SPRING SUMMIT 2017

开放共享 原生共融

北京 | 2017年8月26日

Besides Spring What do we need?

Digital Sonic

Problems

you might meet

Problems

- you cant assume all your team members
  - are all experienced experts
  - match the same standard
  - wont make mistakes that happened before

Problems

- there will always be some infrastructures
  - created / modified by yourselves
  - not supported by Spring officially

Problems

- if you are not start from scratch
  - legacy systems are nightmares
- if you work in a company for years
  - systems will be corrupted
  - technologies will be outdated

dont be hesitate to solve those problems

Specifications

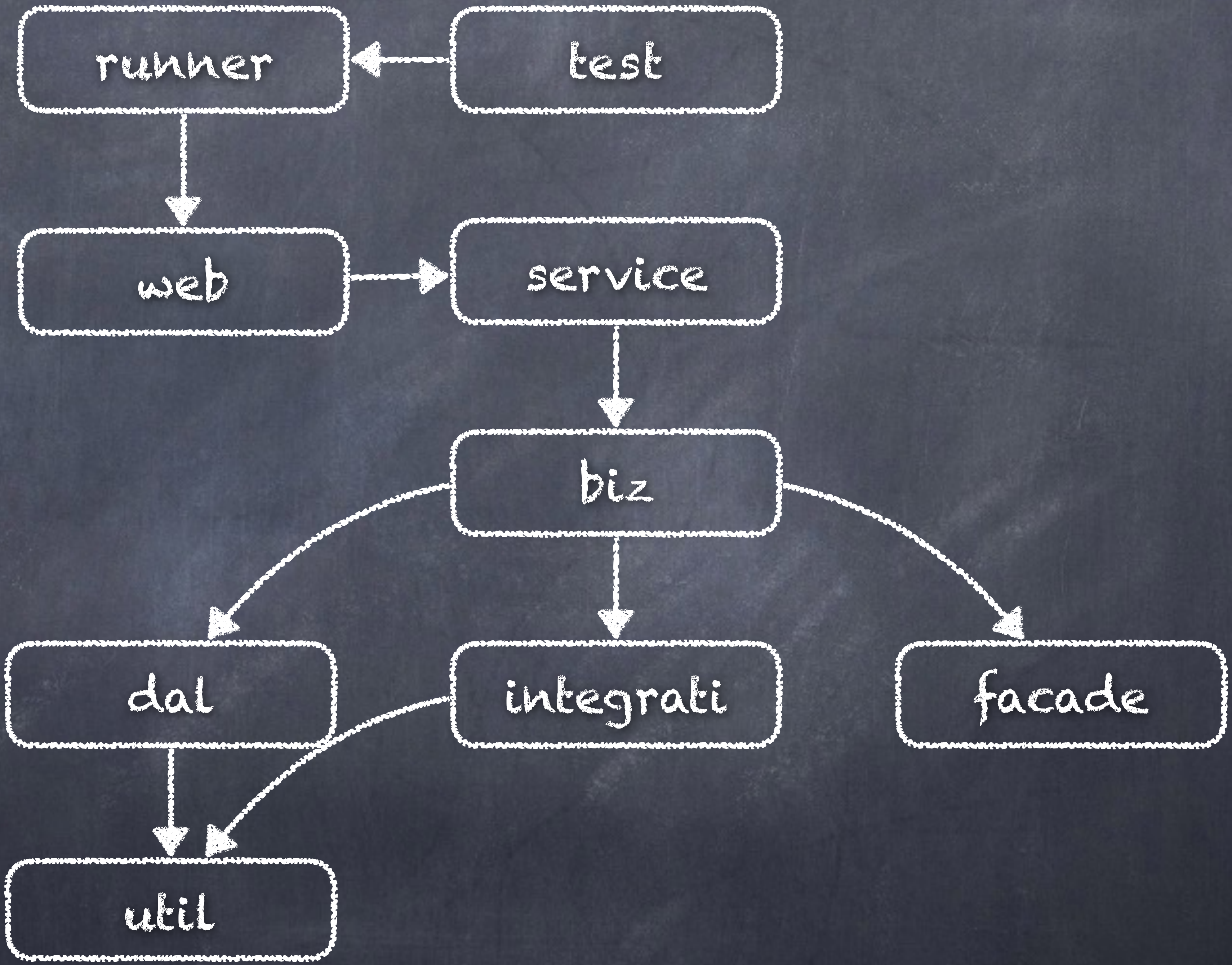
you have to make

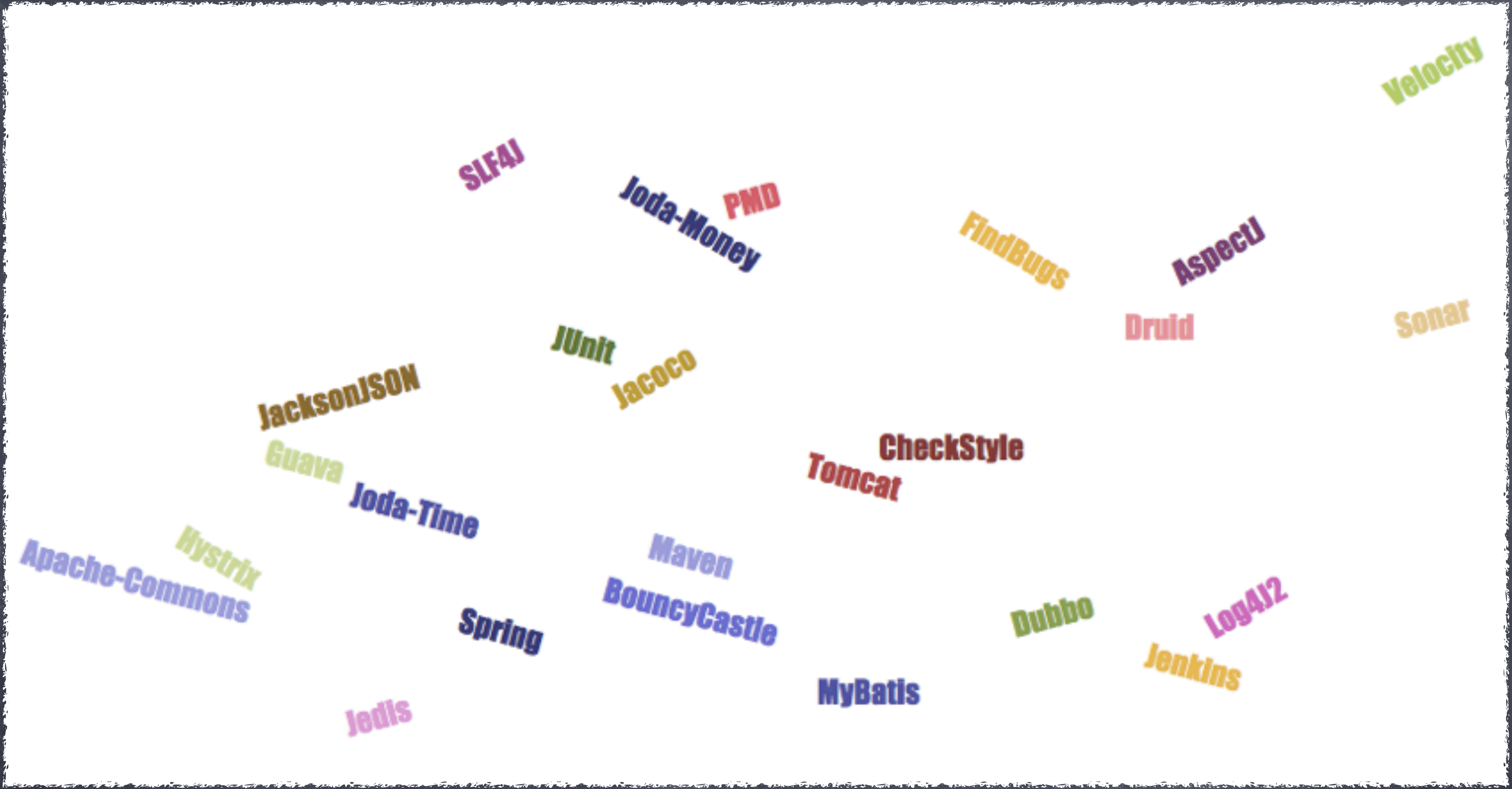
dont make everyone think

- too many choices of
  - technologies
  - methodologies
  - designs

Q: how to struct your projects?

