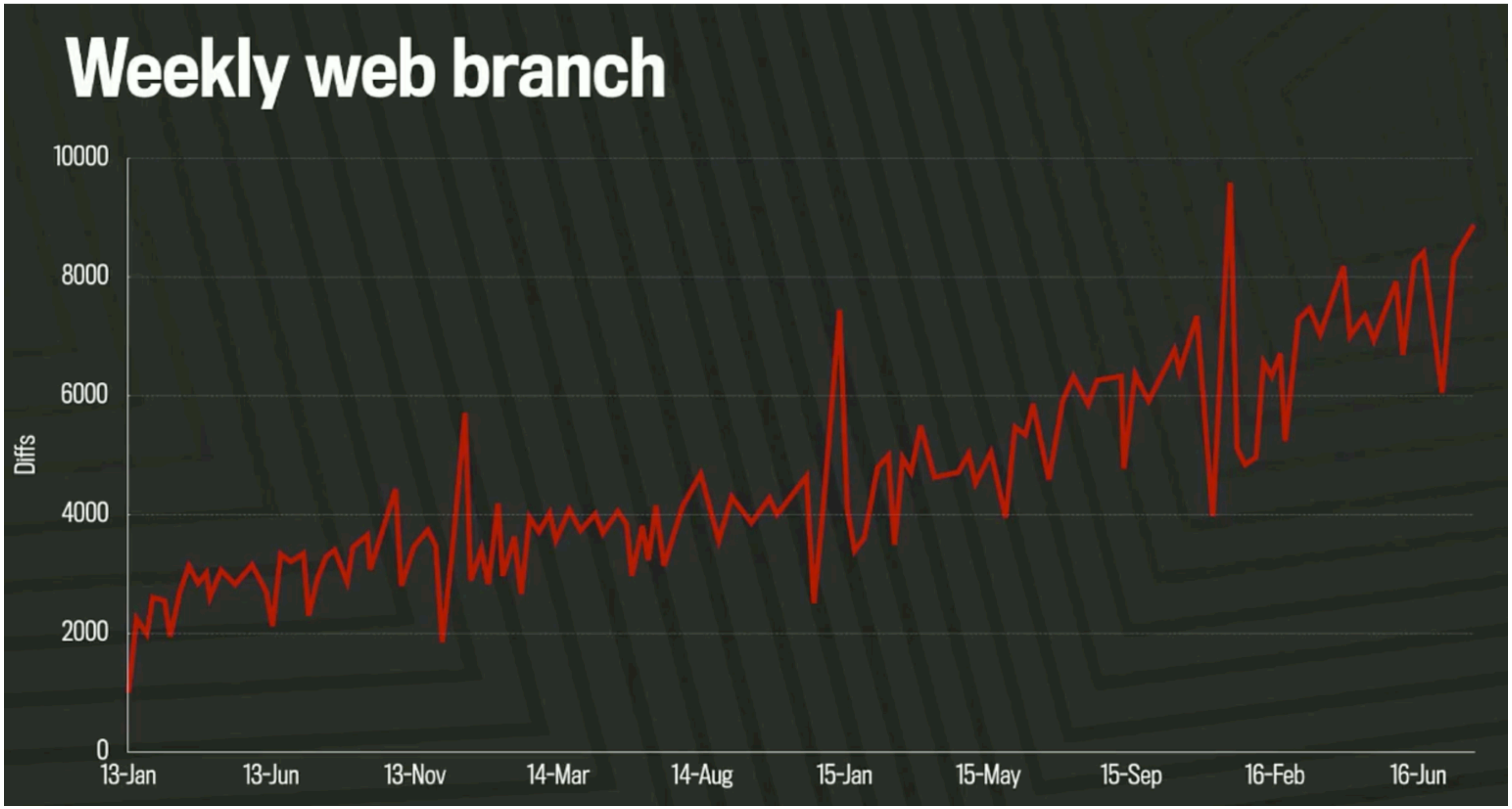
http://semver.org/

Compare to the (former) Facebook release cycle

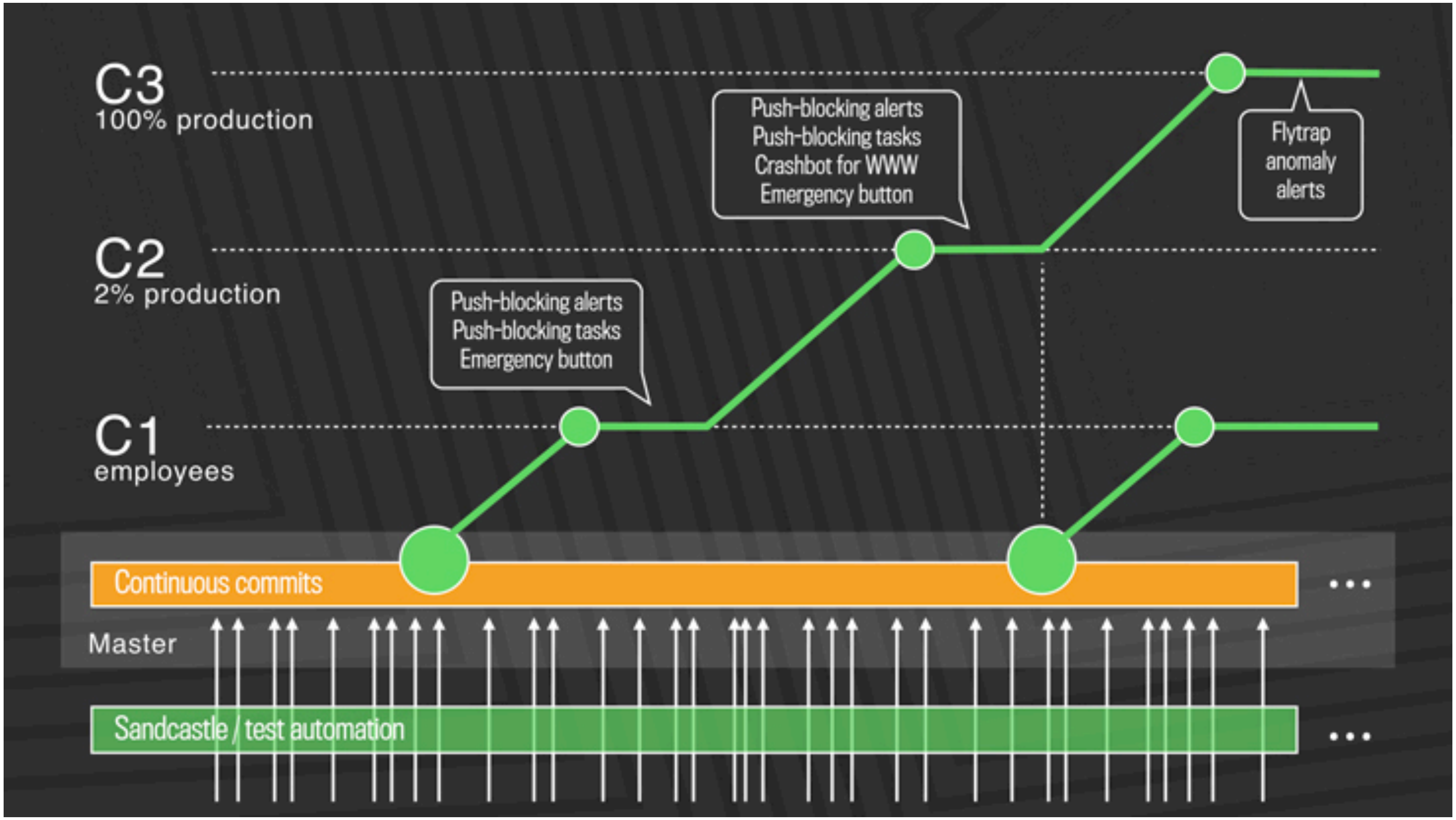


https://engineering.fb.com/web/rapid-release-at-massive-scale/

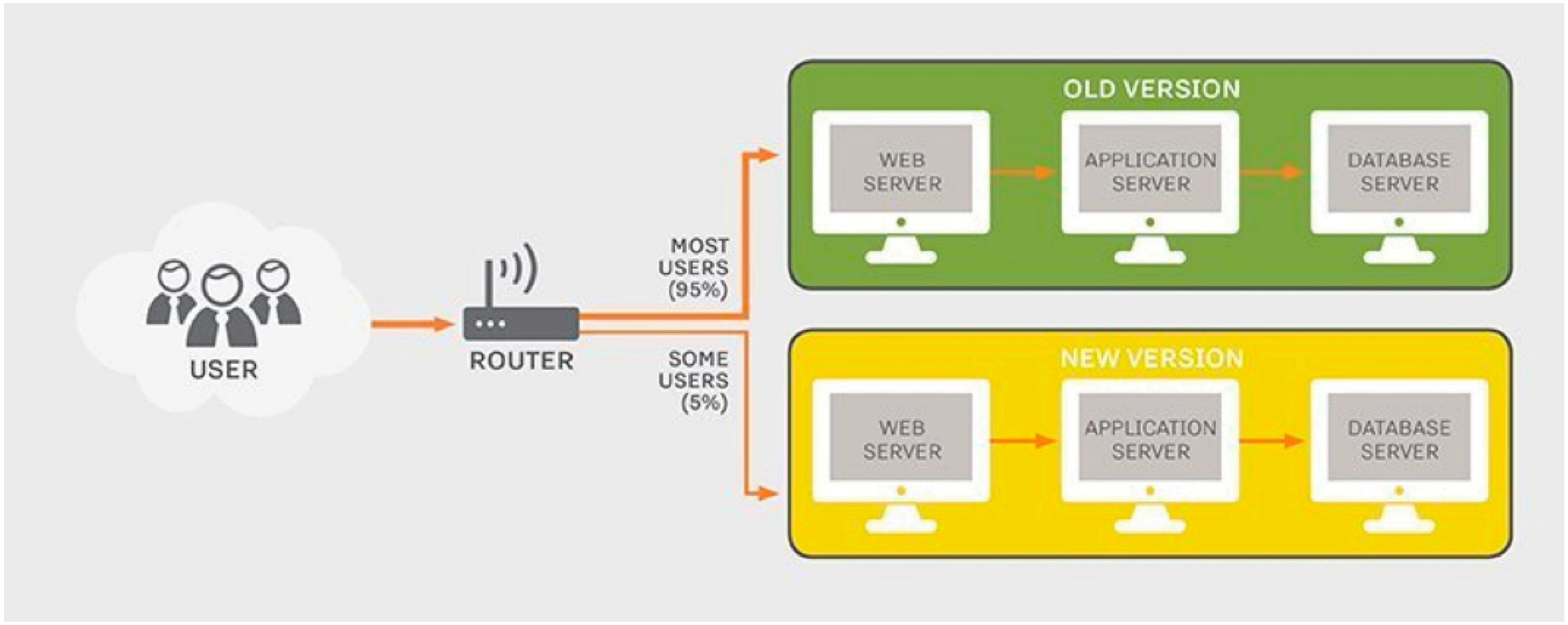
