Distributed Systems Fundamentals [Fall 2013]

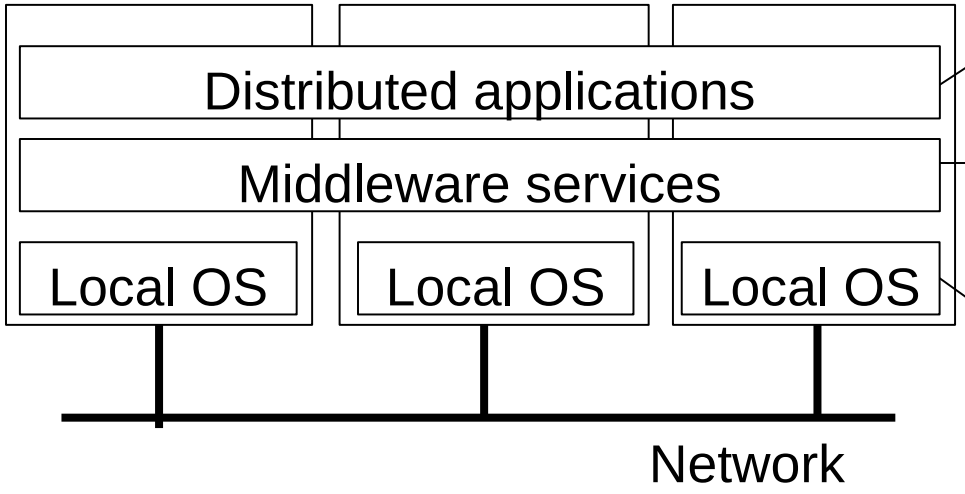
Lec 2: Example use case: The Web

Reminder/Quiz

- Define distributed systems
- Distributed systems goals
- Distributed systems challenges

Reminder/Quiz

- Define distributed systems



Gmail, search, Facebook, mobile apps, data analytics, …

Distributed FSes, distributed computation systems, locking services, CDNs, …

Linux, OSX, Windows

- Distributed systems goals
  - Raise the level of abstraction, provide location transparency, scalable capacity, availability, modularity
- Distributed systems challenges
  - Interfaces, scalability, consistency, fault-tolerance, security, implementation

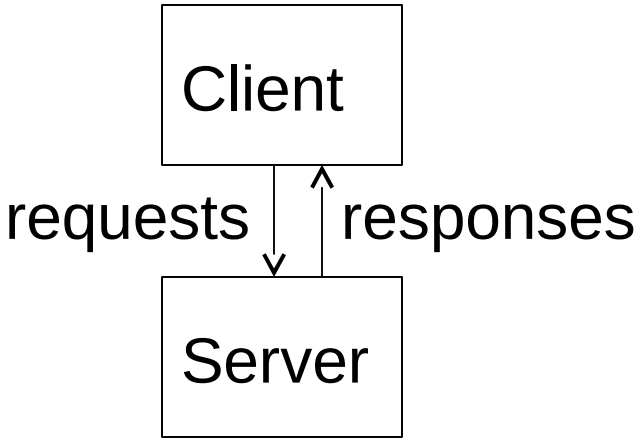
Today: Web Architectures

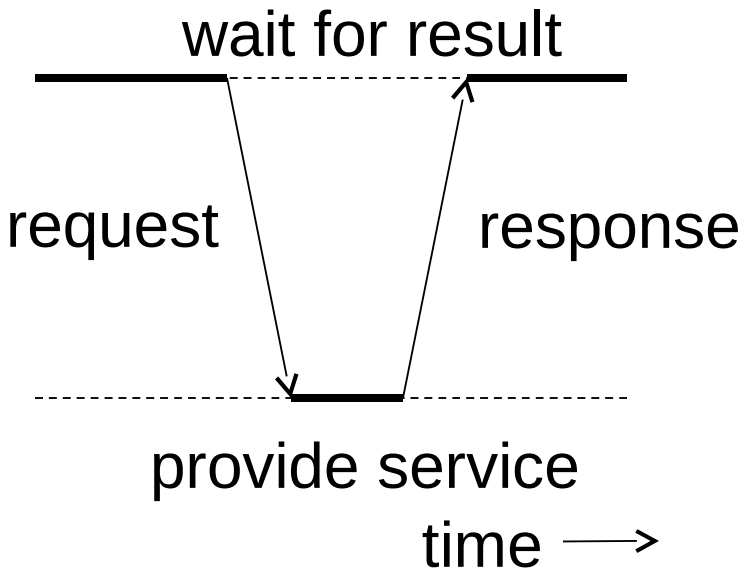
- Simple architectures
  - From Tanenbaum textbook
- Real-world architectures
  - Acknowledgements to Aaron Bannert, whose slides were used here (his slides no longer available online)

What Are Some Simple Architectures?

- Recall Tanenbaum reading for today

1. The Client-Server Model



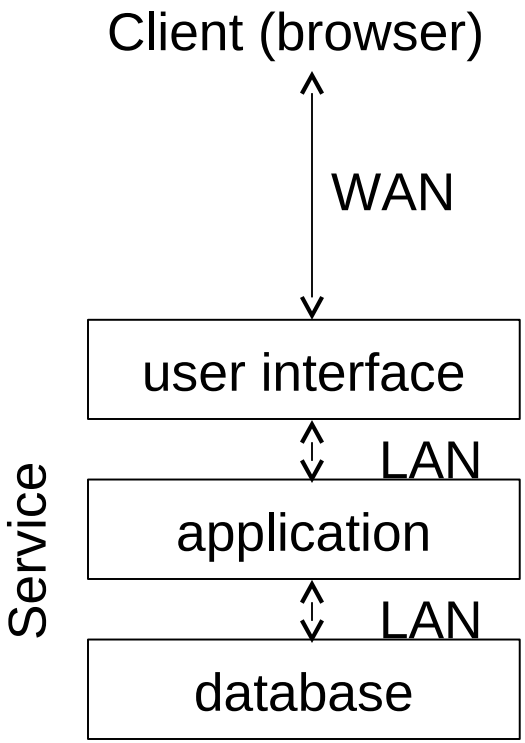


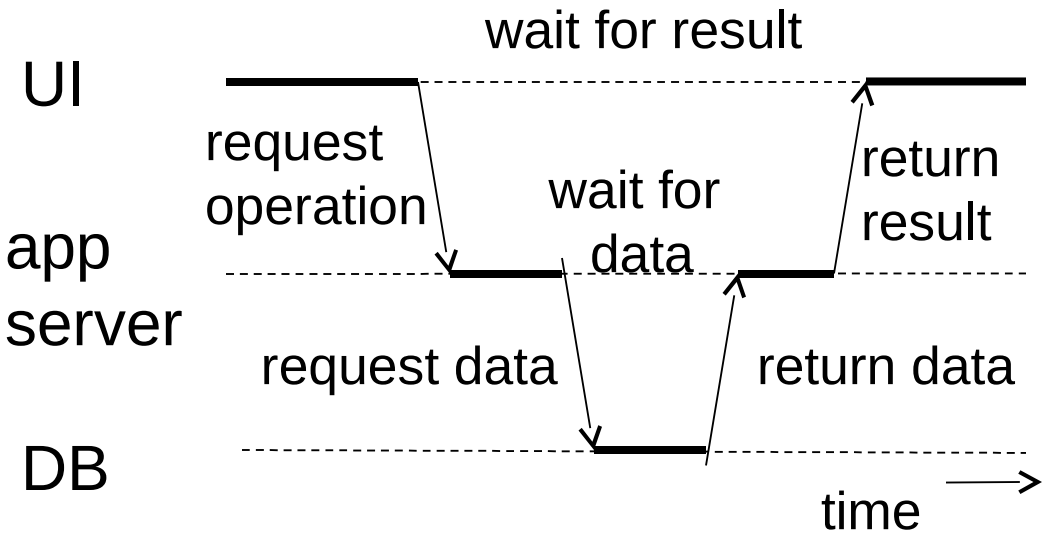
- Popular protocols between clients/servers:
  - HTTP, HTTPS
  - AJAX: asynchronous requests
  - XMLRPC, SOAP: web service API requests

