CO324: Network and Web Application Design

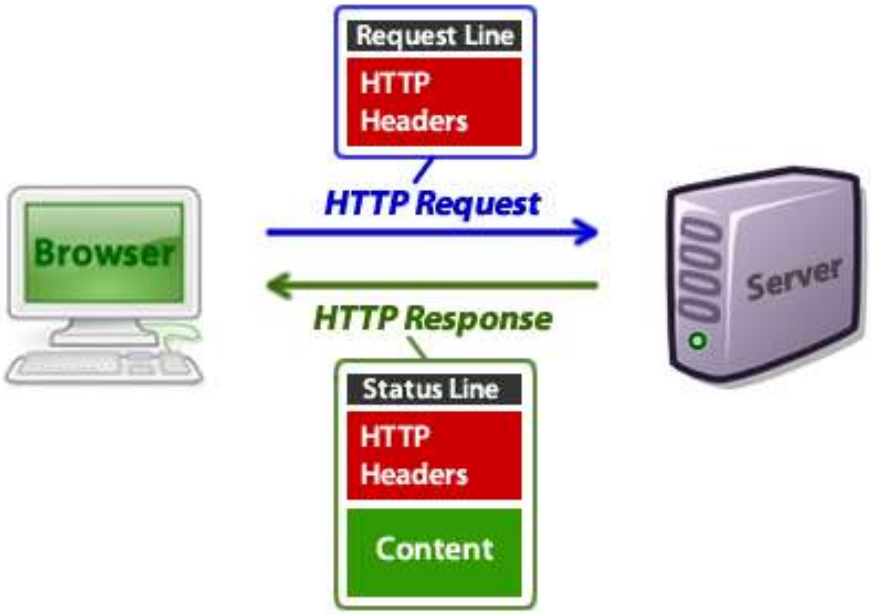
Network Programming

Deployment Consideration

- Database Connectivity
- Cache control headers
- Scaling techniques (reverse proxies, message queues)
- Best practices

HTTP Headers

- HTTP headers are core part of HTTP requests and responses.
- They carry information about client browser, requested page, server, etc.



Example of HTTP request

```
GET /tutorials/other/top-20-mysql-best-practices/ HTTP/1.1
Host: net.tutsplus.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5)
Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
```

HTTP Response

```
HTTP/1.x 200 OK
Transfer-Encoding: chunked
Date: Sat, 28 Nov 2009 04:36:25 GMT
Server: LiteSpeed
Connection: close
X-Powered-By: W3 Total Cache/0.8
Pragma: public
Expires: Sat, 28 Nov 2009 05:36:25 GMT
Etag: "pub1259380237;gz"
Cache-Control: max-age=3600, public
Content-Type: text/html; charset=UTF-8
Last-Modified: Sat, 28 Nov 2009 03:50:37 GMT
X-Pingback: http://net.tutsplus.com/xmlrpc.php
Content-Encoding: gzip
Vary: Accept-Encoding, Cookie, User-Agent

<!-- HTML page -->
```

Cache Control Headers

- HTTP header fields are components of the header section of request and response messages in the HTTP which defines the operating parameters of an HTTP transaction.
- HTTP headers allow the client and the server to pass additional information with the request or the response.
- The Cache-Control general-header field is used to specify directives for caching mechanisms in both requests and responses.

Cache request directives

- Standard Cache-Control directives that can be used by the client in an HTTP request.

  - Cache-Control: max-age=<seconds>
  - Cache-Control: max-stale[=<seconds>]
  - Cache-Control: min-fresh=<seconds>
  - Cache-Control: no-cache
  - Cache-Control: no-store
  - Cache-Control: no-transform
  - Cache-Control: only-if-cached

Cache response directives

- Standard Cache-Control directives that can be used by the server in an HTTP response.

  - Cache-Control: must-revalidate
  - Cache-Control: no-cache
  - Cache-Control: no-store
  - Cache-Control: no-transform
  - Cache-Control: public
  - Cache-Control: private
  - Cache-Control: proxy-revalidate
  - Cache-Control: max-age=<seconds>
  - Cache-Control: s-maxage=<seconds>

Directives

- public
  - Indicates that the response may be cached by any cache.
- private
  - Indicates that the response is intended for a single user and must not be stored by a shared cache. A private cache may store the response.
- no-cache
  - Forces caches to submit the request to the origin server for validation before releasing a cached copy.

Directives (cont.)

- max-age=<seconds>
  - Specifies the maximum amount of time a resource will be considered fresh. Contrary to Expires, this directive is relative to the time of the request.
- s-maxage=<seconds>
  - Overrides max-age or the Expires header, but it only applies to shared caches (e.g., proxies) and is ignored by a private cache.
- max-stale[=<seconds>]
  - Indicates that the client is willing to accept a response that has exceeded its expiration time. Optionally, you can assign a value in seconds, indicating the time the response must not be expired by.
- min-fresh=<seconds>
  - Indicates that the client wants a response that will still be fresh for at least the specified number of seconds.

Directives (cont.)

- must-revalidate
  - The cache must verify the status of the stale resources before using it and expired ones should not be used.
- proxy-revalidate
  - Same as must-revalidate, but it only applies to shared caches (e.g., proxies) and is ignored by a private cache.
- no-store
  - The cache should not store anything about the client request or server response.
- no-transform
  - No transformations or conversions should be made to the resource.

Examples

- Preventing caching
  - To turn off caching, you can send the following response header. In addition, see also the Expires and Pragma headers.

```
Cache-Control: no-cache, no-store, must-revalidate
```

- Caching static assets
  - For the files in the application that will not change, you can usually add aggressive caching by sending the response header below.

