UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION

CHAPTER 10 : Building Blocks of DevOps

DR FARIZUWANA AKMA
farizuwana@utar.edu.my

Topics

- Measurement and Metrics
- How to improve and accelerate software delivery

Measurement and Metrics

- A crucial aspect of software engineering is measuring what you are doing.
  - Traditional approach
  - Agile approach
  - DevOps approach

Traditional Use of Metrics

- Often driven by numbers
  - summarize and aggregate highly complex dependencies into single numbers, e.g.
  - cost effectiveness
  - capacity utilization
  - meeting target scope, etc.
  - “Manage by the numbers”
- Metrics:
  - static code analysis
  - test coverage

Agile Approach to Metrics

- Require a disciplined approach to ensure that customer feedback, continuous testing, and iterative development actually lead to frequent deliveries of working, valuable software.
- Often discusses value instead of specific metrics
- Software must be shipped to customers to be valuable

Definition of Done (DoD)

- A checklist of valuable activities required to produce software.
- The primary reporting mechanism for Agile team members.
- Informed by reality.
- Eg. DoD for a
  - Feature: story or product backlog item
  - Sprint: collection of features developed within a sprint
  - Release: potentially shippable state

Release and Deployment

- **Release**
  - **is a specific software version that you make available to users.**
  - **is created by promoting a specific release candidate.**
- **Release Candidate**
  - **is more than an arbitrary version of the software.**
  - **already has fulfilled specific requirements (e.g., all tests run successfully).**

Release and Deployment

- **Deployment**
  - **Occurs if you install a release (or even a release candidate or a version) of your software into a particular environment.**

MTTR and MTTD

- Metrics commonly used by Operations Team
- Mean Time to Repair/Resolve (MTTR)
  - Time it takes to resolve a system or network issue (after a technician receives a trouble ticket) and restore normal operations following an incident
- Mean Time to Detect (MTTD)
  - The difference between the onset of the incident in production and its detection by the operations team
- A guaranteed MTTR is often defined as part of a service-level agreement (SLA)

