The Term Project (30 Points)

- Form a team of 4 to 6 people.
- Choose one of the following options:
  - **Option 1**: Build a web server using Java, or any language of your choice.
    - Basic requirements: **HTTP** (GET, HEAD & POST), **logging** & **multi-threading**
    - At least 2 optional features such as authentication/authorization, bandwidth throttling, SSL, etc. (See more choices on the next slide.)
    - NIO can count as a replacement for multi-threading
  - **Option 2**: Build a web server using Java and the **HttpCore API** (See Slide 3) **.** Same requirements as Option 1.
  - **Option 3**: Build a **web site** (See Slide 4), using ***XAMPP*** (http://www.apachefriends.org), or ***Apache Tomcat*** + Java Servlets/JSP
- Feel free to be creative and suggest other possibilities
- Project proposals due **November 1** (See the last slide for requirements)

Features That Could Be Added

- There is a Web server program in Textbook #2, called **JHTTP.java,** which you are allowed to reuse in your term project. (But you are not required to use it.) Some of the features that could be added to that program include:
  - Support for other request methods, such as POST, HEAD, etc. (Required)
  - Authentication/Authorization
  - SSL (Secure Socket Layers): Using **SSLSocket** & **SSLServerSocket** classes (Read Chap. 10 of Textbook #2, if interested in this feature.)
  - Cache requests – keeping track of requests you've received and store the data from the most frequently requested files in a **Map** so that they're kept in memory.
  - A server administration interface (UI)
  - Support for multiple document roots so individual users can have their own sites
  - Use non-blocking I/O and channels instead of threads and streams. (Read Chapter 5 of Textbook #1, if you are interested in this feature.)
- Note: **JHTTP.java** uses **RequestProcessor.java**, so you need both.