Server-Side Processing

- Initially, Web servers returned static HTML pages
  - No processing on server, no state, no user-provided data
- 1994: CGI (Common Gateway Interface)
  - Server invokes a program upon each request
  - Program gets client data from stdin, outputs HTML to stdout
  - Example: Listing 1
- Then came a lot of server-side frameworks:
  - Django, ASP, JSP, Ruby-on-Rails, …
  - Much more flexible and extensible than CGI
  - Separate presentation, logic, and DB
  - Example: Listing 2

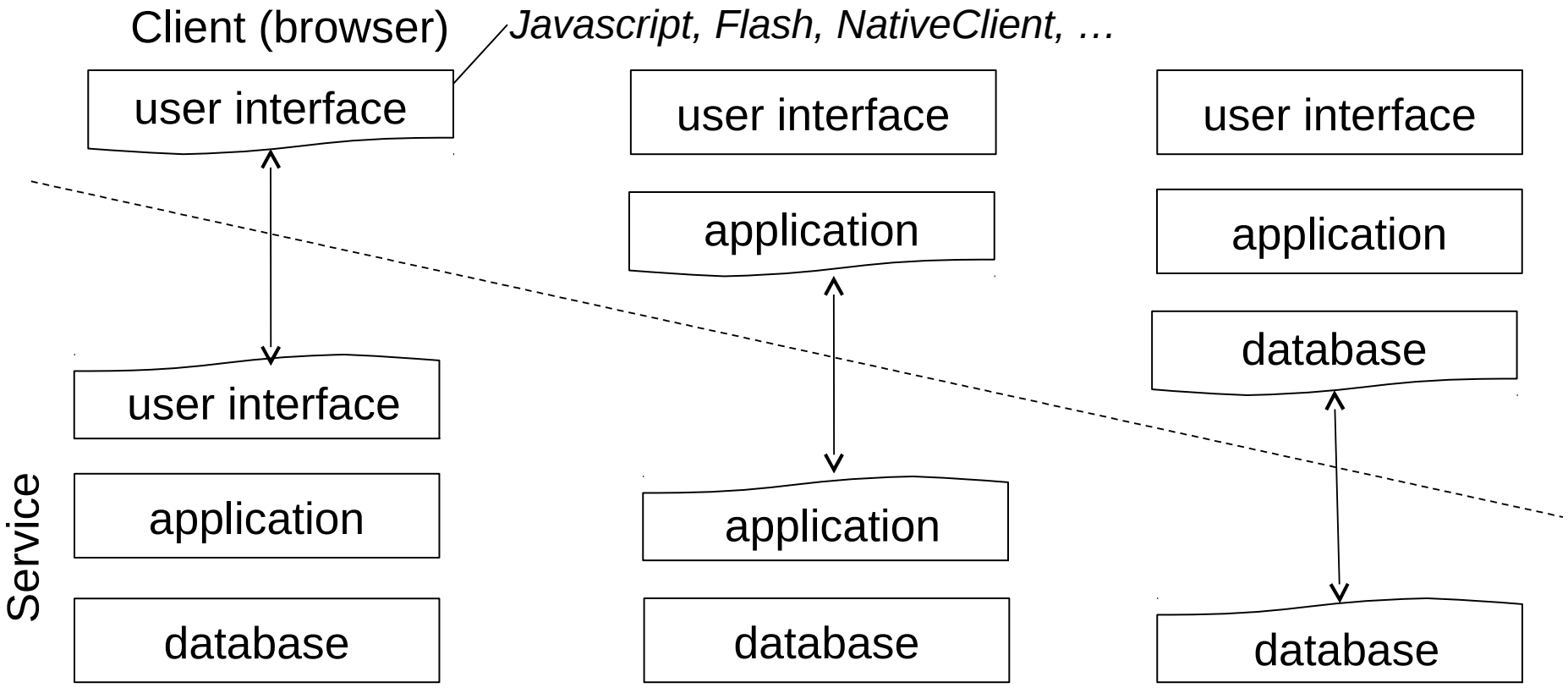
2. The Three-Tiered Architecture





- What are the benefits/problems with this architecture?
  - \+ Modularity, better reliability/scalability opportunities
  - \- Poor user latency

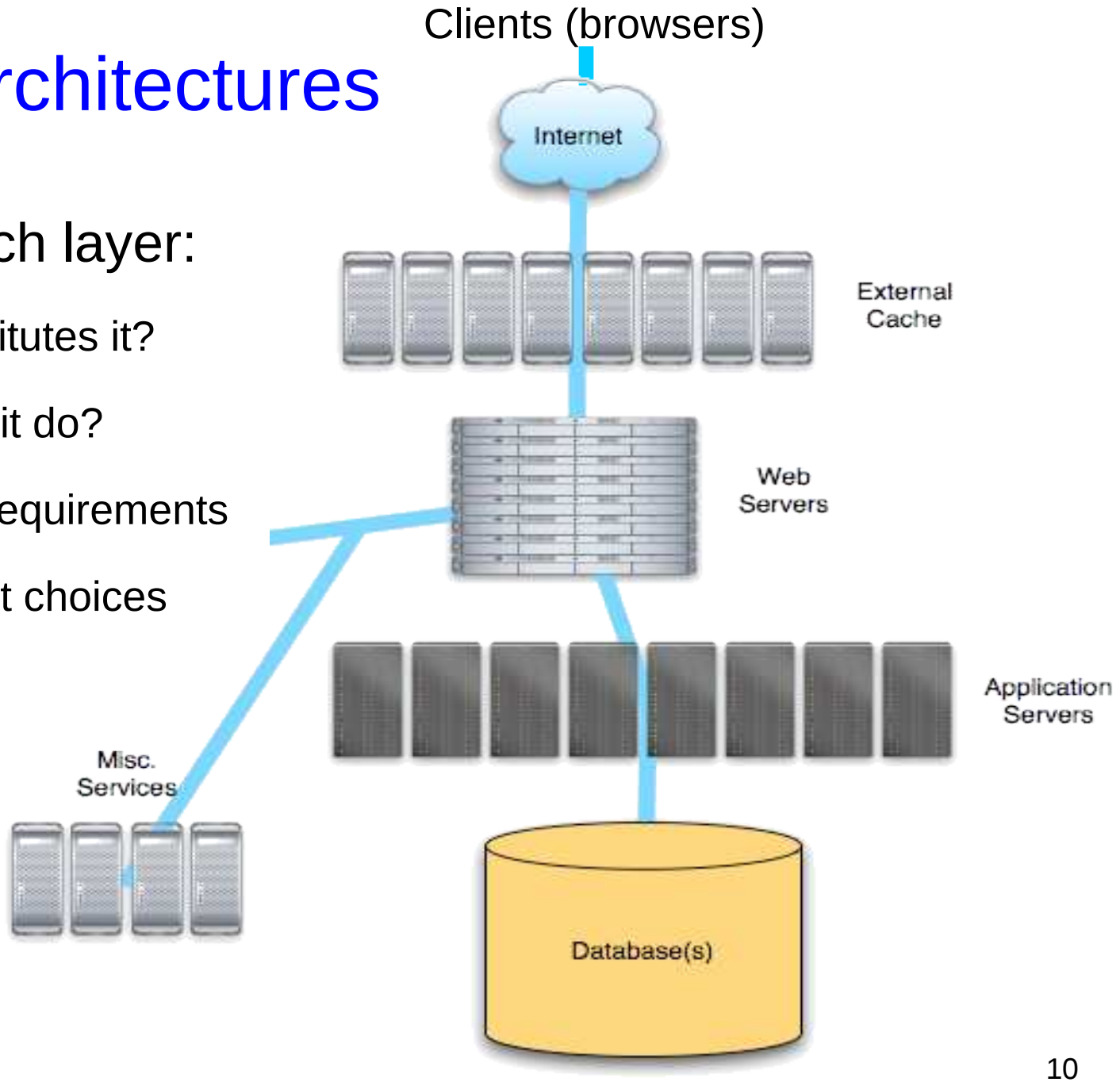
Client-side Computation



- In reality, the line is much fuzzier and the architecture is not as clean on service-side…