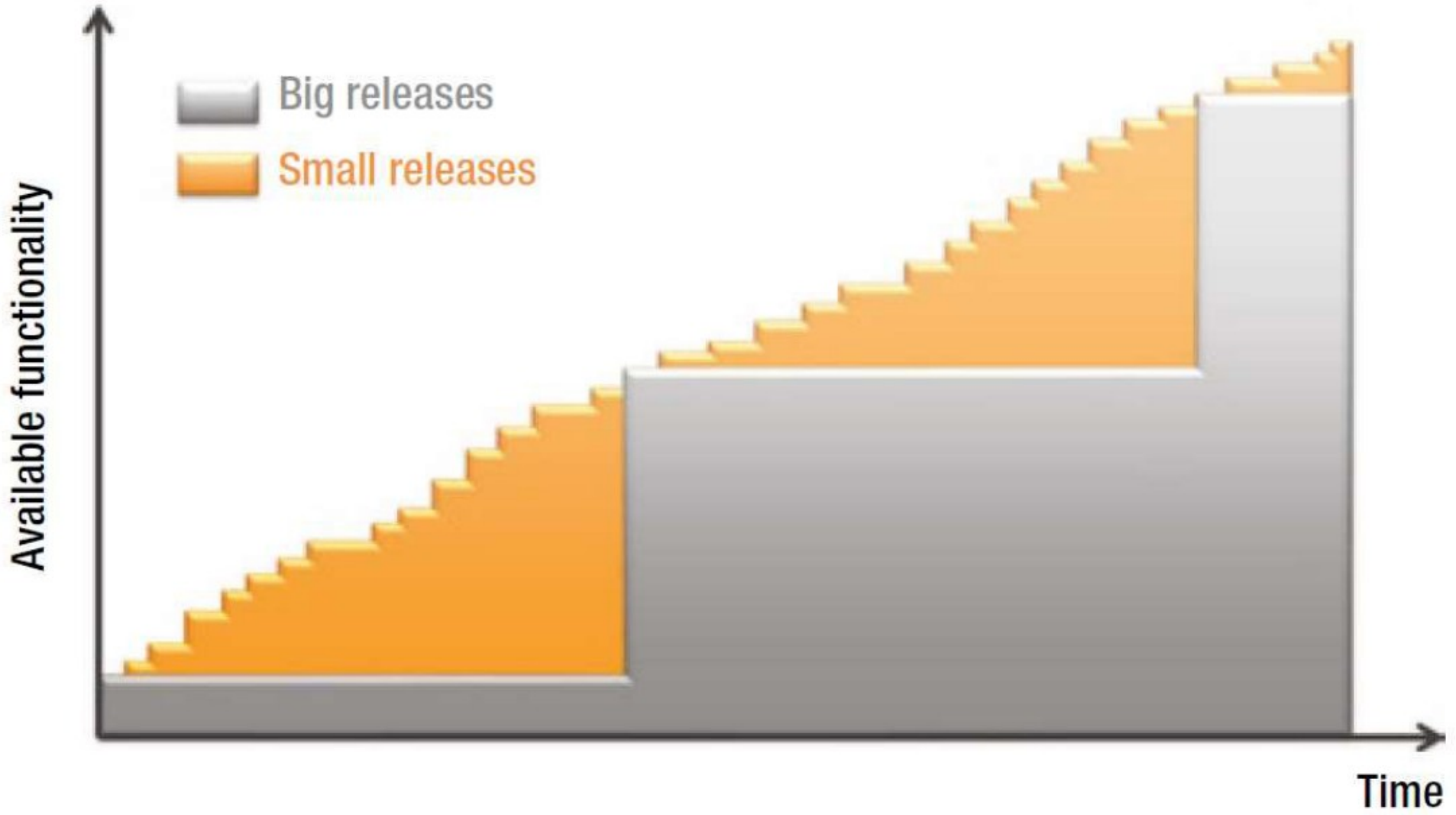
Improving Flow of Features

- DevOps is about
  - gaining fast feedback
  - decreasing the risk of releases
- Improve the flow of features from their inception to availability.
- To reduce batch size without changing capacity or demand.

Cycle Time

- The amount of time required from the start of the development process to the beginning of revenue generation.
- 3 phases:
  - Fuzzy Front End (FFE)
  - development cycle
  - time-to-volume

Improve and Accelerate Delivery



Automatic Releasing

- Reduce the risk of releasing software
- Increase efficiency
- Reproducible process that can be implemented by tool chains

June 2016

Programmer Automates His Job For 6 Years And Gets Fired In The End After Forgetting How To Code

Apply Releases Incrementally and Iteratively

- An iteration is a mini-project that may result in an increment of the software.
- Iterating starts with an idea of what is wanted, and the code is refined to get the desired result.
- An increment is a small unit of functionality.
- Incrementing allows you to build a better understanding of what you need, assembling the software piece by piece

Decoupled Deployment and Release

- A technique to improve and accelerate delivery
- Branch by Abstraction
- Feature Toggles
- Dark Launching
- Blue-Green Deployment

Branch by Abstraction

- A pattern for making large-scale changes to your application incrementally on mainline
- Main steps for implementing the strategy include:

1)Create an abstraction over the part of the system that you need to change.