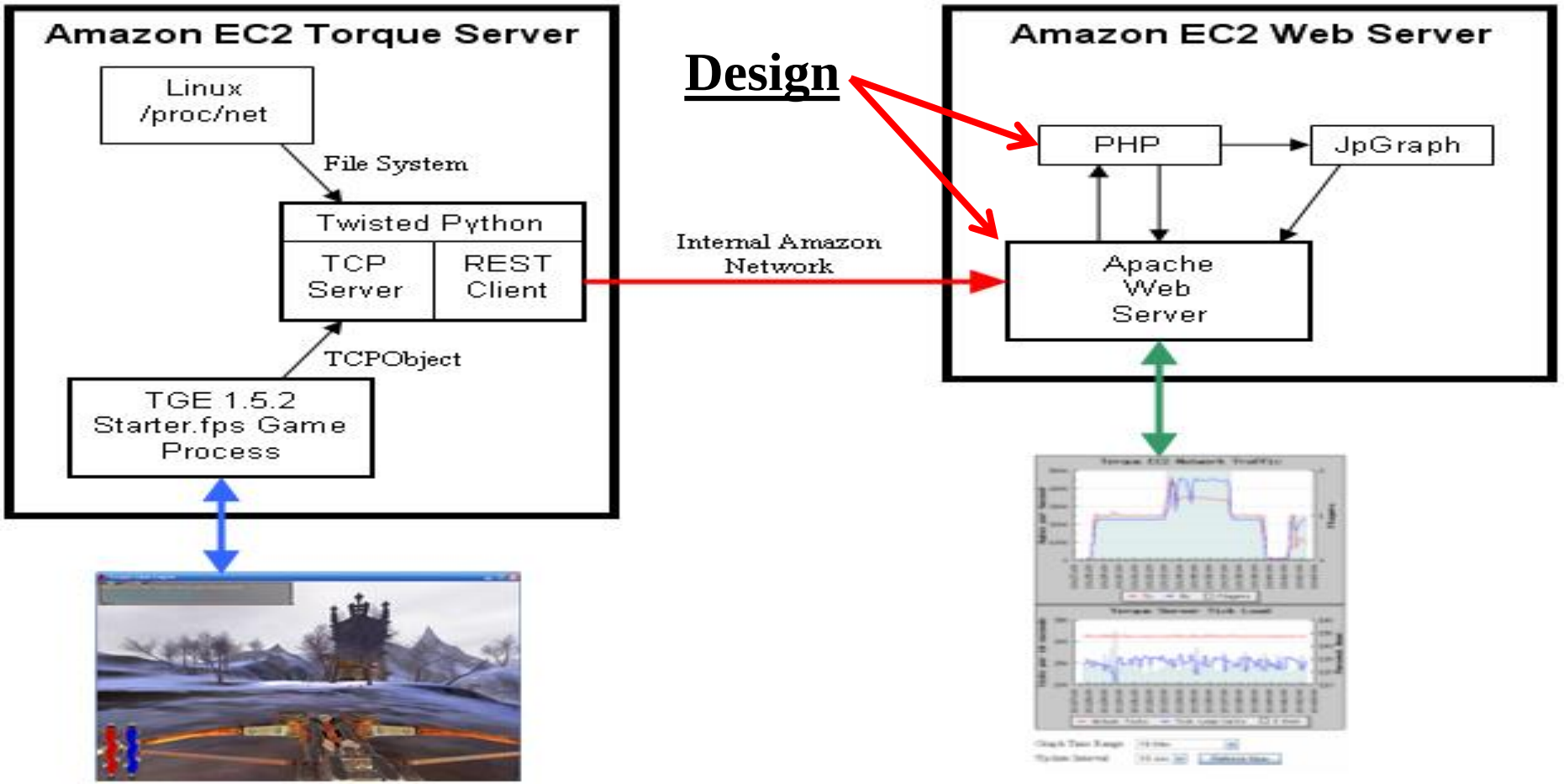
HttpCore Tutorial

Oleg Kalnichevski

4.4.5

Web Servers and Websites

- The most common use of web servers is to host websites, but there are other uses such as gaming, data storage or running enterprise applications.



AWS Reference Architectures

- Amazon EC2
- Auto Scaling
- Elastic Load Balancing
- Amazon DynamoDB
- Amazon S3
- Amazon SES
- Amazon EMR
- Amazon Route 53
- Amazon CloudFront

