Testing in the context of Microservice Architectures

Marlon Pierce, Suresh Marru

CSCI-B 649 Science Gateway Architectures

Lot of these principles are inspired by Martin Fowler Talks.

Refer to them for first hand opinions.

Microservices

- Flexibility is worth the complexity only if you can put the flexibility to good use.
  - well scoped functionality into “micro services”.
  - Scale them independently.
  - well defined component level API’s
  - use “ circuit breaker pattern”

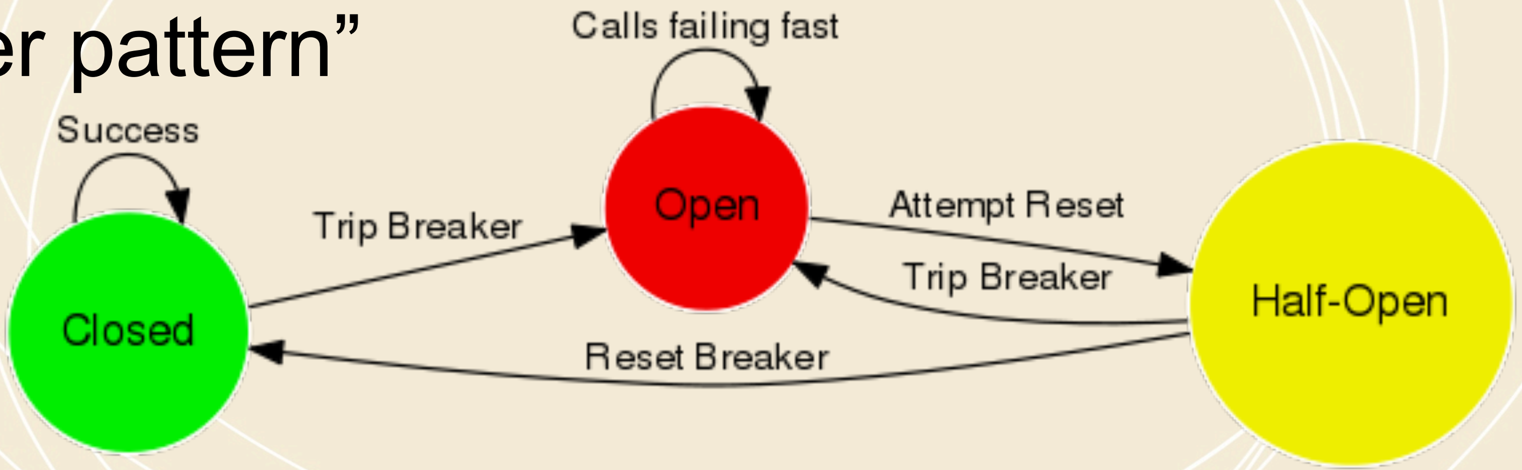
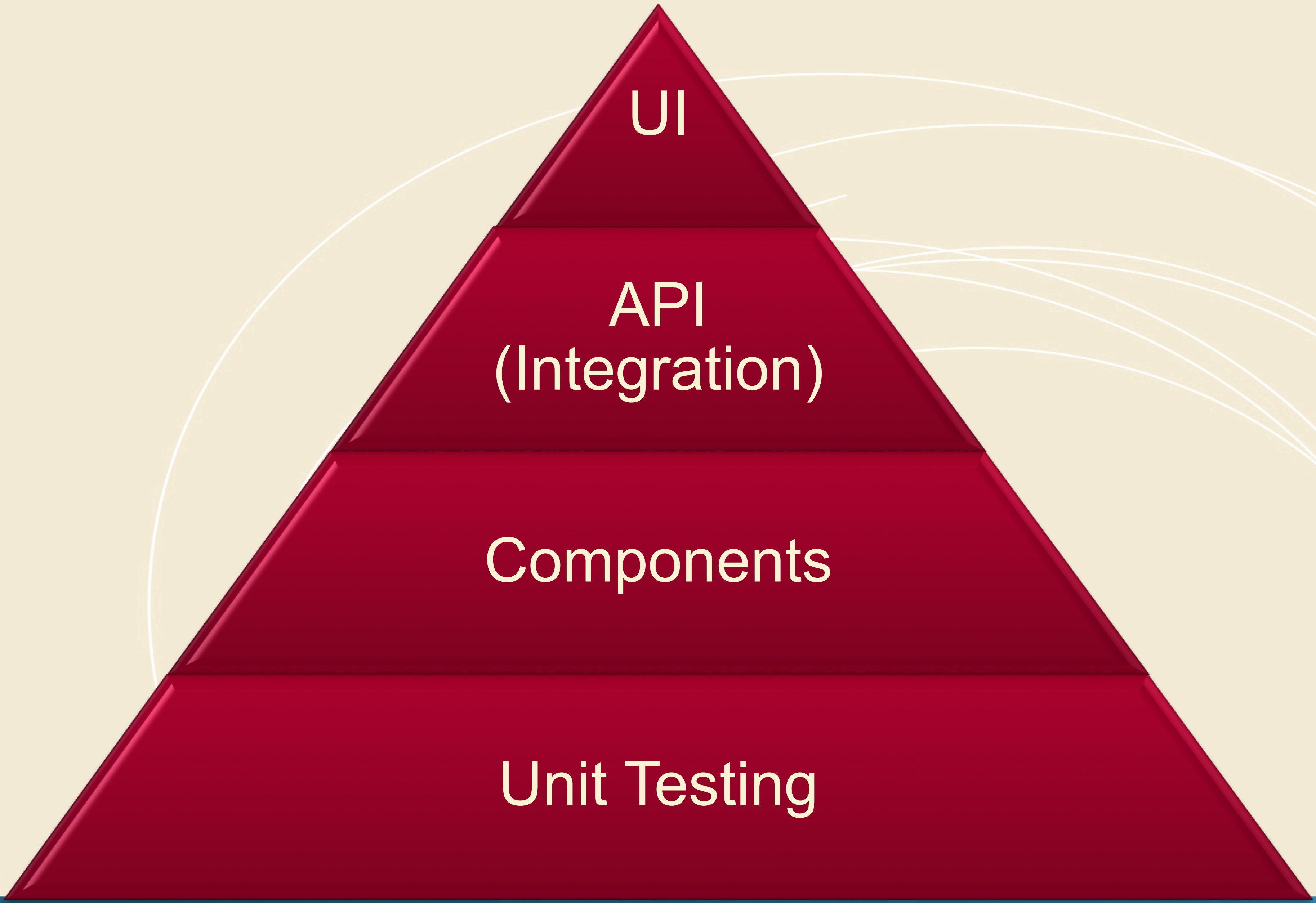