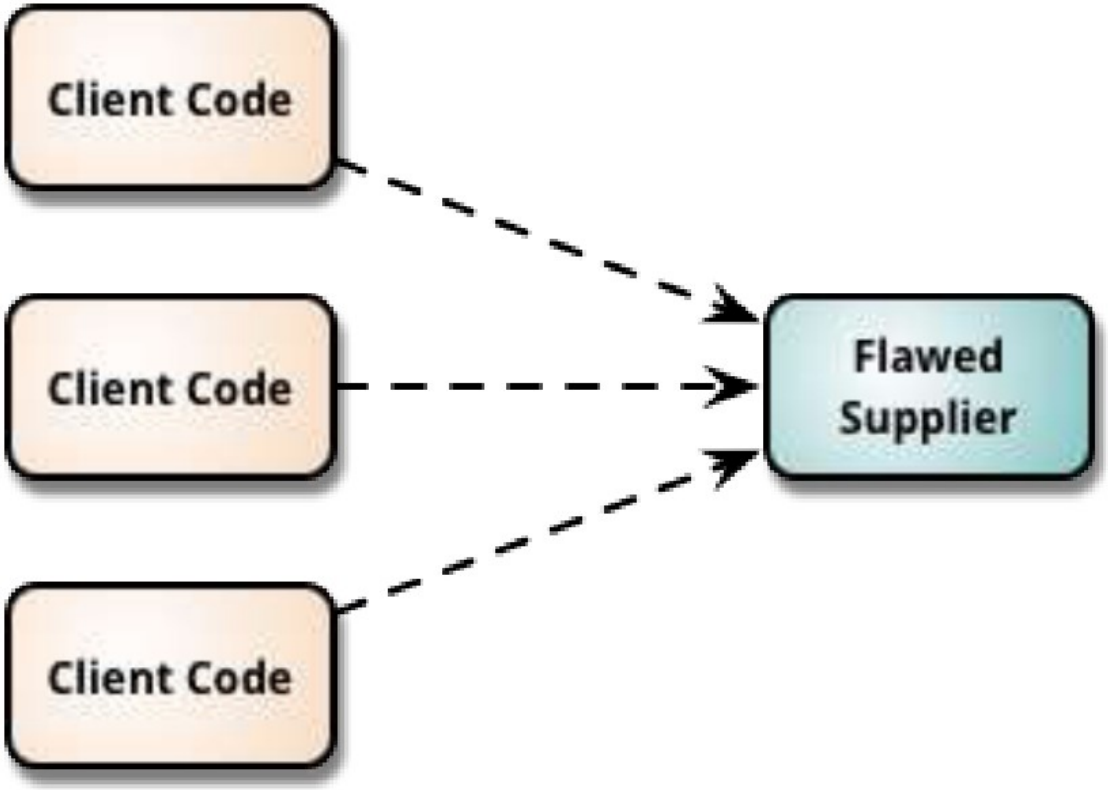
2)Refactor the rest of the system to use the abstraction layer.

Branch by Abstraction

1) Continue coding; the abstraction layer delegates to the old or new code, as required.
2) Remove the old implementation.
3) Iterate over steps 3 and 4. Ship your system in the meantime.
4) Once the old implementation has been completely replaced, remove the abstraction layer.

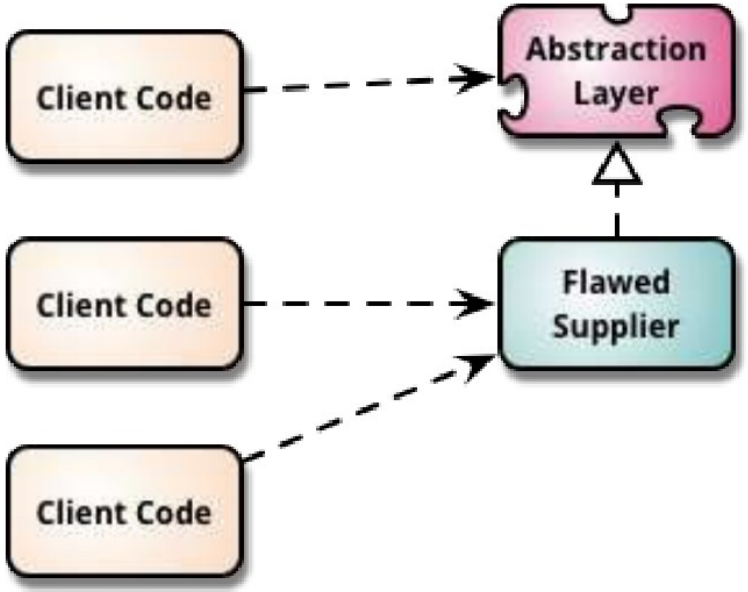
Branch by Abstraction

- Assume a situation where various parts of the software system are dependent on a module, library, or framework that we wish to replace