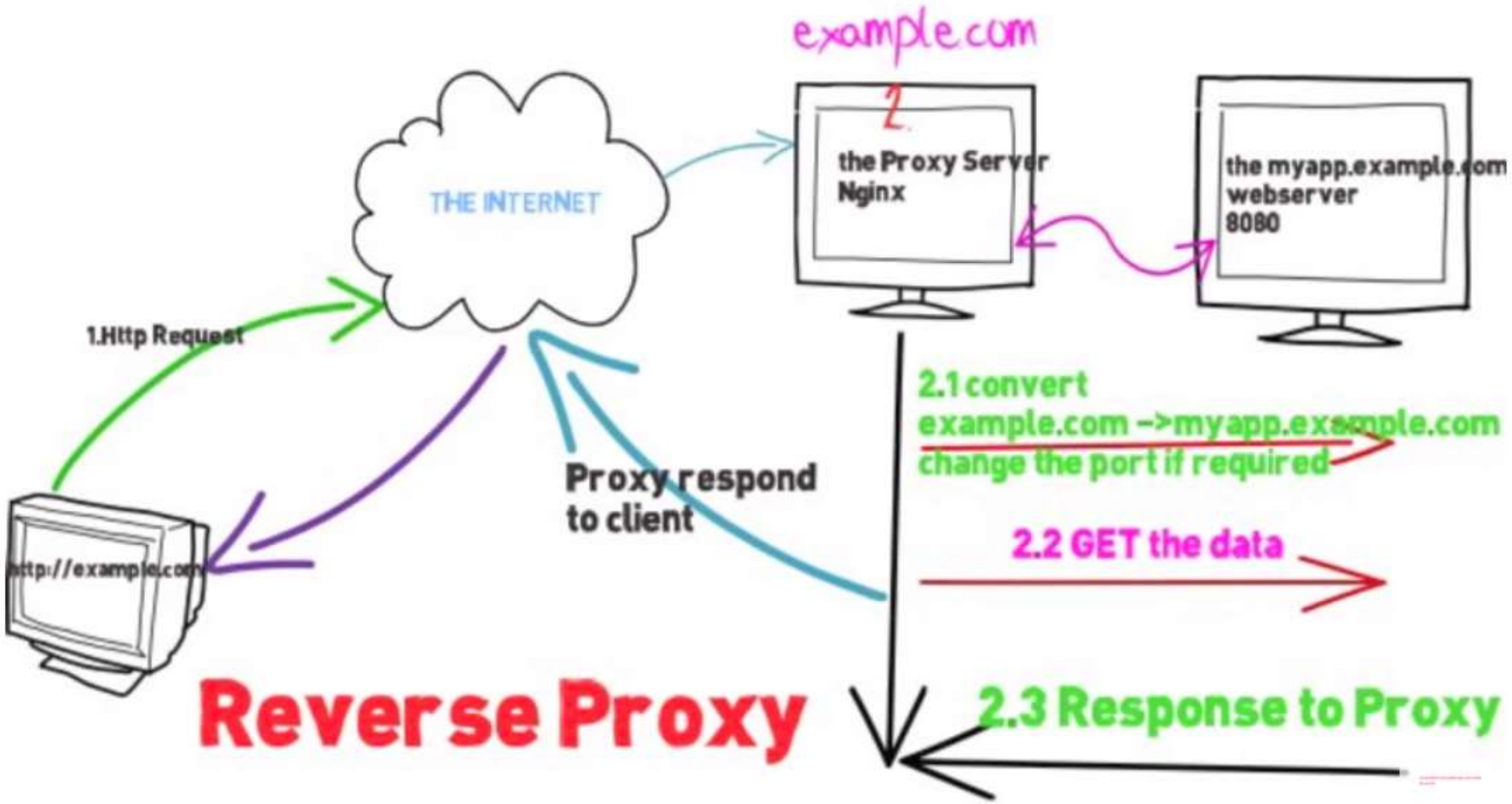
```
Cache-Control: public, max-age=31536000
```

Implementation

- Java Servlet/JSP
  - response.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
  - response.setHeader("Pragma", "no-cache");
  - response.setDateHeader("Expires", 0);
- PHP
  - header('Cache-Control: no-cache, no-store, must-revalidate');
  - header('Pragma: no-cache');
  - header('Expires: 0');

Reverse Proxy

- A reverse proxy is a type of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client as if they originated from the Web server itself. It is typically sits behind the firewall in a private network and directs client requests to the appropriate backend servers. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.
- Benefits
  - Increased security – No information about your backend servers is visible outside internal network, so malicious clients cannot access them directly to exploit any vulnerabilities. Many reverse proxy servers include features that help protect backend servers from distributed denial-of-service (DDoS) attacks.
  - Increased scalability and flexibility – Because clients see only the reverse proxy's IP address, you are free to change the configuration of your backend infrastructure. You can scale the number of servers up and down to match fluctuations in traffic volume.
  - Web acceleration – reducing the time it takes to generate a response and return it to the client. Techniques for web acceleration includes Compression and Caching.