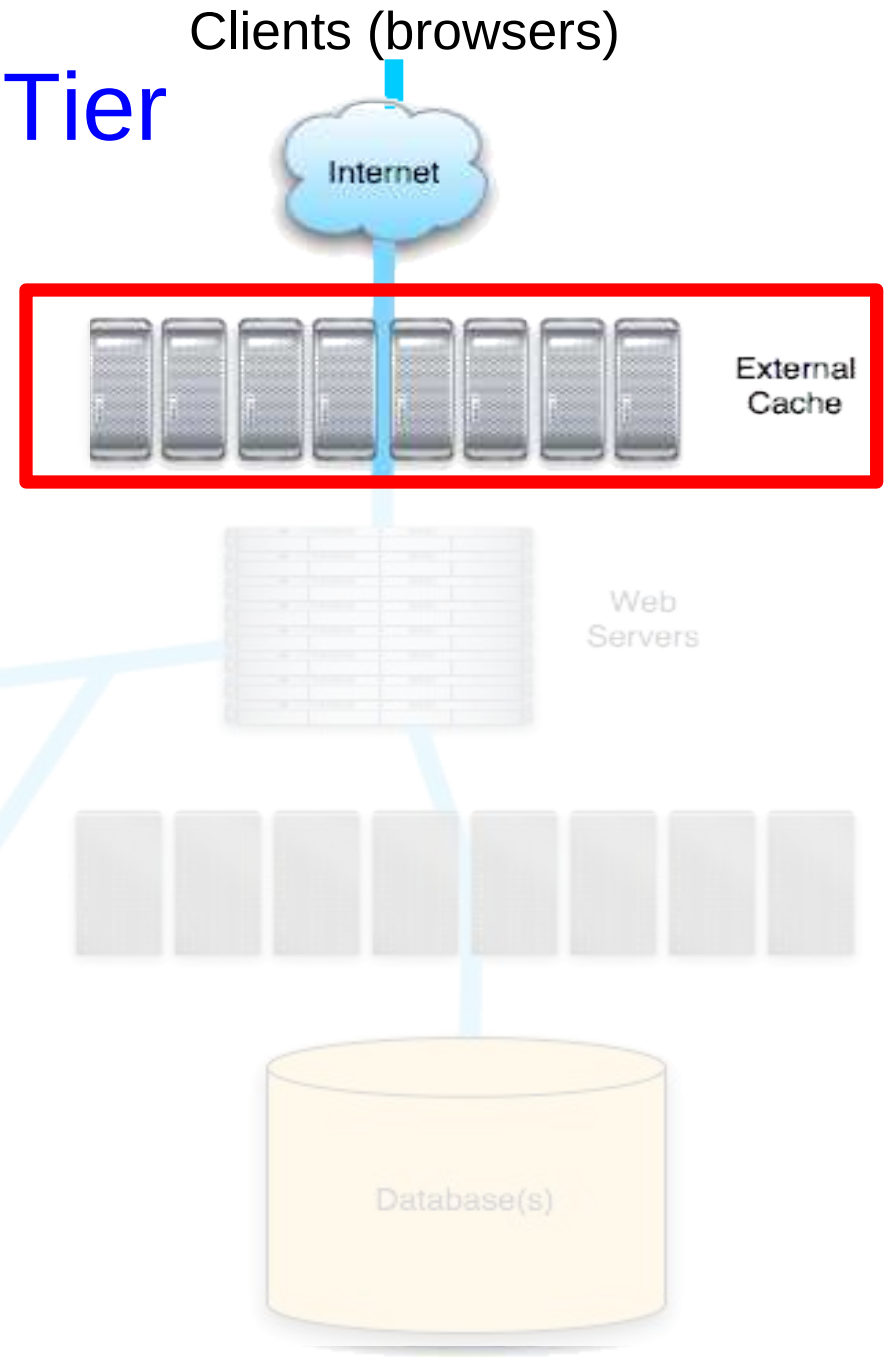
3. Real Architectures

- Discuss each layer:
  - What constitutes it?
  - What does it do?
  - Hardware requirements
  - Deployment choices



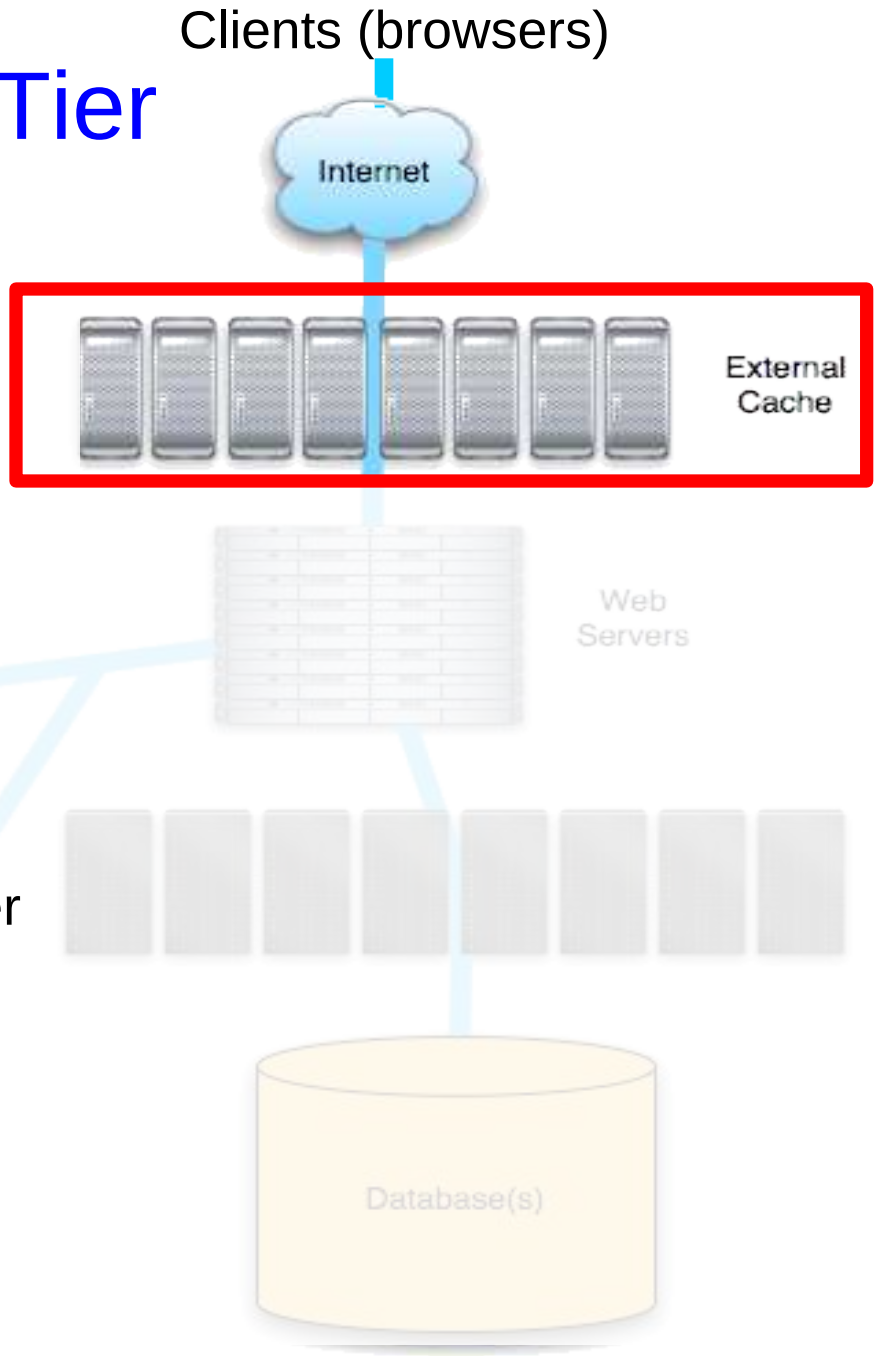
External Caching Tier

- What is this?
  - Squid, Apache mod_proxy
  - Content-delivery networks (CDNs), e.g., Akamai