A: create your own maven archetype like this





let few people make the call

log files

- different types of logs & structure-patterns
  - common logs
    - error
    - digest & performance
    - infrastructure
  - AOP and Config-Generator come to help

out-of-box

Quality Measurements

- Code Style
  - Go is better than Java at this point
  - Google Code Style (CheckStyle & plugins)
- Code Quality - static code scans
  - DIY your own FindBugs & PMD rules
  - Sonar is a good partner of Jenkins
  - Security scans should not be ignored

take advantages of all your stuffs

do you have

- infrastructures
  - famous but not supported by Spring official
    - eg. TAIR, CAT, Open-Falcon
  - create or patch by yourselves

Infrastructures

- prefer public clients, utilities and tools
- patch when the public one doesn't work
- make everything look like Spring stuffs
- contribute your codes when possible
- etc.

do you have

- common nonfunctional requirements
  - rate limit
  - data-sharding
  - degradation
  - etc.

nonfunctional requirements

- rate limit
  - with Zuul
    - Spring Cloud Zuul RateLimit is OK
  - without Zuul
    - we need a general rate limit solution
      - write your own with Guava RateLimiter
  - configurable thresholds are necessary

nonfunctional requirements

- data-sharding
  - not covered by Spring Data
  - Client Way
    - something like TDDL
  - Proxy Way
    - we prefer something like Cobar & MyCAT

Tests, Tests, Tests

Tests are easy with Spring

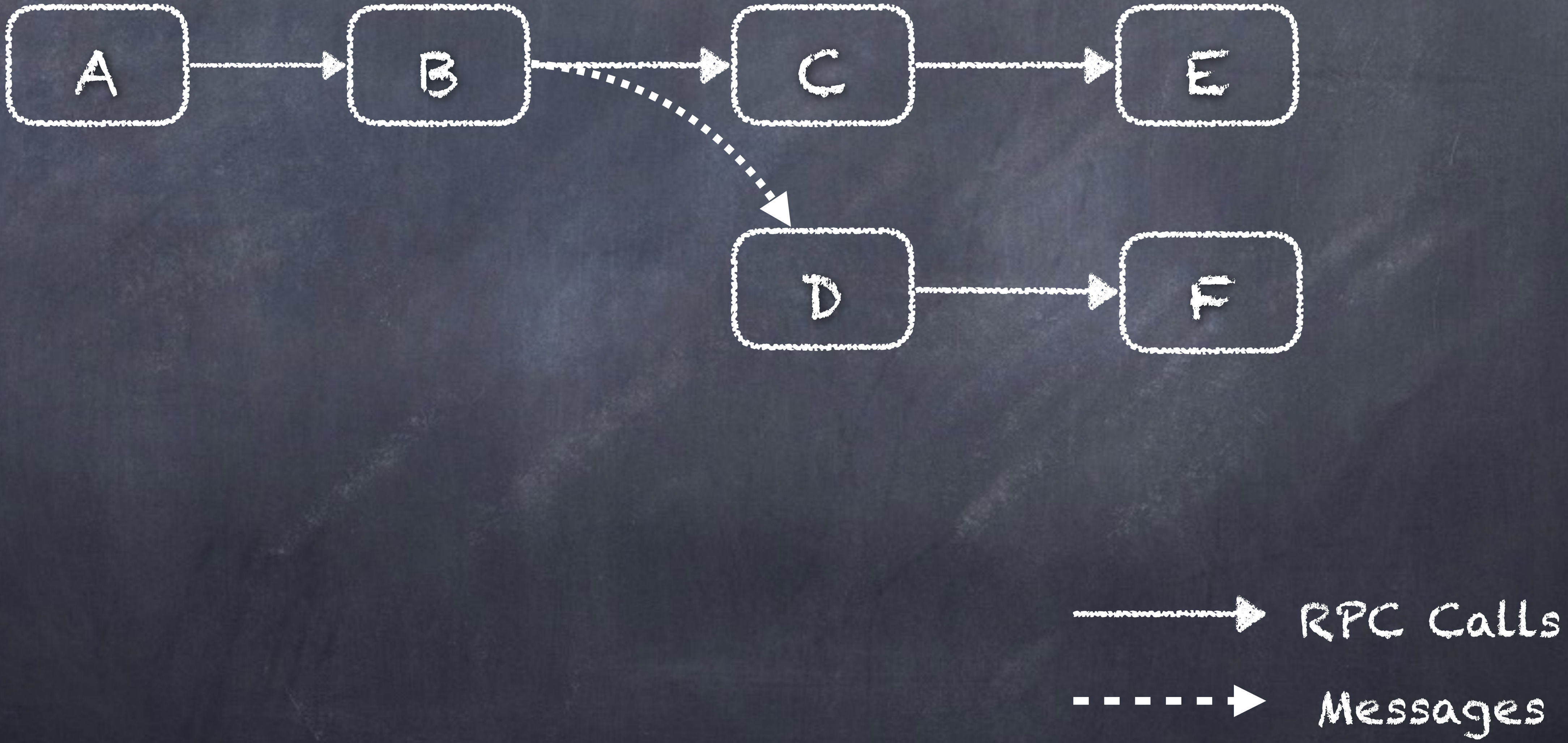
You mean Unit Tests? Maybe yes with mocks.