Source: https://www.youtube.com/watch?v=Dgf9uBDX0-g

Building reliable, scalable, cheap, flexible, extendable Web Applications

Scalability

- Grows in small steps
- Stays up when it counts
- Can grow with your traffic
- Room for the future

Reliability

- High Quality of Service
- Minimal Downtime
- Stability
- Redundancy
- Resilience

Low Cost

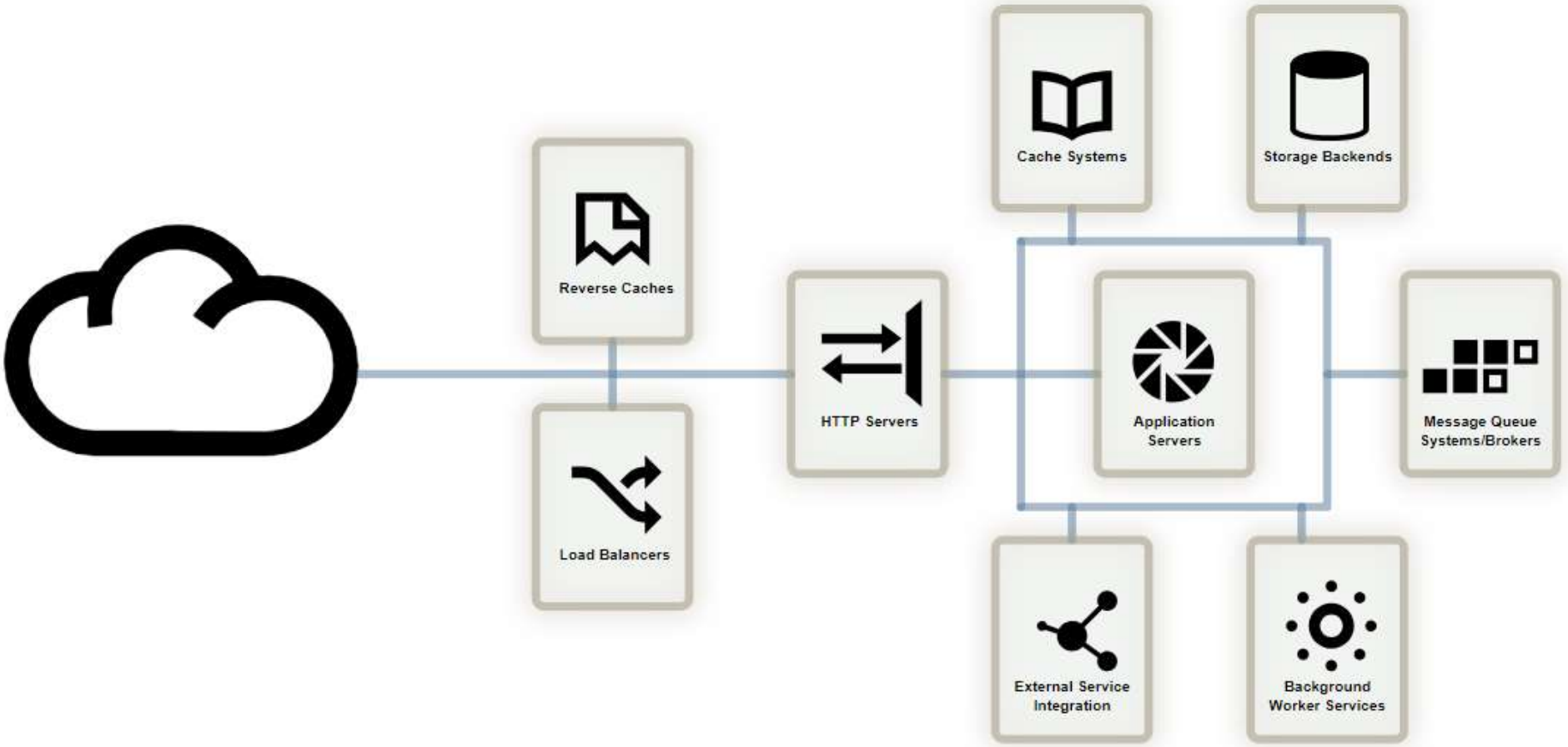
- Little or no software licensing costs
- Minimal hardware requirements
- Abundance of talent
- Reduced maintenance costs

Flexible

- Modular Components
- Public APIs
- Open Architecture
  - Vendor Neutral
  - Many options at all levels