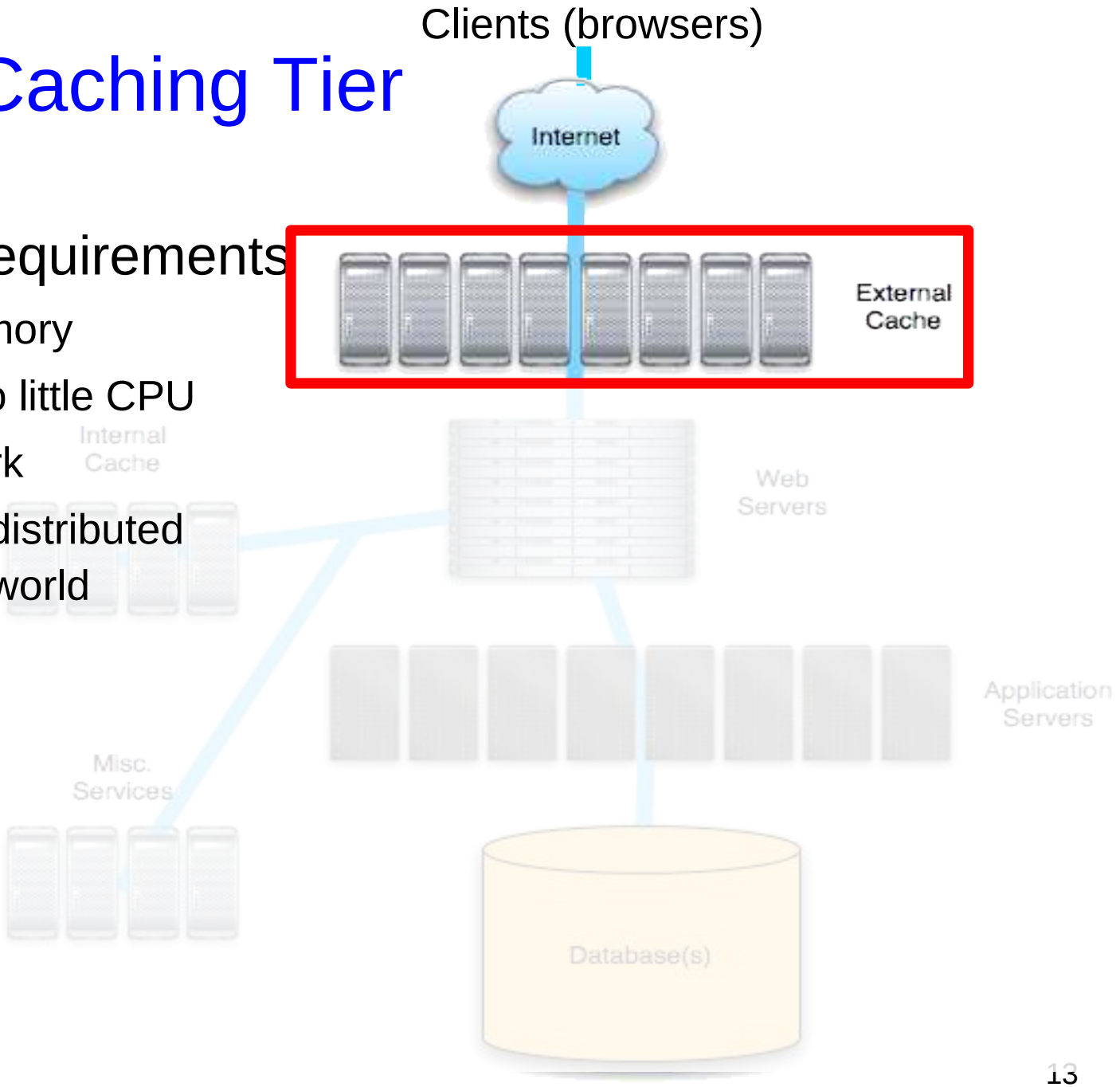
External Caching Tier

- What does it do?
  - Caches outbound data
    - Images, CSS, XML, HTML, pictures, videos, …
  - Denial of Service defense
  - Cache may be close to user



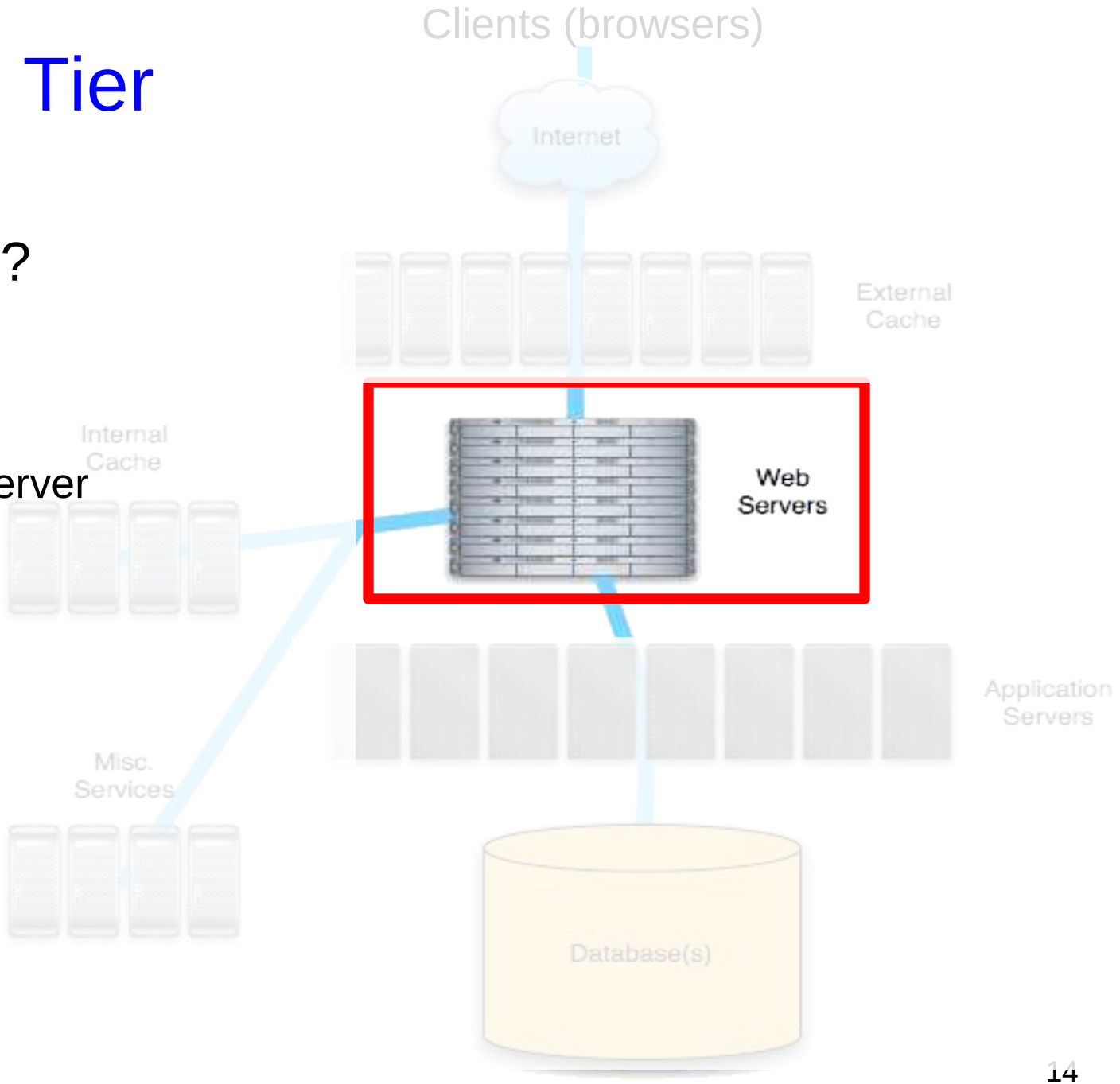
External Caching Tier

- Hardware requirements
  - Lots of memory
  - Moderate to little CPU
  - Fast network
  - Potentially distributed across the world