What if a service call needs to
go through dozens of systems to fulfill its task ...
Will you still think integration tests are easy?

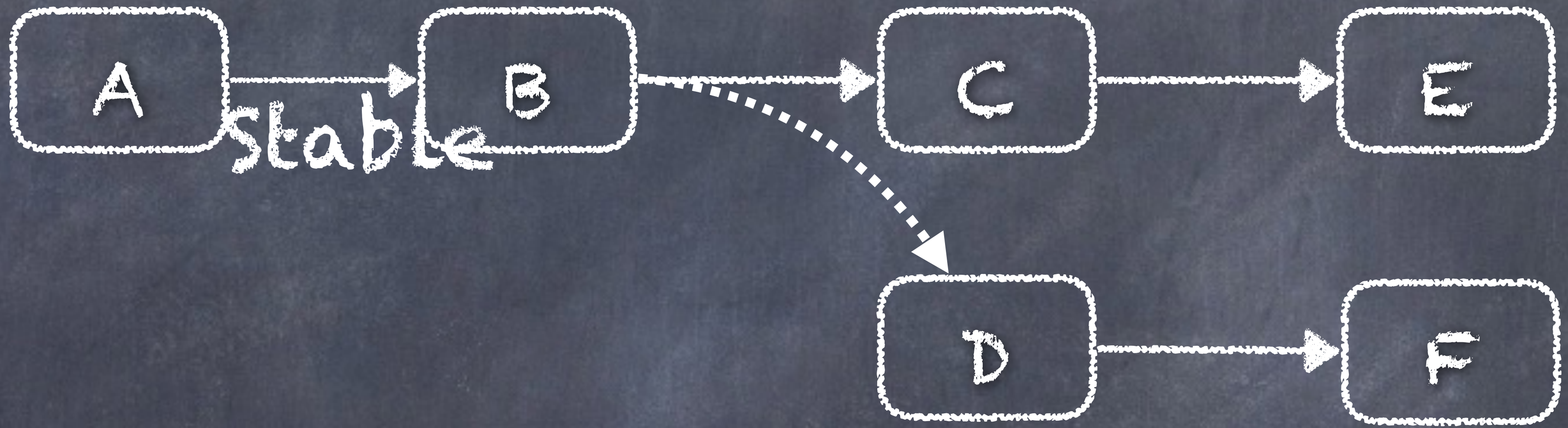
"Stable" Environment

- Provide an environment with
  - systems of product version
  - precise service route controls
  - careful maintains