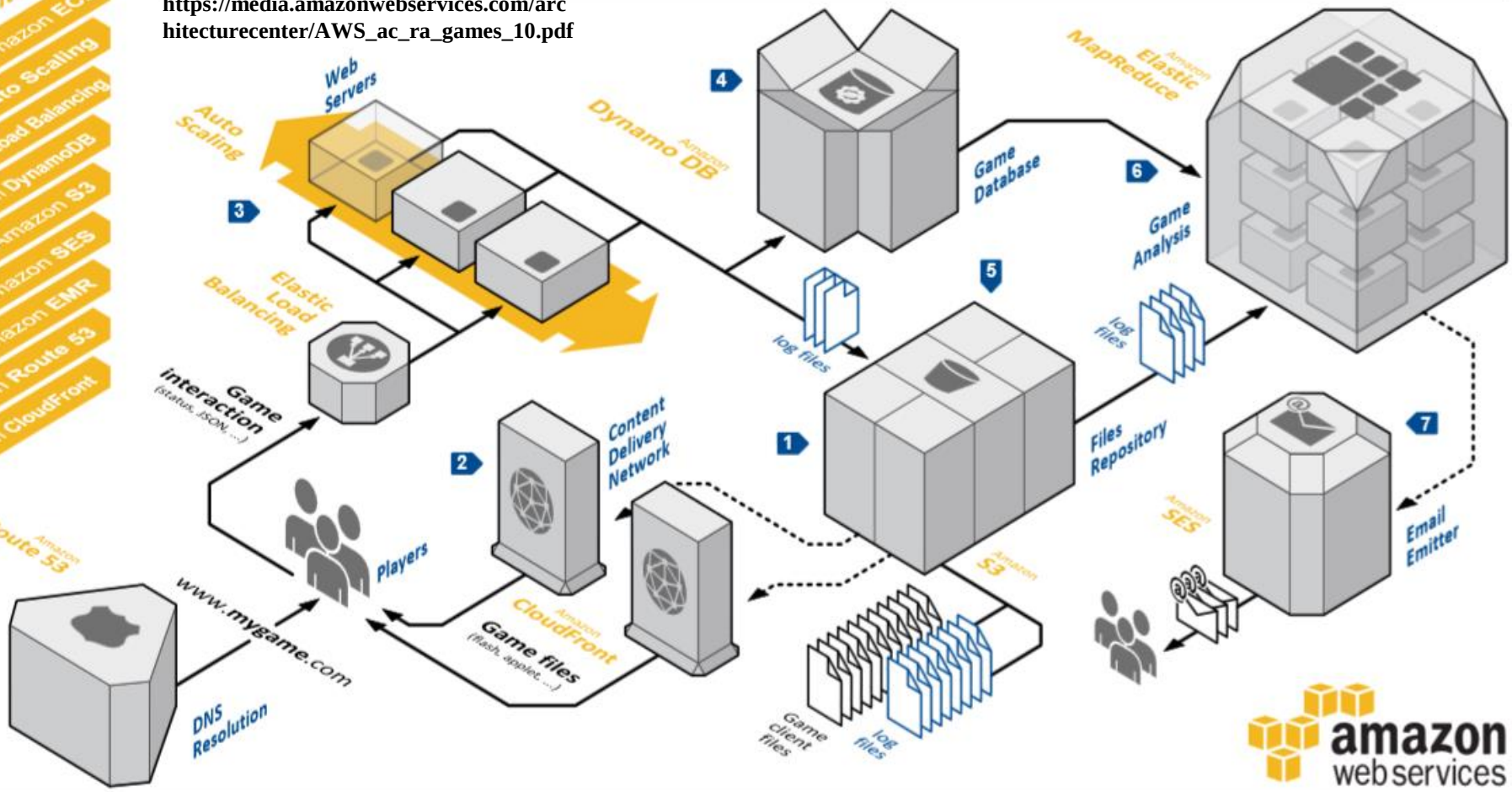
ONLINE GAMES

Online games back-end infrastructures can be challenging to maintain and operate. Peak usage periods, multiple players, and high volumes of write operations are some of the most common problems that operations teams face.

But the most difficult challenge is ensuring flexibility in the scale of that system. A popular game might suddenly receive millions of users in a matter of hours, yet it must continue to provide a satisfactory player experience. Amazon Web Services provides different tools and services that can be used for building online games that scale under high usage traffic patterns.

This document presents a cost-effective online game architecture featuring automatic capacity adjustment, a highly available and high-speed database, and a data processing cluster for player behavior analysis.

https://media.amazonwebservices.com/architecturecenter/AWS_ac_ra_games_10.pdf



System Overview

1 Browser games can be represented as client-server applications. The client generally consists of static files, such as images, sounds, flash applications, or Java applets. Those files are hosted on **Amazon Simple Storage Service** (Amazon S3), a highly available and reliable data store.

2 As the user base grows and becomes more geographically distributed, a high-performance cache like **Amazon CloudFront** can provide substantial improvements in latency, fault tolerance, and cost. By using **Amazon S3** as the origin server for the **Amazon CloudFront** distribution, the game infrastructure benefits from fast network data transfer rates and a simple publishing/caching workflow.

3 Requests from the game application are distributed by **Elastic Load Balancing** to a group of web servers running on **Amazon Elastic Compute Cloud** (Amazon EC2) instances. **Auto Scaling** automatically adjusts the size of this group, depending on rules like network load, CPU usage, and so on.

4 Player data is persisted on **Amazon DynamoDB**, a fully managed NoSQL database service. As the player population grows, **Amazon DynamoDB** provides predictable performance with seamless scalability.

5 Log files generated by each web server are pushed back into **Amazon S3** for long-term storage.

6 Managing and analyzing high data volumes produced by online games platforms can be challenging. **Amazon Elastic MapReduce** (Amazon EMR) is a service that processes vast amounts of data easily. Input data can be retrieved from web server logs stored on **Amazon S3** or from player data stored in **Amazon DynamoDB** tables to run analytics on player behavior, usage patterns, etc. Those results can be stored again on **Amazon S3**, or inserted in a relational database for further analysis with classic business intelligence tools.

7 Based on the needs of the game, **Amazon Simple Email Service** (Amazon SES) can be used to send email to players in a cost-effective and scalable way.

Term Project Proposal

- Due **Sunday, November 1**, submit via Blackboard.
- Your term project proposal should include:
  - **Names** of the team members:
  - Which **Option** (1, 2 or 3), using which language
    - For options 1 & 2, specify the 2 optional features you plan to implement, and provide as many details as you can.
    - For Option 3, provide as many details as you can about the **web site** you plan to build.
  - Last but not least, who is responsible for what. "Everybody is responsible for everything" is not good enough.
- Although it will not be graded, you will be asked to resubmit if it doesn't provide enough details.