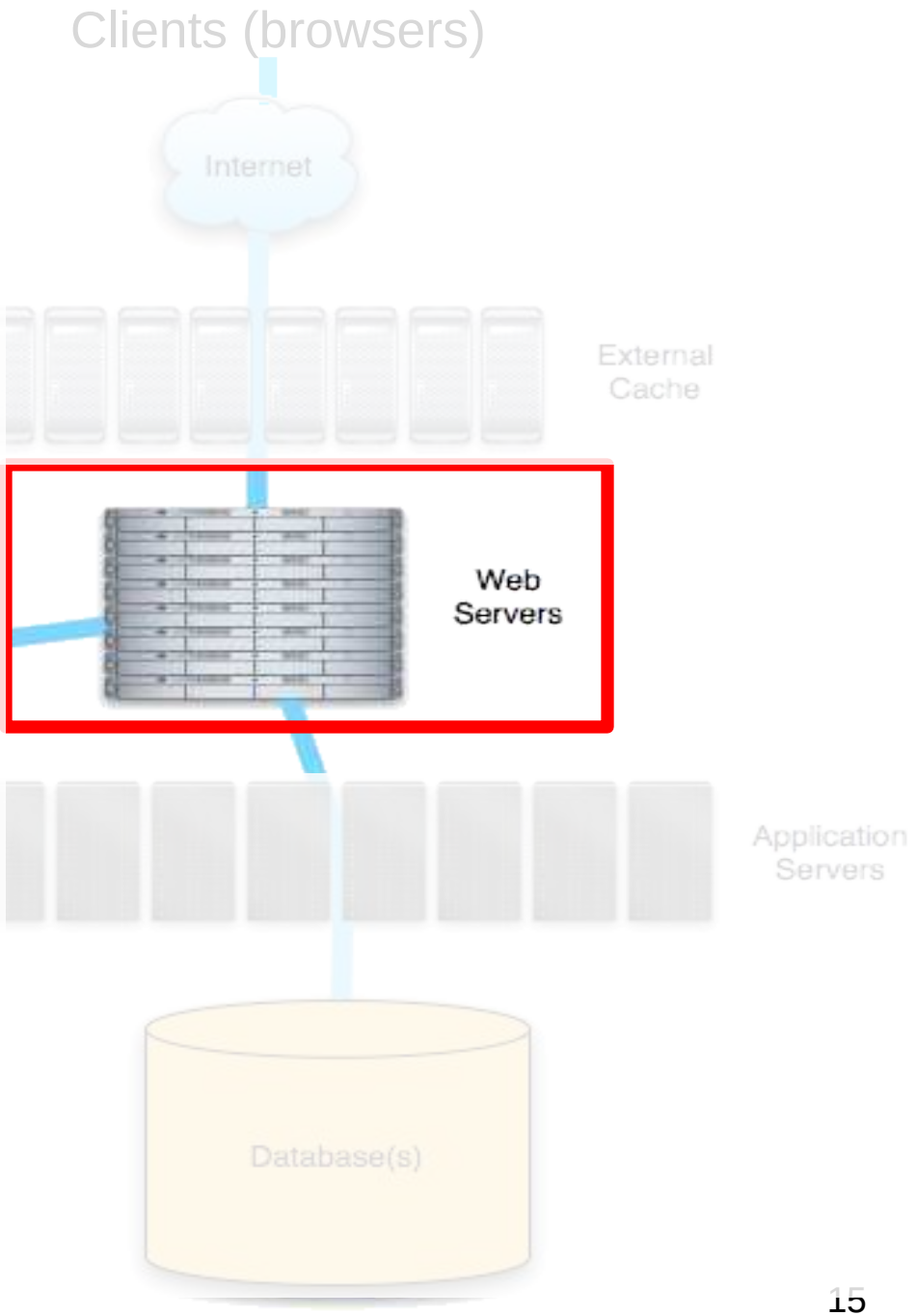
Front-end Tier

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



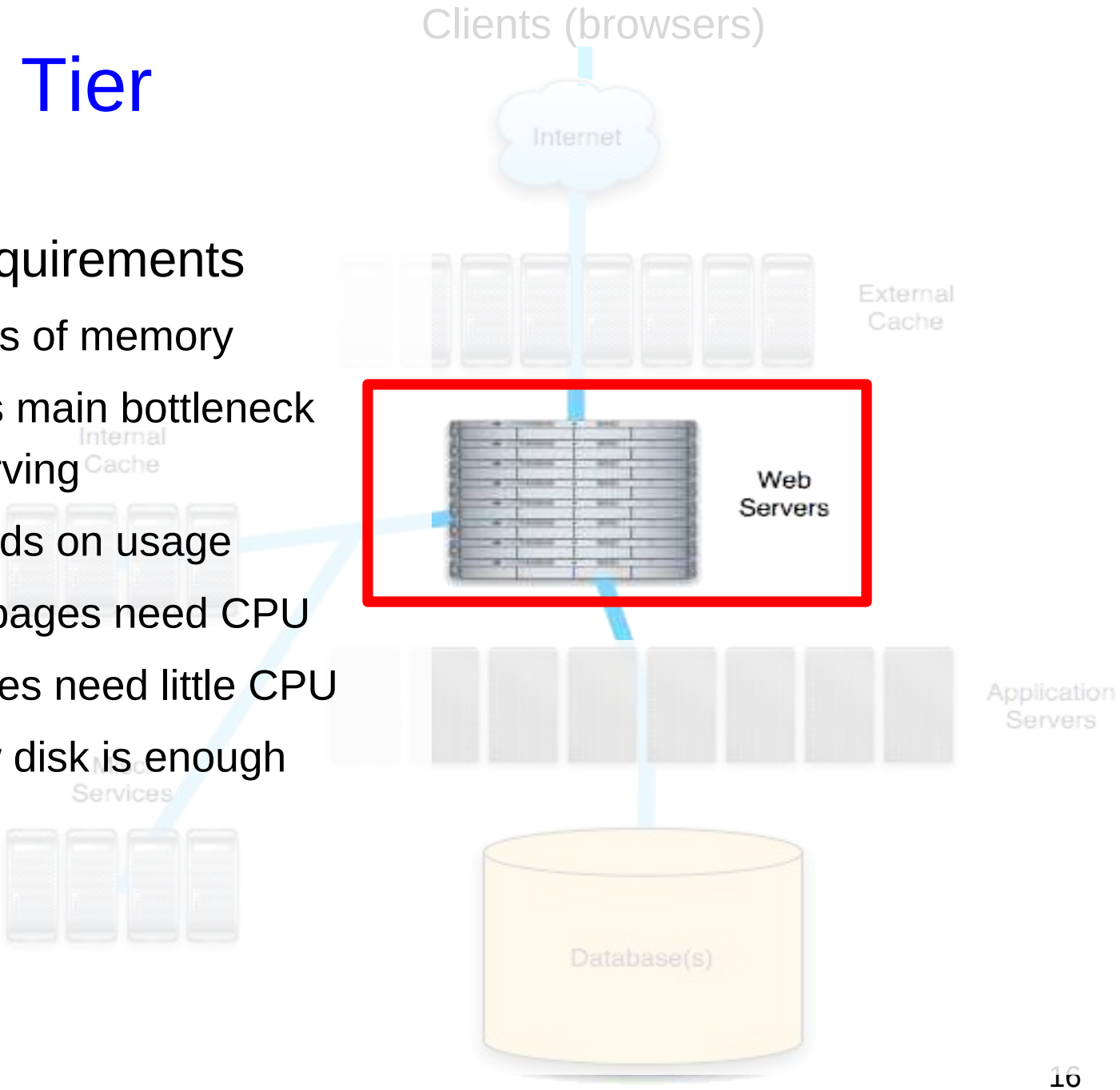
Front-end Tier

- What does it do?
  - HTTP, HTTPS
  - Serves static content from disk
  - Generates dynamic content
    - CGI/PHP/python/Django/..
  - Dispatches requests to the App Server Tier
    - Tomcat, Weblogic, Websphere, JRun, …