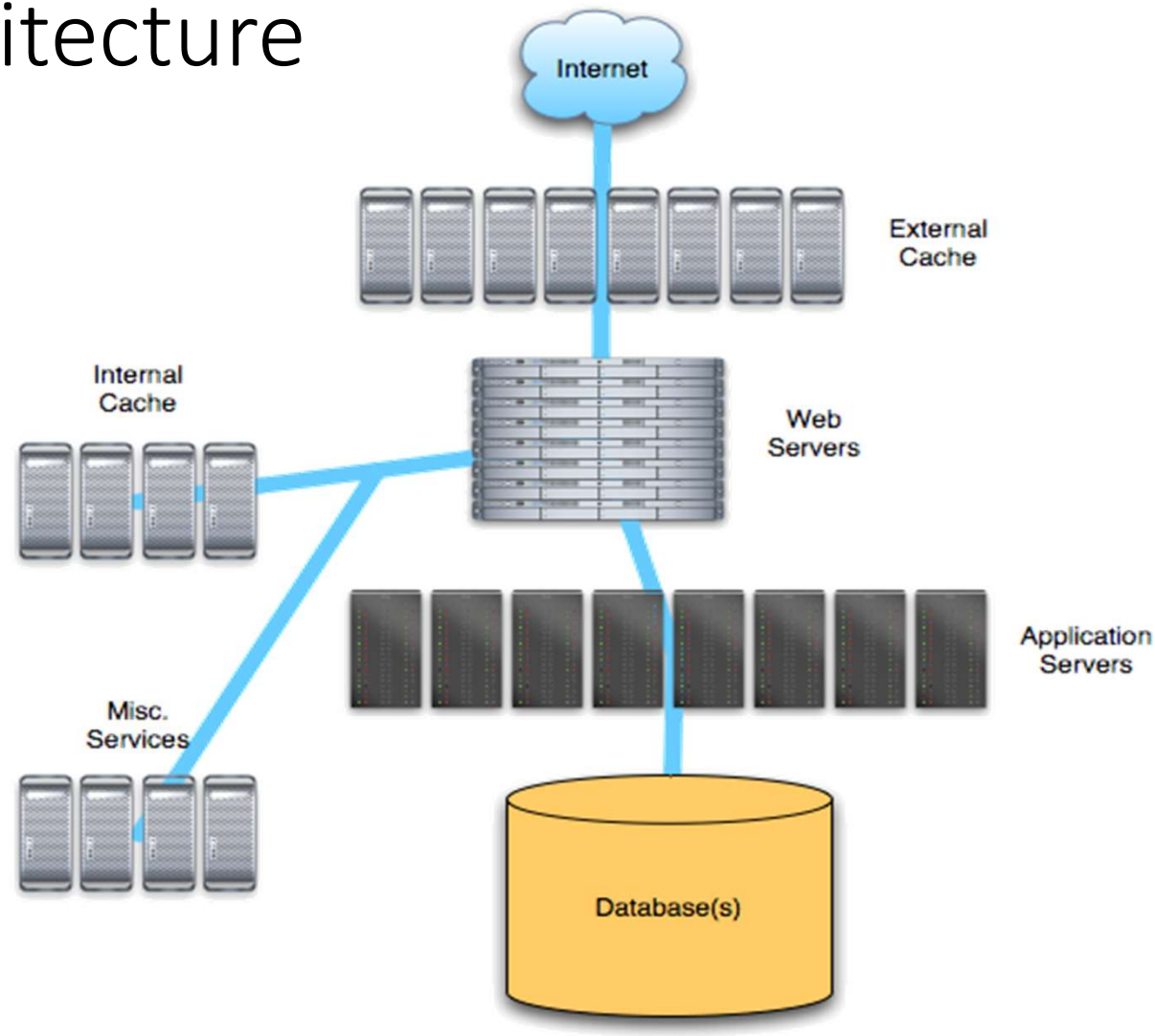
Extendable

- Free/Open Source Licensing
  - Right to Use
  - Right to Inspect
  - Right to Improve
- Plugins
  - Some Free
  - Some Commercial
  - Can always customize