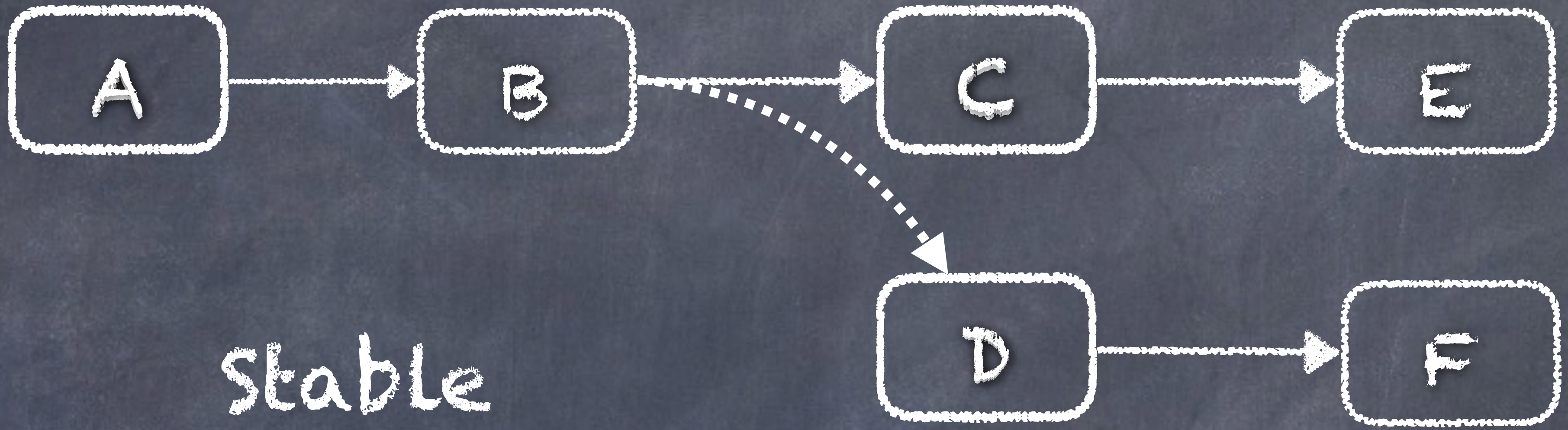
Demo



Demo



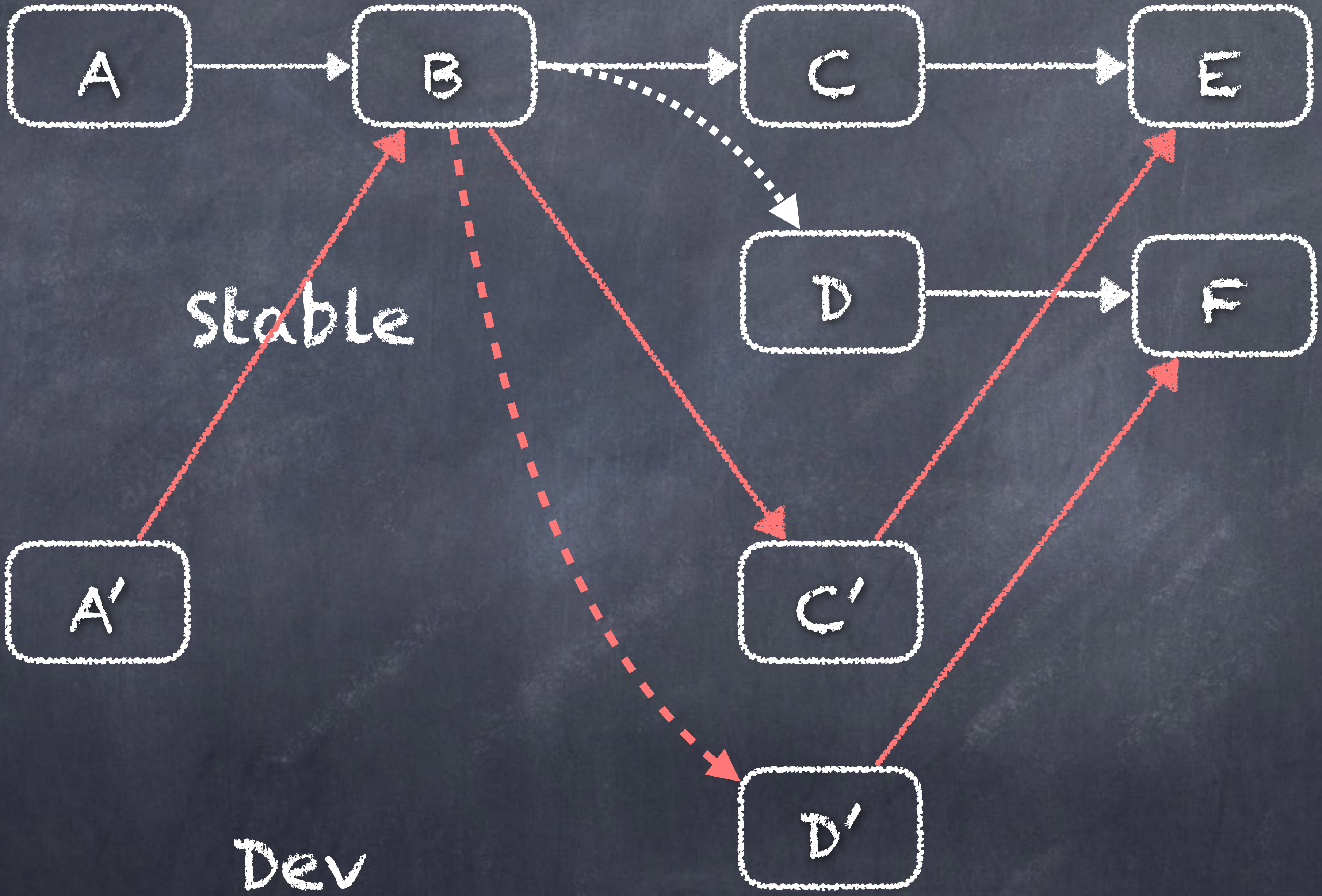
Demo



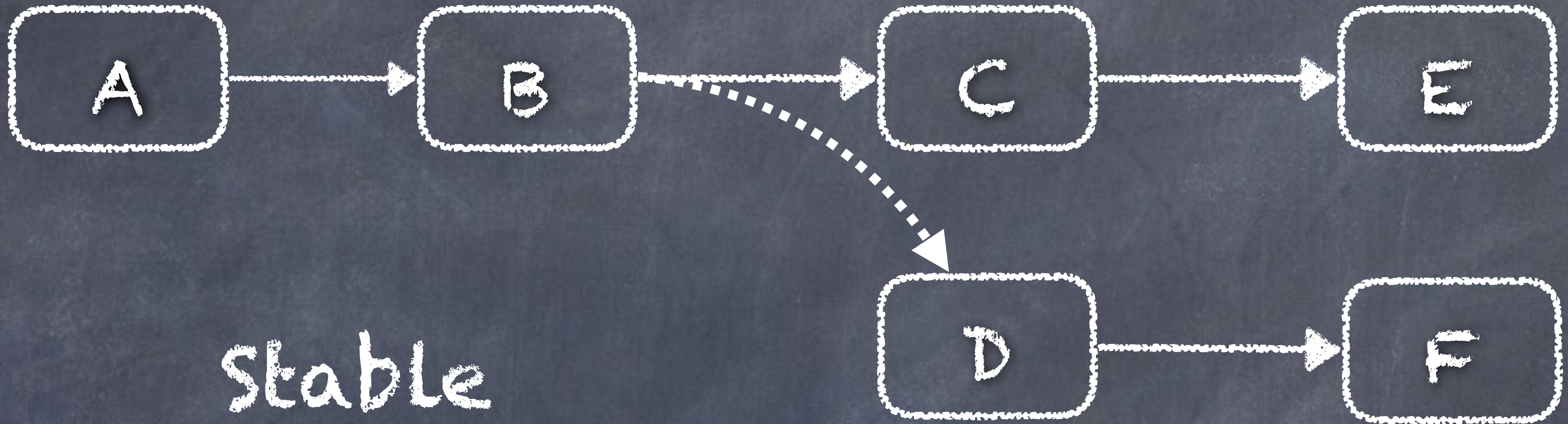
Stable

Dev

Demo


$$\begin{tikzcd}
A \arrow[r] & B \arrow[r] \arrow[dr, dashed, bend left] \arrow[ddr] \arrow[dddr, dashed] & C \arrow[r] & E \\
 & & D \arrow[r] & F \\
A' \arrow[uur] & & C' \arrow[uur] & \\