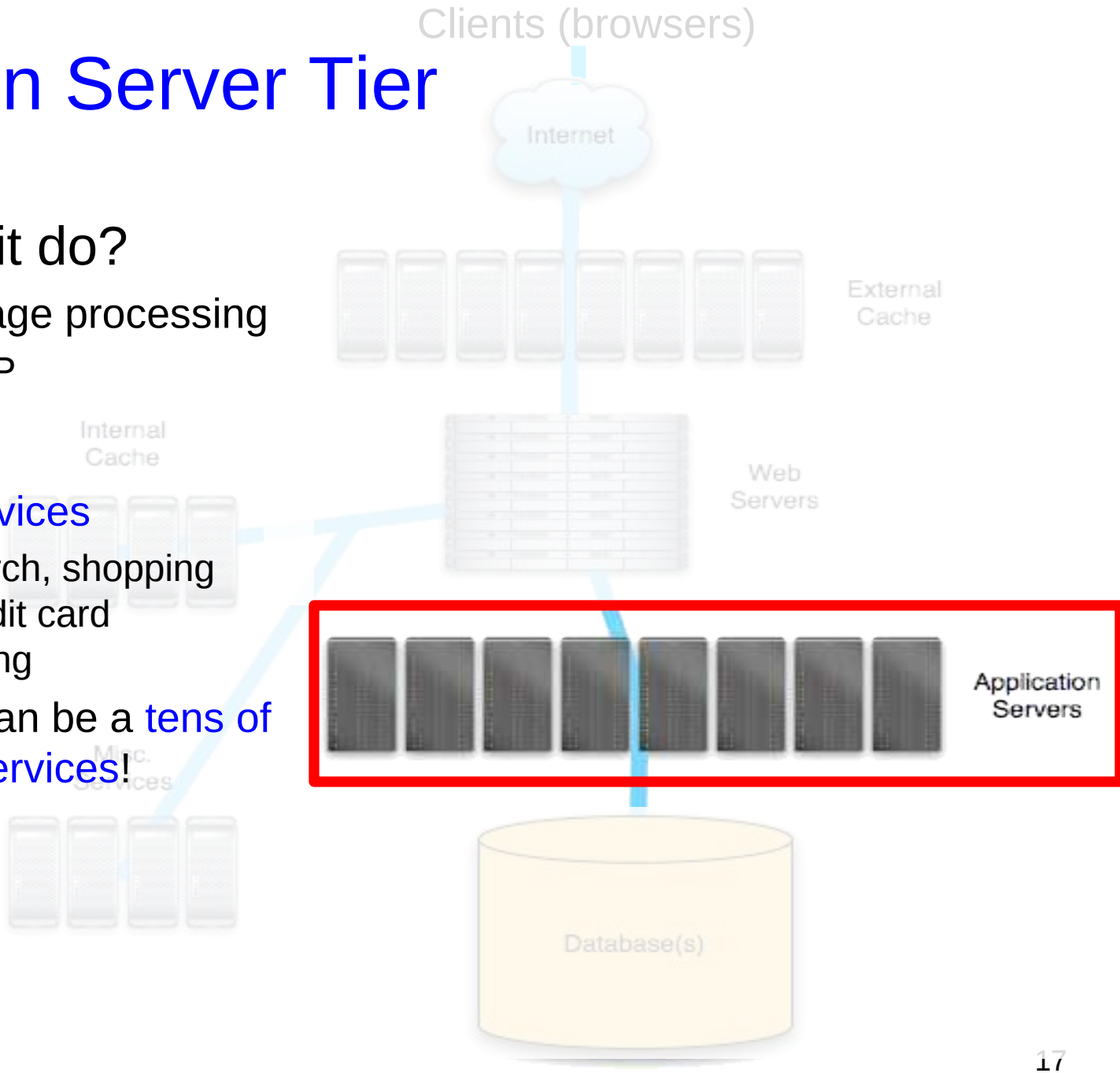
Front-end Tier

- Hardware requirements
  - Lots and lots of memory
    - Memory is main bottleneck in web serving
  - CPU depends on usage
    - Dynamic pages need CPU
    - Static pages need little CPU
  - Cheap slow disk is enough



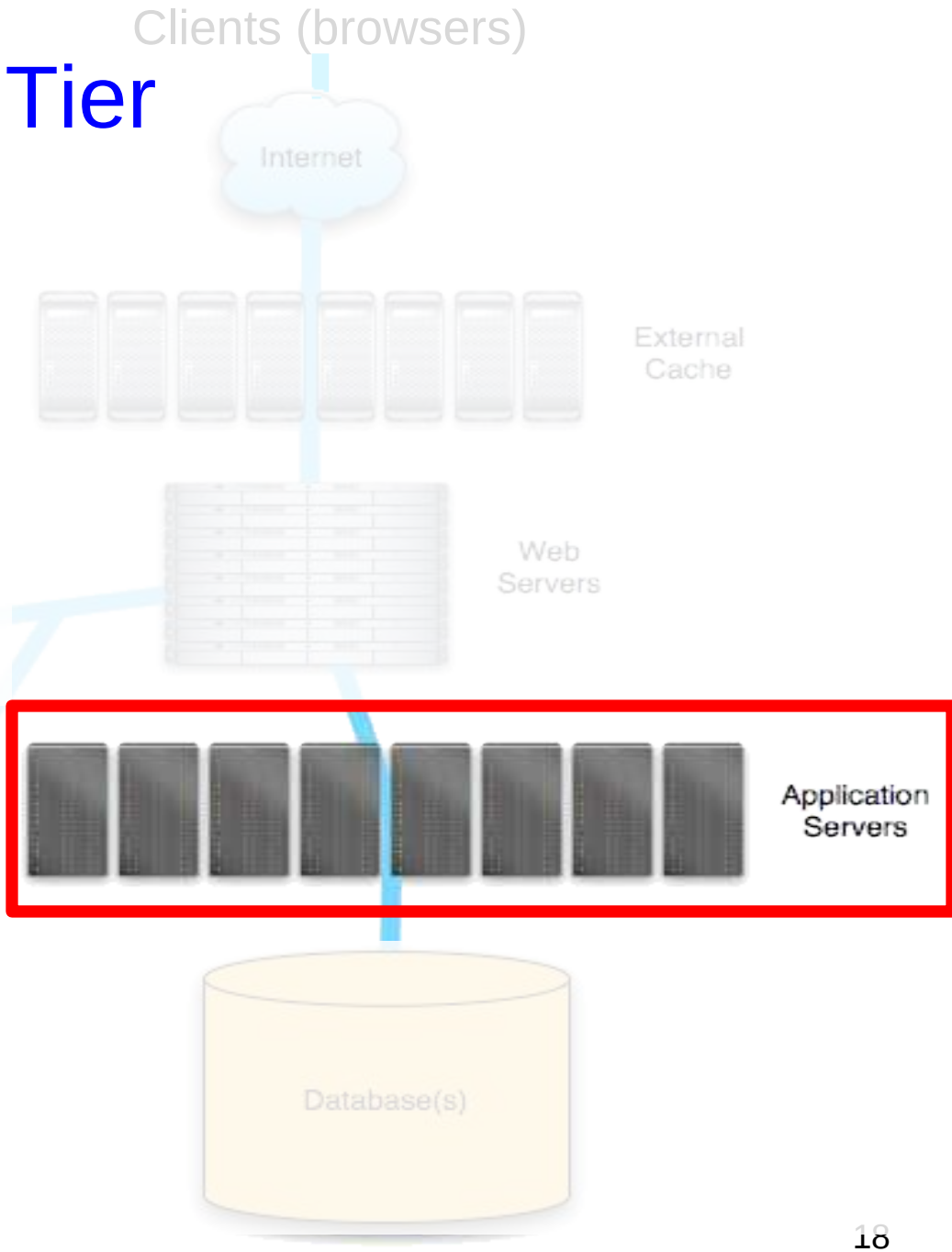
Application Server Tier

- What does it do?
  - Dynamic page processing
    - ASP, JSP
    - Servlets
  - Internal services
    - Eg.: search, shopping cart, credit card processing
    - There can be a tens of these services!