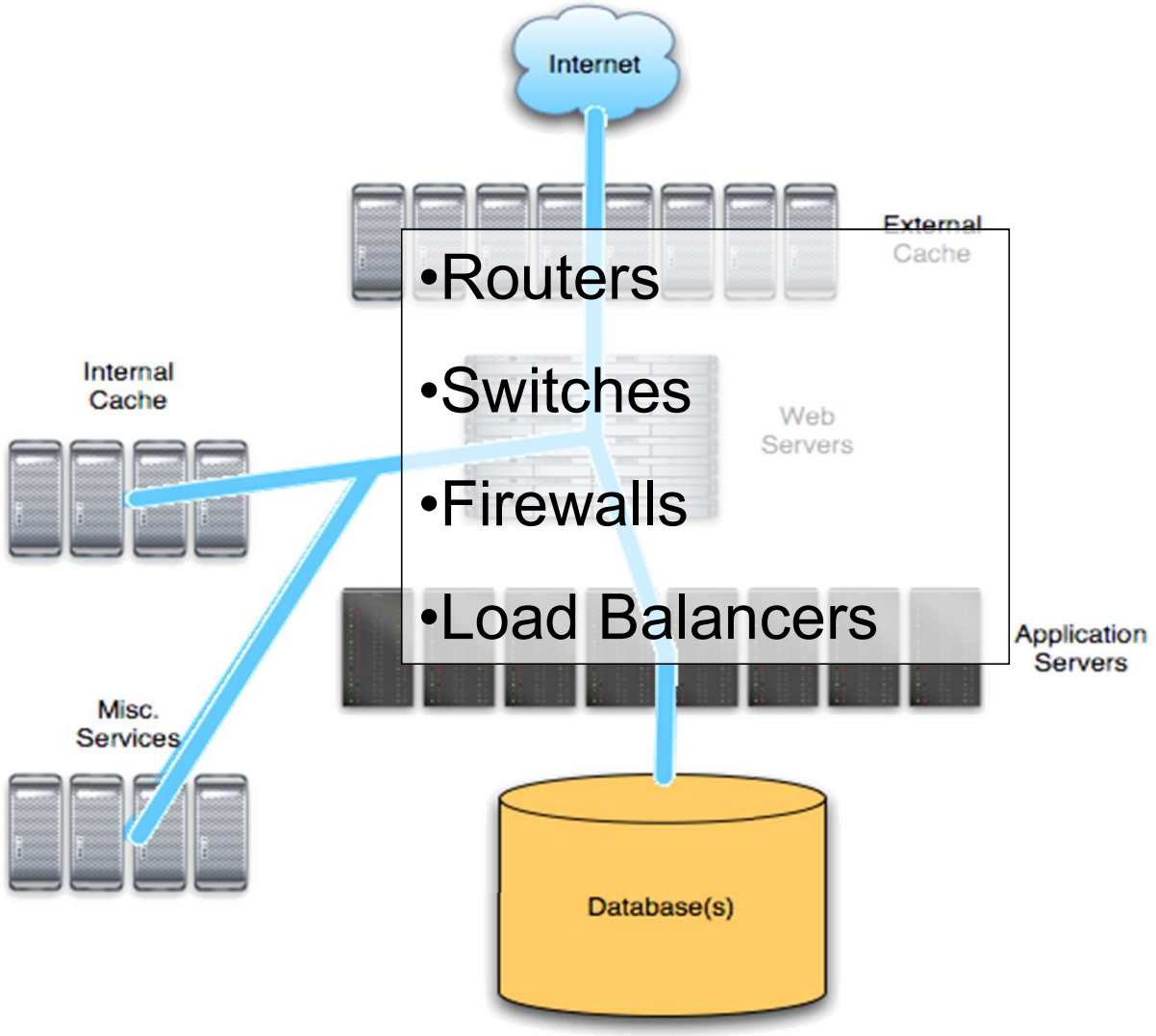


http://berb.github.io/diploma-thesis/community/033_archmodel.html

Typical Web Architecture

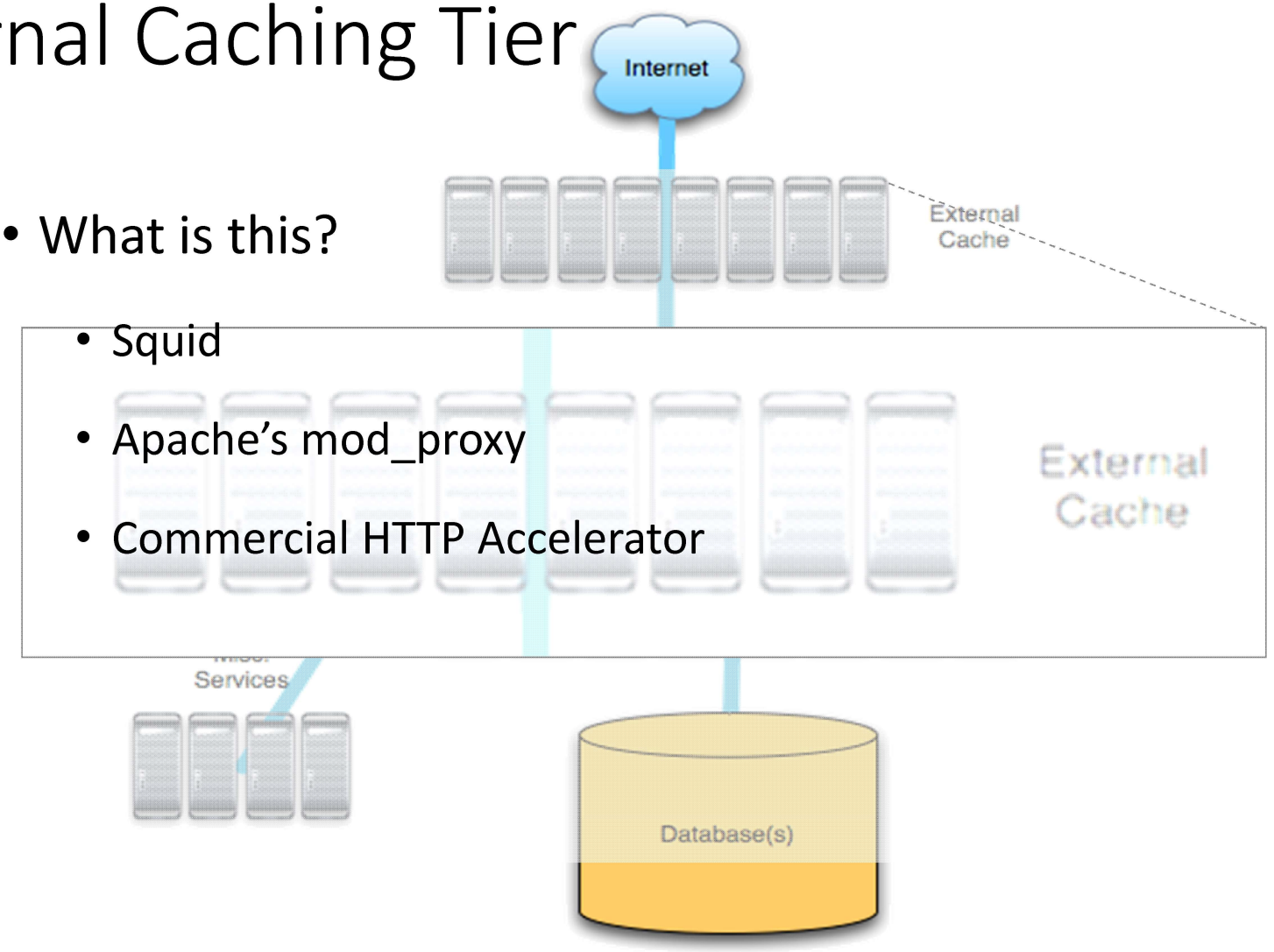


The Glue


Internet
External Cache
•Routers
•Switches
•Firewalls
•Load Balancers
Internal Cache
Web Servers
Application Servers
Misc. Services
Database(s)