Figure Source: akka.io

Todays Outline

- Layers of Testing
- What is different for micro service architecture
- Capacity Testing
- Milestone 2 Testing Goals and Expectations


UI
API (Integration)
Components
Unit Testing

Testing Levels

- Unit Testing
  - lowest level testing to ensure implementation behaves as expected. Best to use “Test Driven Development”.
- Component Testing
  - isolates each micro service functionality from larger system behaviours.
- Integration Testing (API for this course)
  - ensure interactions between component interfaces.

Testing Levels Contd..

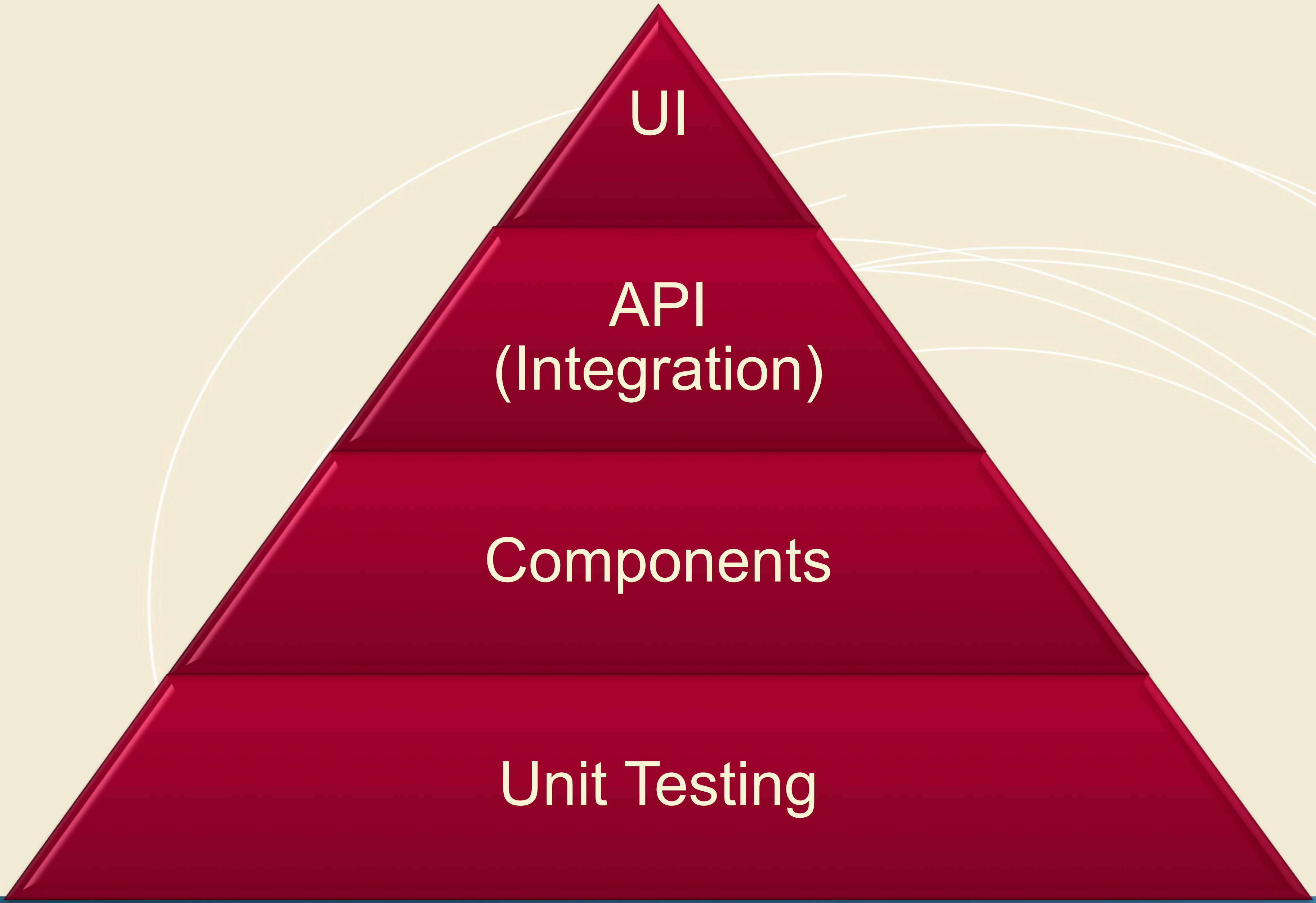
- UI Testing (covers end to end scenarios)
  - ensures the overall goals of the system are met and remain interact with evolutions.
- How to detect the unknowns?
  - understand the limits of the system?
  - predict surprises?

What is testing different with Micro Services?

- Smaller functionality exposes interactions previously hidden.
  - testing at these well defined boundaries exposes well scoped vulnerabilities.
- Not all services have the same critical function within the system.
  - prioritize testing based on their importance.

Microservice Testing Contd..

- Spend less time on UI (end to end) testing
  - comprehensive lower level testing should expose lot of system behaviors.
  - they do not accurately test asynchronous operations.
  - test to ensure services are working in “harmony”
- Use “infrastructure as a code”
  - everything should be able to reproducible programmatically.


UI
API (Integration)
Components
Unit Testing

Testing Granularity

- The complexity increases as you go from bottom to top of the pyramid.
- The pyramid illustrates the number of tests required at each stage.
- Testing frameworks help with what we know and ensuring they don’t break.
- Significant effort have to be put in writing tests to “learn about the system”.