Application Server Tier

- How does it work?
  1. Web Tier generates the request using
     - *Home-brewed RPC*
     - *REST*
     - *Corba*
     - Java RMI
     - *SOAP*
     - *XMLRPC*
  2. App Server processes request and responds

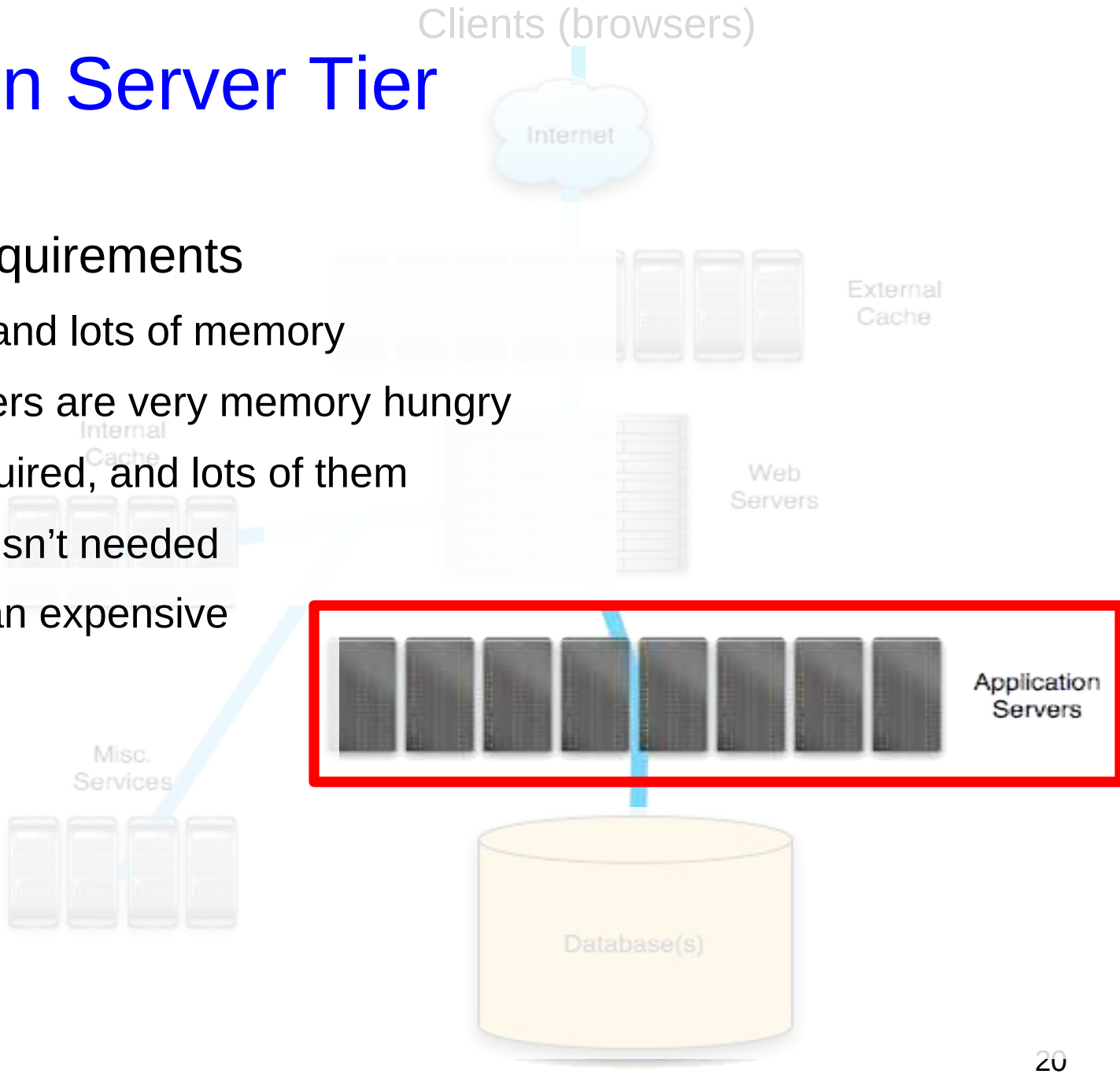


Application Server Tier

- Decoupling of services is GOOD
  - Manage Complexity using well-defined APIs
- BUT: remote calling overhead can be expensive!
  - Marshaling of data, sockets, net latency, …
  - SOAP, XMLRPC … don't scale that well…
  - We'll talk about some efficient RPC systems next week

Application Server Tier

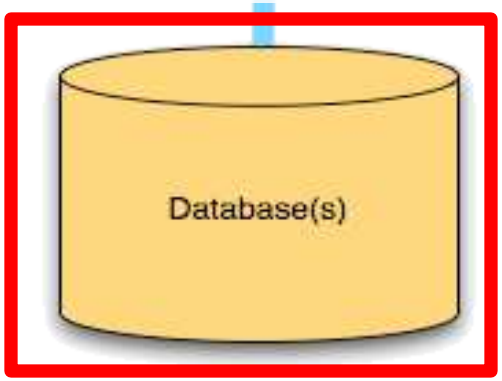
- Hardware requirements
  - Lots and lots and lots of memory
    - App Servers are very memory hungry
  - Fast CPU required, and lots of them
  - Disk typically isn't needed
  - (This will be an expensive machine.)



Database Tier

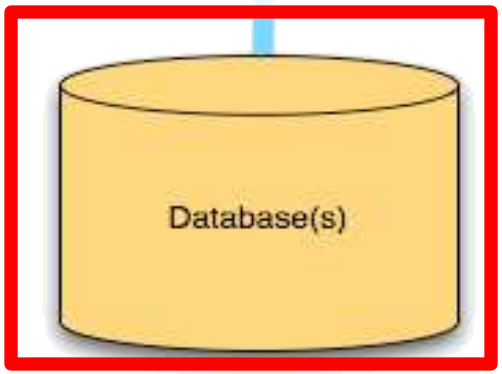
- What is this?
  - Relational databases (distributed or not)
    - PostgreSQL, SQLite, Oracle, MySQL, Berkeley DB
  - Non-relational databases or distributed file systems
    - Bigtable, Megastore, MongoDB, Hadoop Hbase, HDFS, …

- Tradeoffs:
  - Relational databases don't scale that well, but provide convenient interface, sound properties (e.g., strong consistency)
  - Non-relational DBs scale better

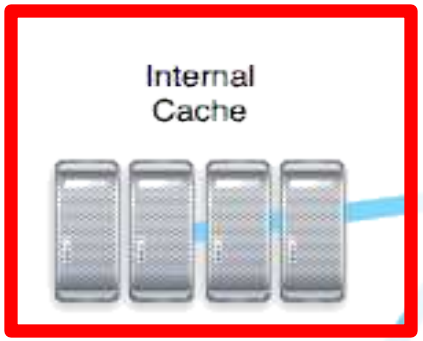


Database Tier

- Hardware Requirements
  - Entirely dependent upon application
  - Likely to be your most expensive machine(s)
  - Tons of memory
  - Large disks
  - Spindles galore
  - RAID is useful for redundancy