External Caching Tier

- What is this?
  - Squid
  - Apache's mod_proxy
  - Commercial HTTP Accelerator

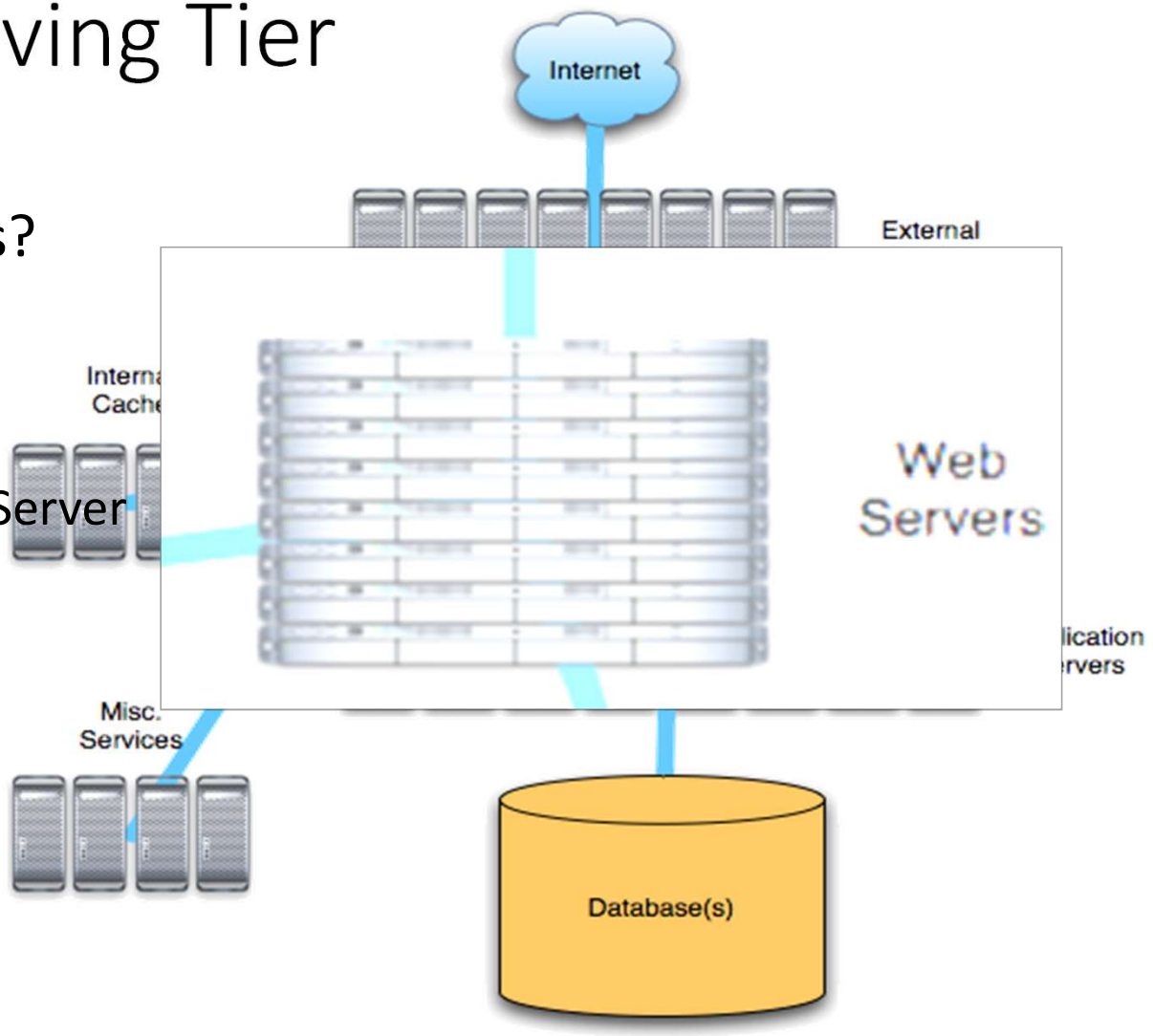


External Caching Tier

- What does it do?
  - Caches outbound HTTP objects
    - Images, CSS, XML, HTML, etc…
  - Flushes Connections
    - Useful for modem users, frees up web tier
  - Denial of Service Defense