Number of commits/week became unsustainable



Modern Facebook release cycle (1000+ diffs / day)

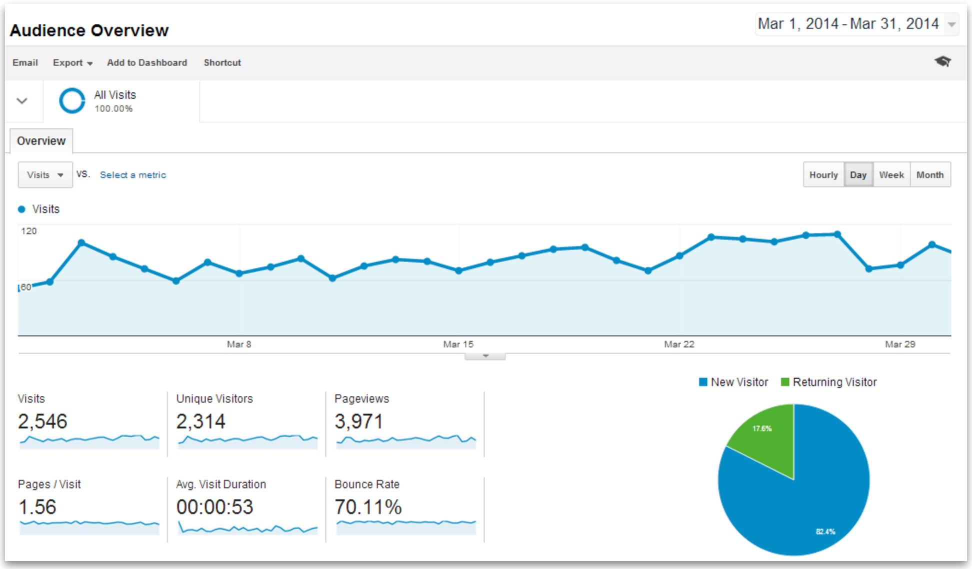


Aside: Canary testing



Aside: Dark launches

- Focuses on user response to frontend changes rather than performance of backend
- Measure user response via *metrics: engagement, adoption*



Visits	Unique Visitors	Pageviews
2,546	2,314	3,971

Pages / Visit	Avg. Visit Duration	Bounce Rate
1.56	00:00:53	70.11%

To be continued…