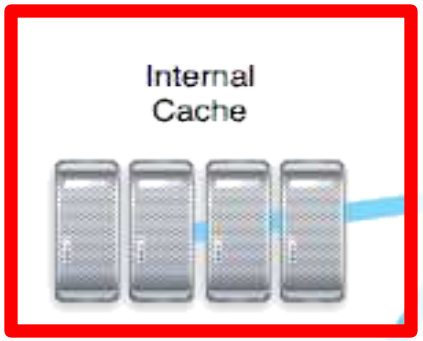


Internal Caching Tier



- What is this?
  - Object cache (e.g., intermediary app-level results)
- What applications?
  - Memcached
  - Application-level caching inside the application servers

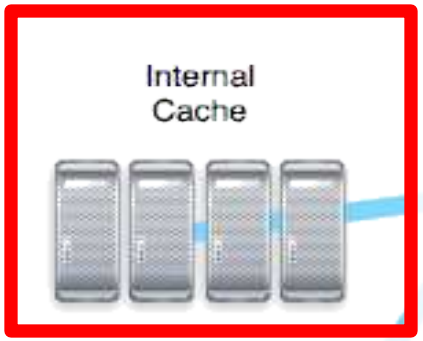
Internal Caching Tier



- What does it do?
  - Caches objects closer to the Application or Web Tiers
  - Tuned for the application
    - The external cache is generic
  - Very fast access (<1ms)

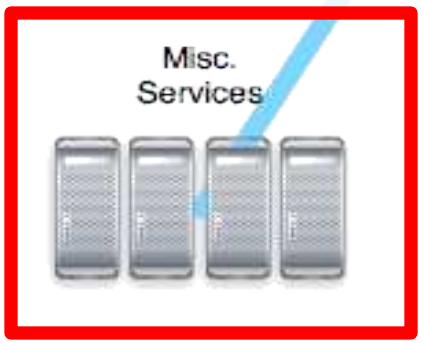
Internal Caching Tier

- Hardware requirements
  - Lots of Memory
  - Little or no disk
  - Moderate to low CPU
  - Fast Network



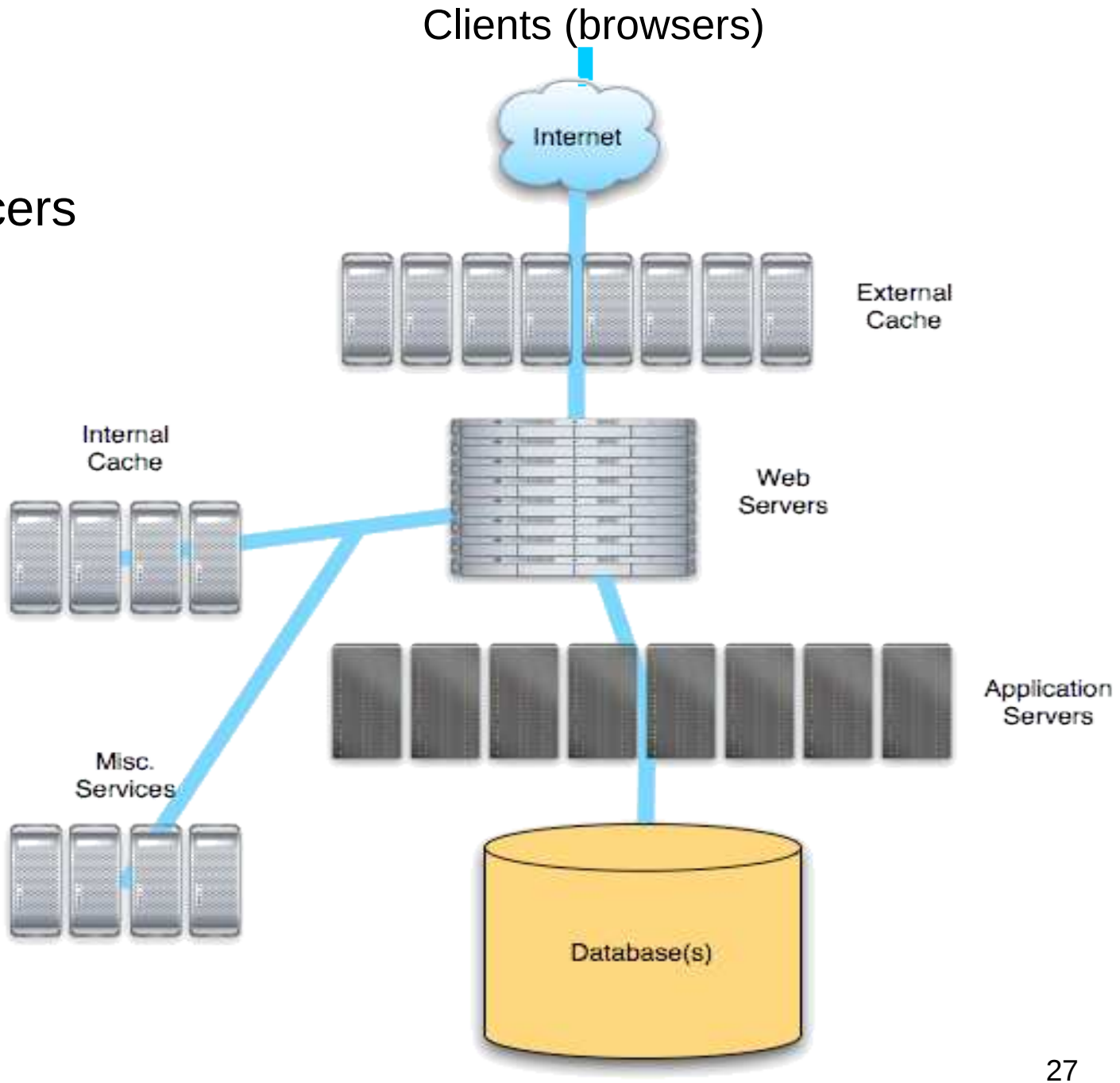
Misc. Services

- Lots of extra services commonly used in Web services
  - DNS
  - Time synchronization (we'll see why this is very important)
  - System health monitoring
  - Intrusion detection systems
  - …



The Glue

- Load balancers
- Routers
- Switches
- Firewalls



Whew! What Did We Learn?

- Web architectures are complex
- But there are well-known solutions
- There are lots of tradeoffs and understanding the workload is key in choosing the right product to use at each layer

- Each layer has distinct hardware requirements and likely distinct bottlenecks
  - Except for RAM, which is very popular

- What does the last observation tell us?

Next time

- Another case study: Cloud computing
  - What it means and how it began

- Remember to look on website for HW2
  - HW 2 is graded and is MUCH longer than HW1
  - So start it ASAP after it's released
  - TA will give an overview next time of YFS series

Code Listing 1: CGI Script

http://www.djangobook.com/en/1.0/chapter01/

```
#!/usr/bin/python

import MySQLdb

print "Content-Type: text/html"
print
print "<html><head><title>Books</title></head>"
print "<body>"
print "<h1>Books</h1>"
print "<ul>"

connection = MySQLdb.connect(user='me', passwd='letmein', db='my_db')
cursor = connection.cursor()
cursor.execute("SELECT name FROM books ORDER BY pub_date DESC LIMIT 10")
for row in cursor.fetchall():
    print "<li>%s</li>" % row[0]

print "</ul>"
print "</body></html>"

connection.close()
```

Code Listing 2: Django

http://www.djangobook.com/en/1.0/chapter01/

```
# models.py (the database tables)

from django.db import models

class Book(models.Model):
    name = models.CharField(maxlength=50)
    pub_date = models.DateField()


# views.py (the business logic)

from django.shortcuts import render_to_response
from models import Book

def latest_books(request):
    book_list = Book.objects.order_by('-pub_date')[:10]
    return render_to_response('latest_books.html', {'book_list': book_list})


# (continued on other side)
```

```
# urls.py (the URL configuration)

from django.conf.urls.defaults import *
import views

urlpatterns = patterns('',
    (r'latest/$', views.latest_books),
)


# latest_books.html (the template)

<html><head><title>Books</title></head>
<body>
<h1>Books</h1>
<ul>
{% for book in book_list %}
<li>{{ book.name }}</li>
{% endfor %}
</ul>
</body></html>
```