Web Serving Tier

- What is this?
  - Apache
  - thttpd
  - Tux Web Server
  - IIS
  - Netscape

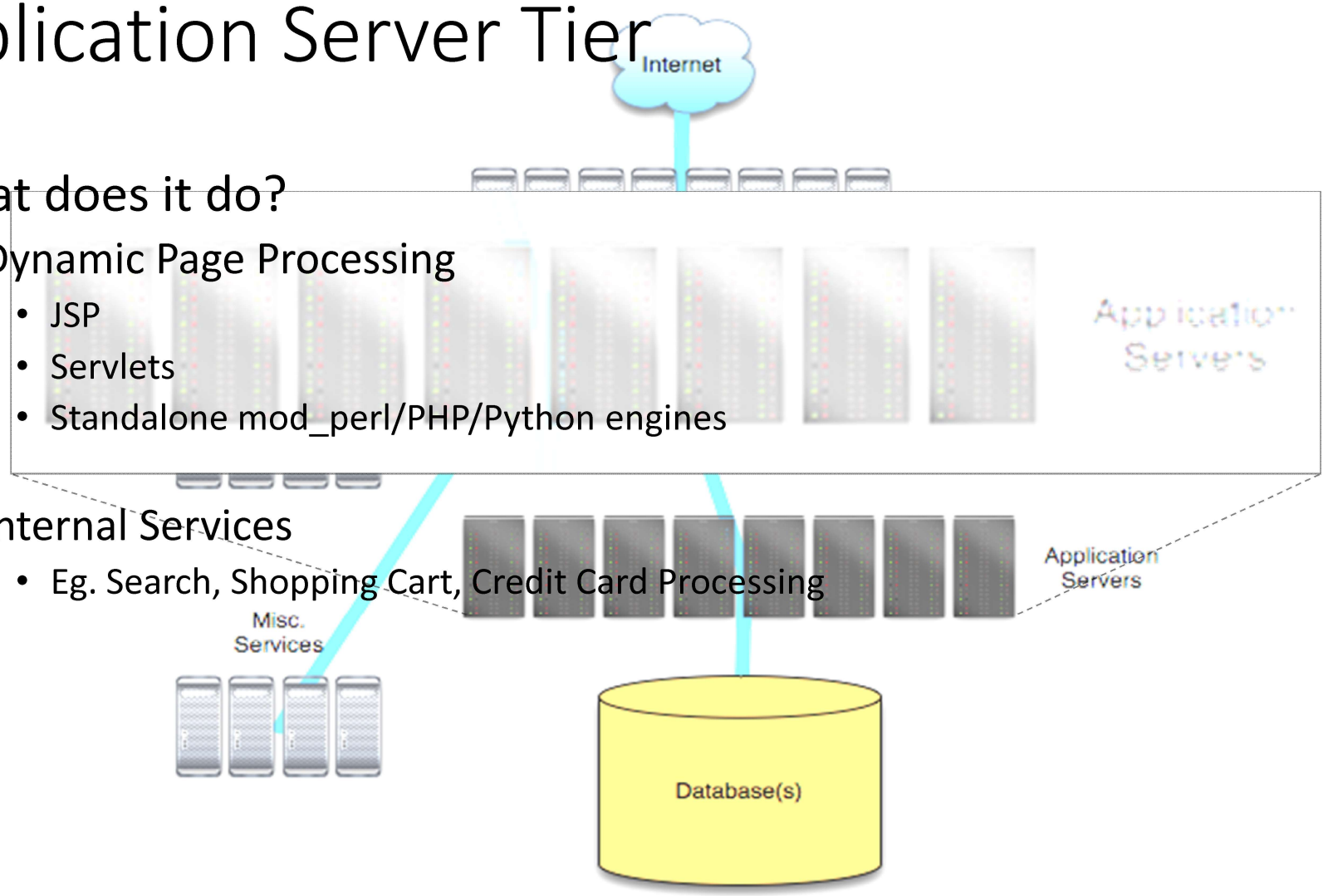


Web Serving Tier

- What does it do?
  - HTTP, HTTPS
  - Serves Static Content from disk
  - Generates Dynamic Content
    - CGI/PHP/Python/mod_perl/etc…
  - Dispatches requests to the App Server Tier
    - Tomcat, Weblogic, Websphere, JRun, etc…

Application Server Tier

- What does it do?
  - Dynamic Page Processing
    - JSP
    - Servlets
    - Standalone mod_perl/PHP/Python engines
  - Internal Services
    - Eg. Search, Shopping Cart, Credit Card Processing

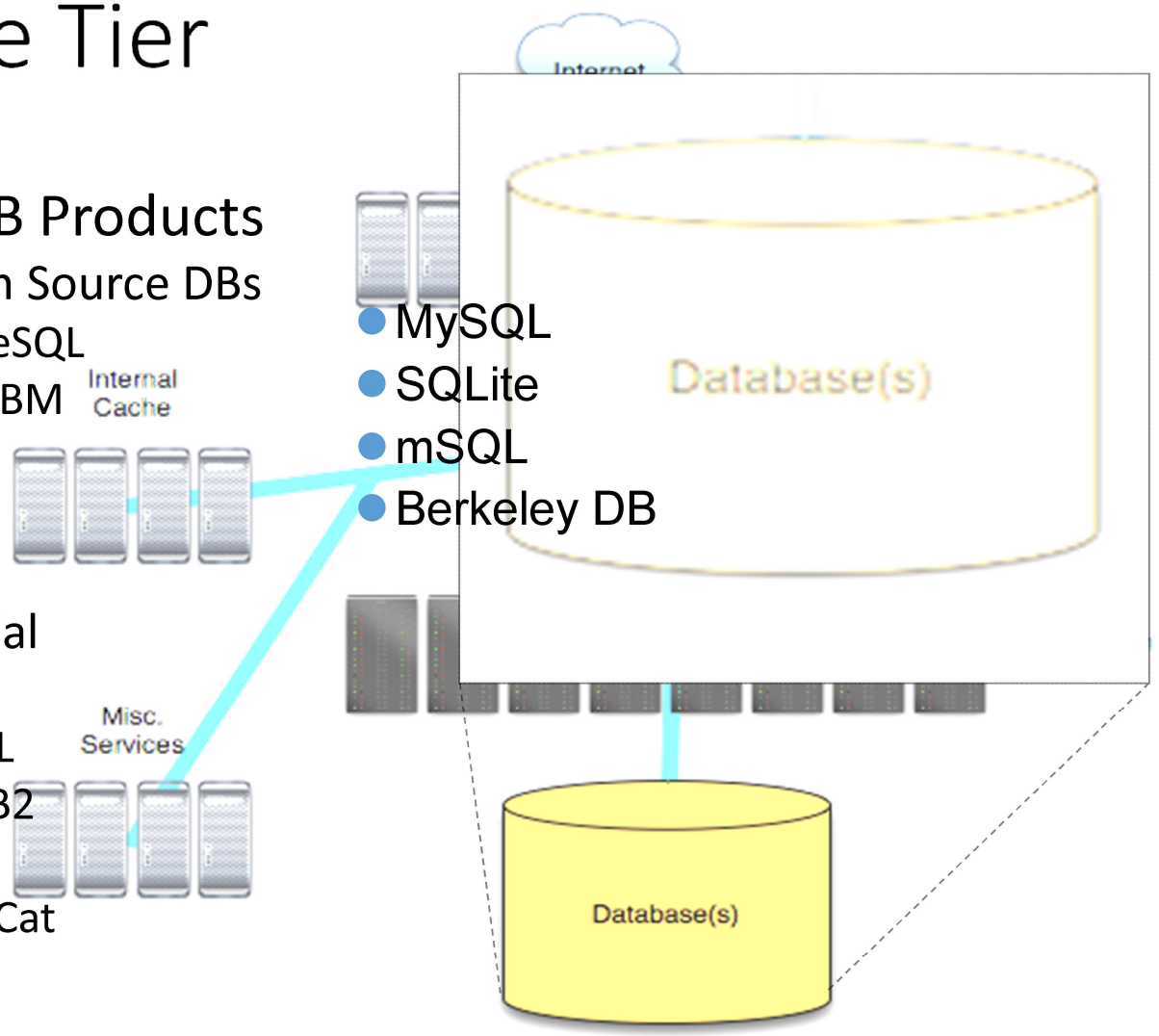


Application Server Tier

1. How does it work?
    1. Web Tier generates the request using
        - HTTP (aka "*REST*", sortof)
        - RPC/Corba
        - Java RMI
        - XMLRPC/Soap
        - (or something homebrewed)
    2. App Server processes request and responds