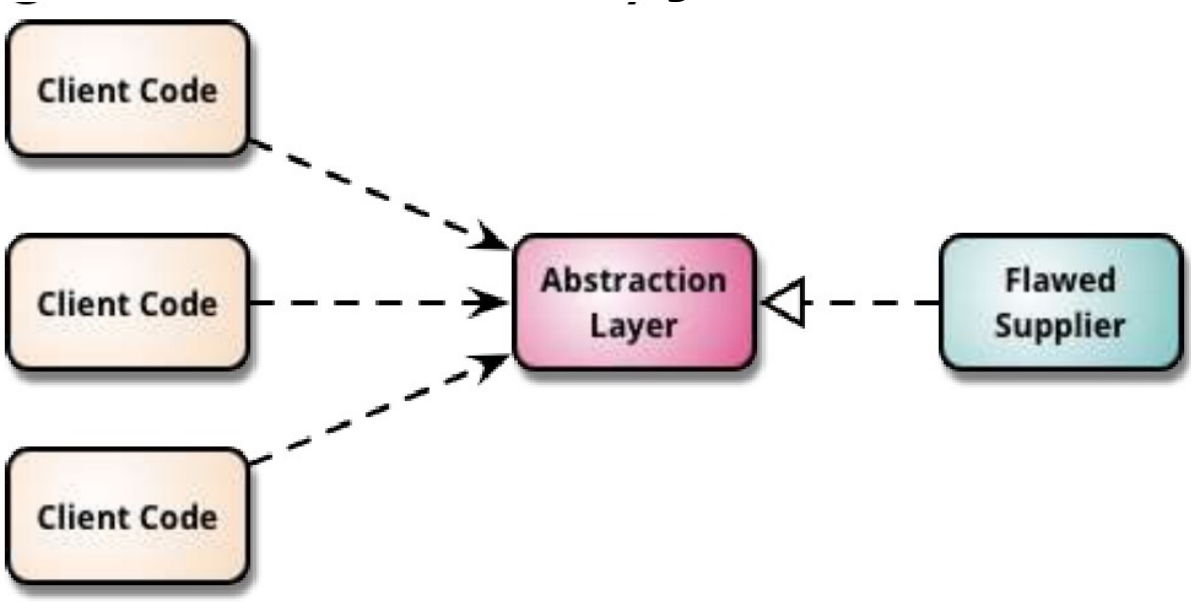
Branch by Abstraction

- Create an abstraction layer that captures the interaction between one section of the client code and the current supplier
- Change that section of the client code to call the supplier entirely through this abstraction layer



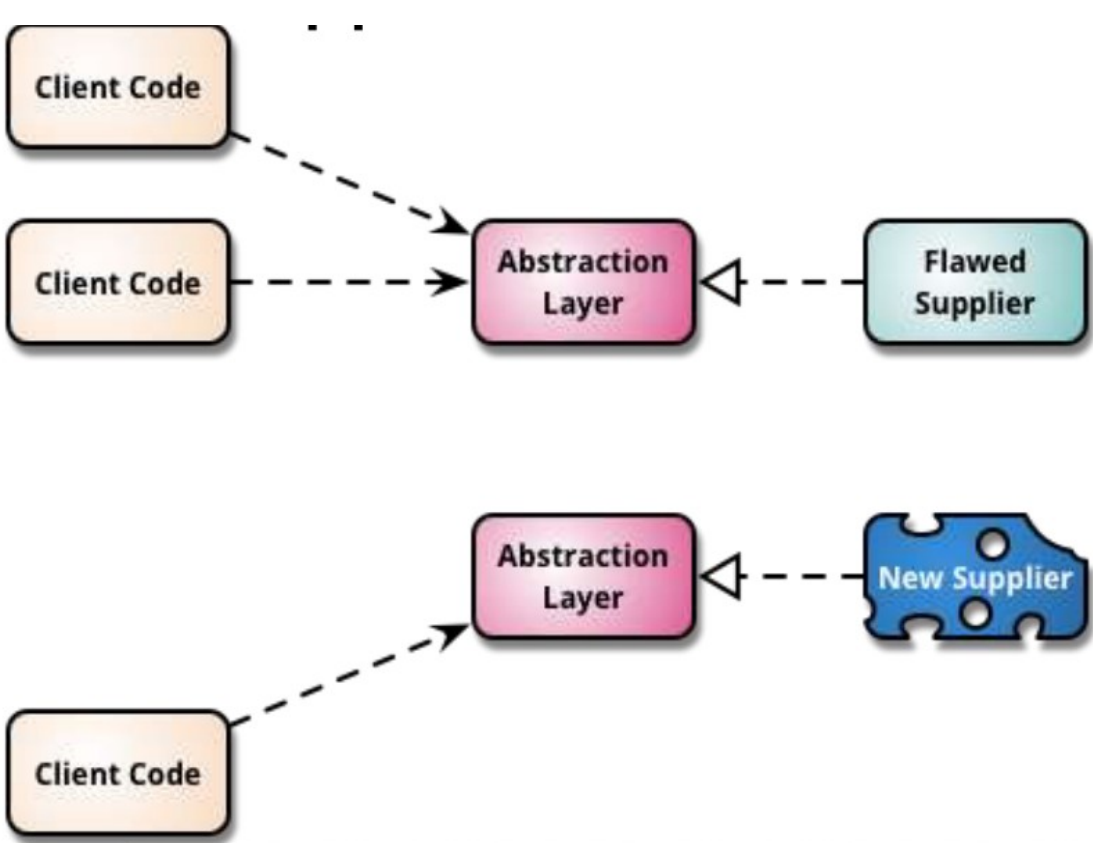
Branch by Abstraction

- Gradually move all client code over to use the abstraction layer until all interaction with the supplier is done by the abstraction layer
- Improve the unit test coverage of the supplier through this abstraction layer