Project Milestone 2 Testing modules

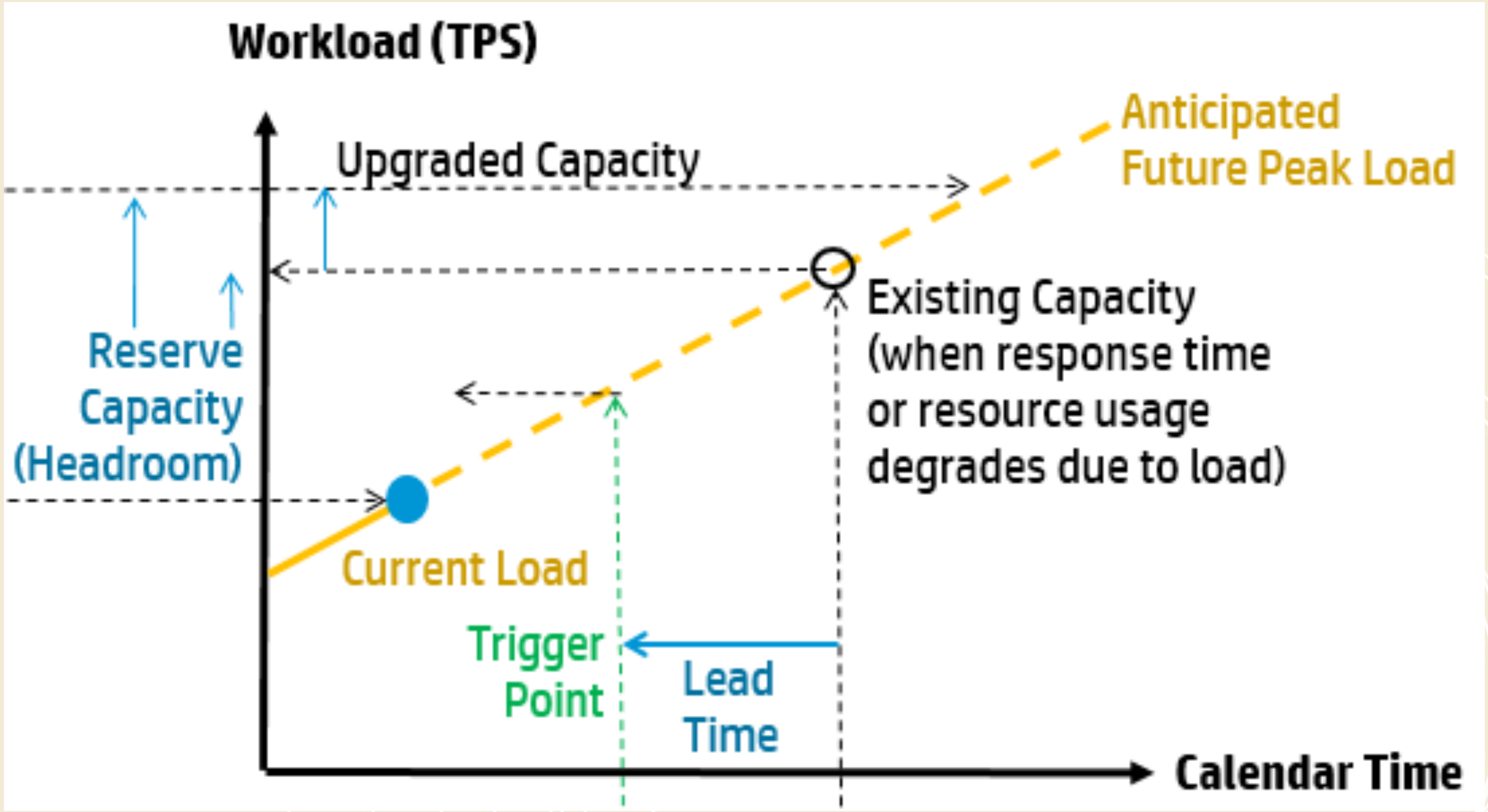
- Unit testing is a always a good practice.
- Start writing individual component level test cases.
- Write API level test cases.
- Understand your system
    - Make it break (modify implementations to keep busy within a request).
    - Plot the testing metrics
    - Identity “operating range”.

Capacity Testing

- Simulate destructive behavior at system level
  - Netflix Chaos Monkey
- Load test individual services
- Remember 80/20 rule (Pareto Principle)
  - 80% of the effects are derived from 20% of causes.
- Test early and often

Capacity Testing Contd..

- Find out how many calls per second can a service receive before it deviates form normal performance.
- Write more and small tests.
- Monitor services
- Log detailed analytics


Workload (TPS)
Upgraded Capacity
Anticipated Future Peak Load
Reserve Capacity (Headroom)
Existing Capacity (when response time or resource usage degrades due to load)
Current Load
Trigger Point
Lead Time
Calendar Time

Source: http://www.wilsonmar.com/perftest.htm

Thank You!

Marlon Pierce, Suresh Marru

{marpierc, smarru}@iu.edu