Database Tier

- Available DB Products
  - Free/Open Source DBs
    - PostgreSQL
    - GNU DBM
    - Ingres
    - SQLite
  - Commercial
    - Oracle
    - MS SQL
    - IBM DB2
    - Sybase
    - SleepyCat

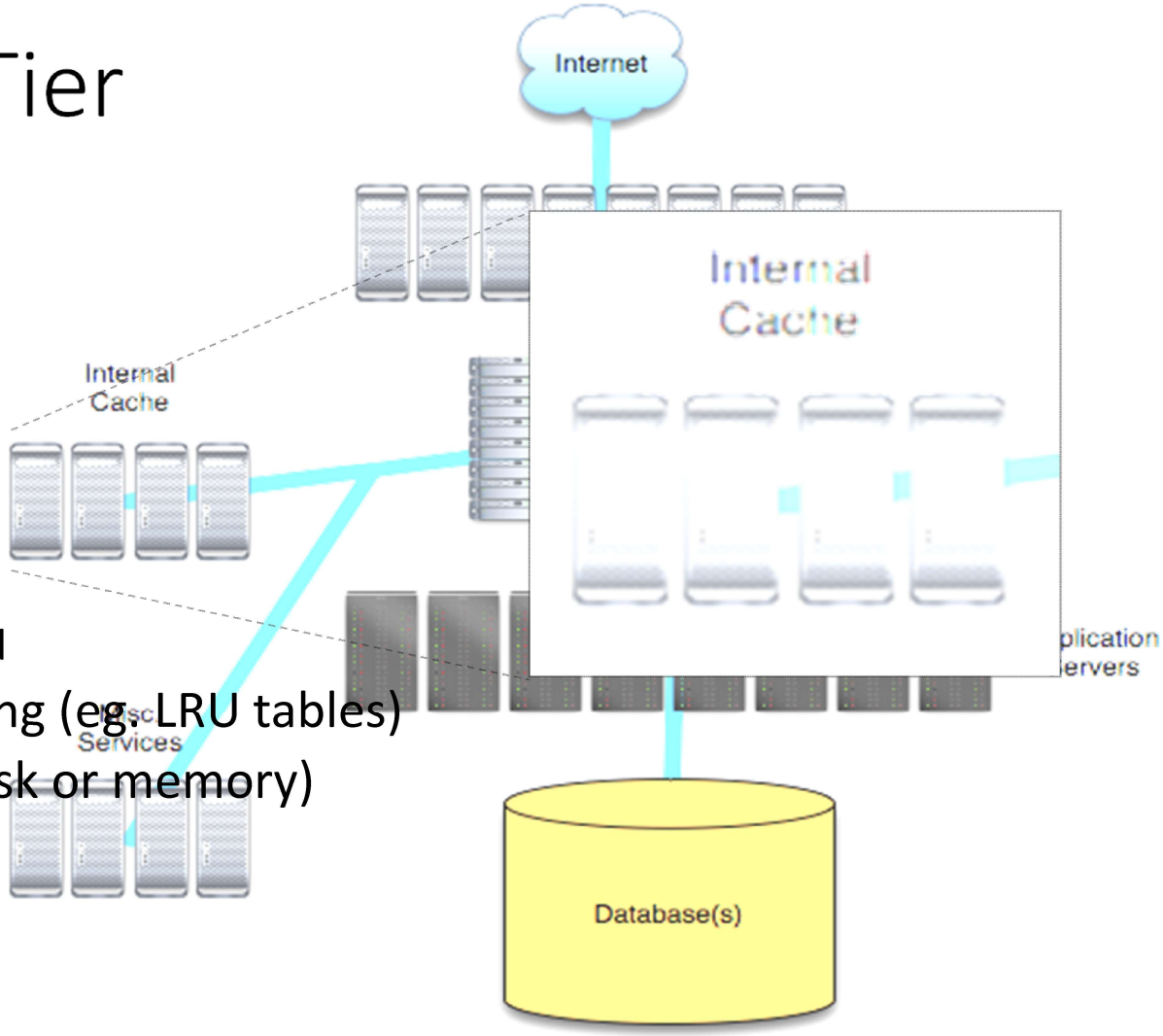


Database Tier

- What does it do?
  - Data Storage and Retrieval
  - Data Aggregation and Computation
  - Sorting
  - Filtering
  - ACID properties
    - (Atomic, Consistent, Isolated, Durable)

Internal Cache Tier

- What is this?
  - Object Cache
- What Applications?
  - Memcache
  - Local Lookup Tables
    - BDB, GDBM, SQL-based
  - Application-local Caching (eg. LRU tables)
  - Homebrew Caching (disk or memory)



Internal Cache Tier

- What does it do?
  - Caches objects closer to the Application or Web Tiers
  - Tuned for your application
  - Very Fast Access
  - Scales Horizontally

Misc. Services (DNS, Mail, etc…)