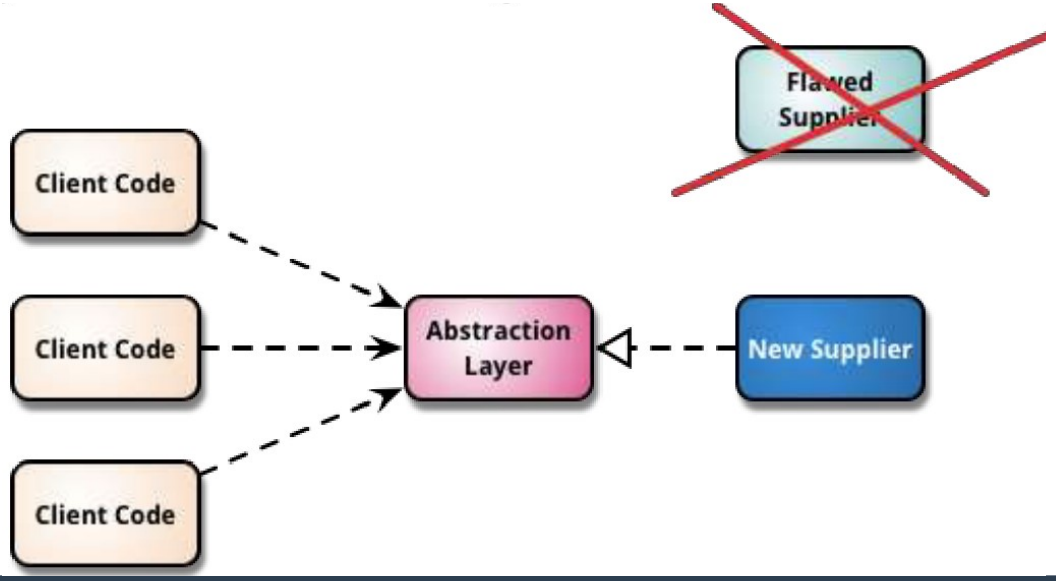
Branch by Abstraction

- Build a new supplier that implements the features required by one part of the client code using the same abstraction layer.
- Once ready, switch that section of the client code to use the new supplier.



Branch by Abstraction

- Gradually swap out the flawed supplier until all the client code uses the new supplier.
- Once the flawed supplier isn't needed, delete it.
- May choose to delete the abstraction layer once it is no longer needed for migration.



Feature Toggles

- Deliver complete code to production but use data-driven switches to decide which feature is made available during runtime.
- To enable data-driven switches, configuration files are often used.
- With this, the team can develop on the same development mainline and ship the complete code to production.

Dark Launching

- Strategy of deploying first version of functionality into production before releasing the functionality to all users.
- Can deploy new versions of the software continuously, regardless of which features are available to which users.
- One can find any bugs more easily, before making the release available to all users.

Dark Launching

- It provides an approach to remediate in a low-risk way.
- If problems occur with an early version of a feature, only a few users may experience the problem.
- Additionally, you can address the incident without a complete heavyweight rollback, just by switching off the feature (e.g., feature toggle) or by changing a router setting.
- Can be combined with blue-green deployments.