 & & D' \arrow[uur] &
\end{tikzcd}$$

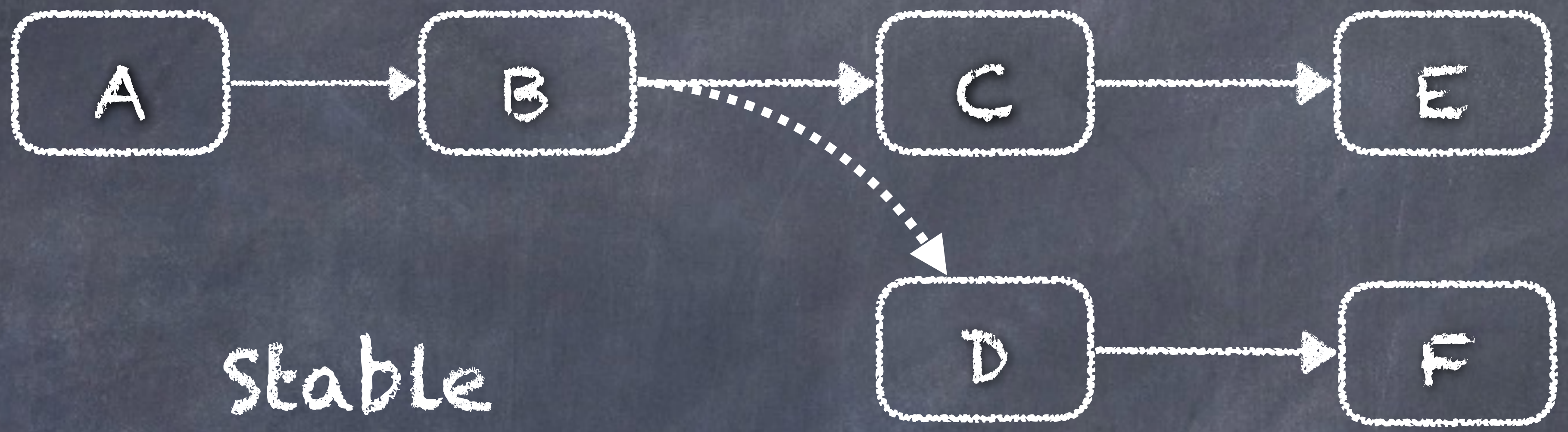
Demo



Stable

Dev 1

Demo



and more ...

Dev 1

Dev 2

Survive and Thrive
in a complex environment

Dont forget

- You have systems
  - running on Java 6 or earlier versions
  - using Spring 3.2.x or earlier versions
  - not modified for a long time
  - no one knows WTF it is

Legacy Systems

- Spring Boot is not an option
- Even Spring 4 is not an option
- Upgrade work is hard

- Just add some useful functions
- Not every system needs to be upgraded

make innovation acceptable

- think from your users' perspective
- dont force people to change
- dont work alone
- use the strategy of small wins
- top-down vs. bottom-up

Thanks!