Blue-Green Deployment

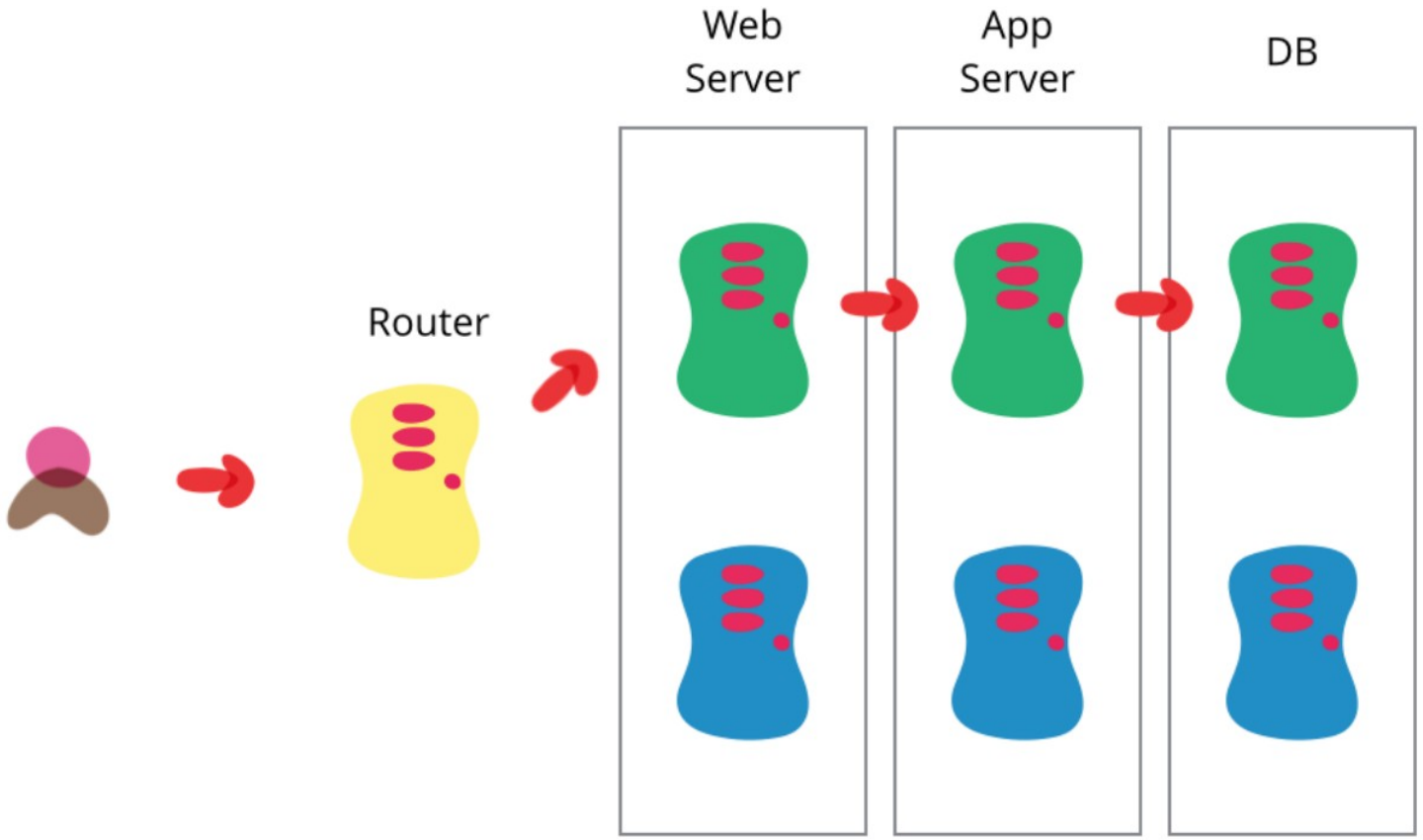
- This strategy deploy the new version of the application side by side with the old version.
- To switch over to the new version or roll back to the old version, back and forth, we merely have to change a load balancer or router setting.
- Blue-green deployment ensures that you have two production environments that are as similar as possible.

Blue-Green Deployment

- At any one time, one of them (e.g., the green environment) is live. While bringing a new release of your software to production, you conduct the final steps of testing in the blue environment.
- Once the software is working in the blue environment as expected, configurations are done, and smoke tests are run successfully, we switch the router to redirect all incoming requests to go to the blue environment.

Blue-Green Deployment

- **Afterward, the green environment is out of production and can be used to prepare the next release.**



Canary Release

- A variation on blue-green deployment that can be applied when running a cluster of servers
- Incremental deployment rather than to the whole cluster of servers.

Canary Release

- The new release first goes to a subset of production systems, to which only specific users or a closed user group (e.g., employees) are routed.
- If this deployment is successful, the release can be deployed to more users, or finally to all available users.

END OF LECTURE 12