# Domain Naming System

Lecture

# Overview

- Introduction to the DNS
- DNS Components
- DNS Structure and Hierarchy
- The DNS in Context

# DNS History (1)

- ARPANET utilized a central file HOSTS.TXT
  - Contains names to addresses mapping
  - Maintained by SRI's NIC (*Stanford-Research-Institute: Network-Information-Center)*
- Administrators email changes to NIC
  - NIC updates HOSTS.TXT periodically
- Administrators FTP (download) HOSTS.TXT

# DNS History (2)

- As the system grew, HOSTS.TXT had problems with:
  - Scalability (traffic and load)
  - Name collisions
  - Consistency

- In **1984**, Paul Mockapetris released the first version (**RFCs 882** and **883**, superseded by **1034** and **1035** …)

# The DNS is…

- The "**Domain Name System**"
- What Internet users use **to reference anything by name** on the Internet.
- The mechanism by which Internet software translates **names to** attributes such as **addresses**

# The DNS is also…

- A globally distributed, scalable, reliable **database**
- Comprised of three components
  - A "**name space**"
  - **Servers** making that name space available
  - **Resolvers** (clients) which query the servers about the name space

# DNS as a Lookup Mechanism

- Users generally prefer names to numbers
- Computers prefer numbers to names
- DNS provides the mapping between the two
  - I have “x”, give me “y”

# DNS as a Database

- Keys to the database are "**domain names**"
  - www.test.com, 18.in-addr.arpa, 6.4.e164.arpa
- Over 200,000,000 domain names stored
- Each **domain** name contains **one or more attributes**
  - Known as "***resource records***"
- Each attribute **individually retrievable**

# Global Distribution

- Data is **maintained locally**, but **retrievable globally**
  - No single computer has all DNS data
- **DNS lookups** can be performed by **any device**
- Remote DNS data is **locally cachable** to improve performance

# Scalability

- No limit to the **size of the database**
- No limit to the **number of queries**
  - Tens of thousands of queries handled easily every second
- **Queries distributed** among masters, slaves, and caches

# Reliability

- Data is replicated
  - Data from master is copied to multiple slaves(replicas)
- Clients can query
  - Master server
  - Any of the copies at slave servers
- Clients will typically query local caches
- DNS protocols can use either UDP or TCP
  - If UDP, DNS protocol handles retransmission, sequencing, etc.

# Loose Coherency

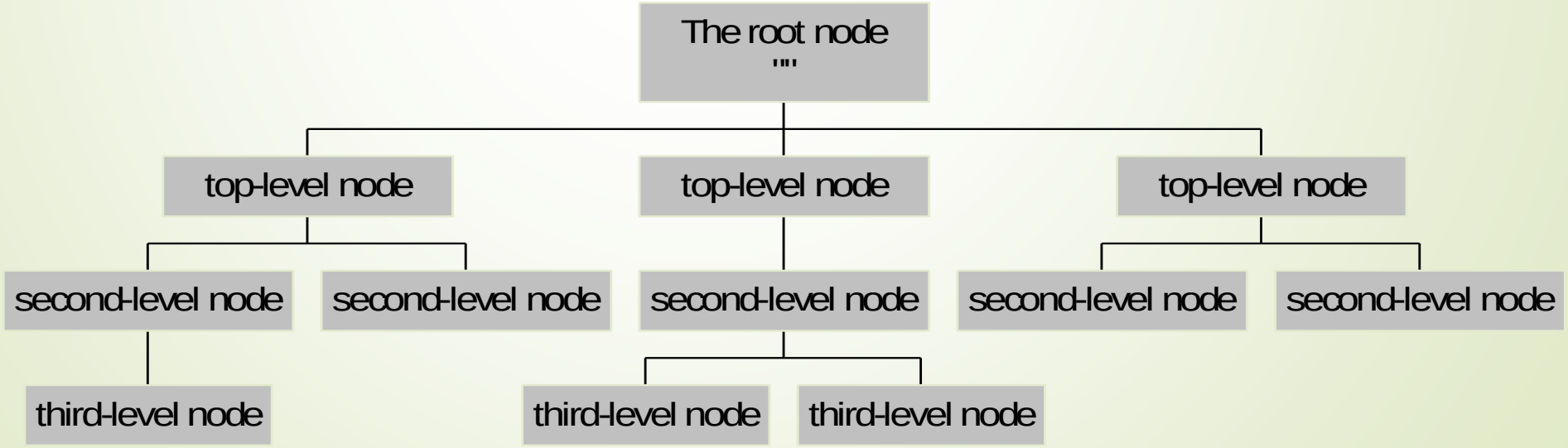
- Each version of a subset of the database (a **zone**) has a **serial number**
  - The serial number is **incremented** on each **database change**
- **Changes** to the **master** copy of the database are **propagated to replicas** according to timing set by the zone administrator
- **Cached** data **expires** according to **timeout** set by **zone administrator**

# Dynamicity

- Database can be **updated dynamically**
  - Add/delete/modify of any record
  - **Only master** can be dynamically updated
- **Modification** of the master database **triggers replication**

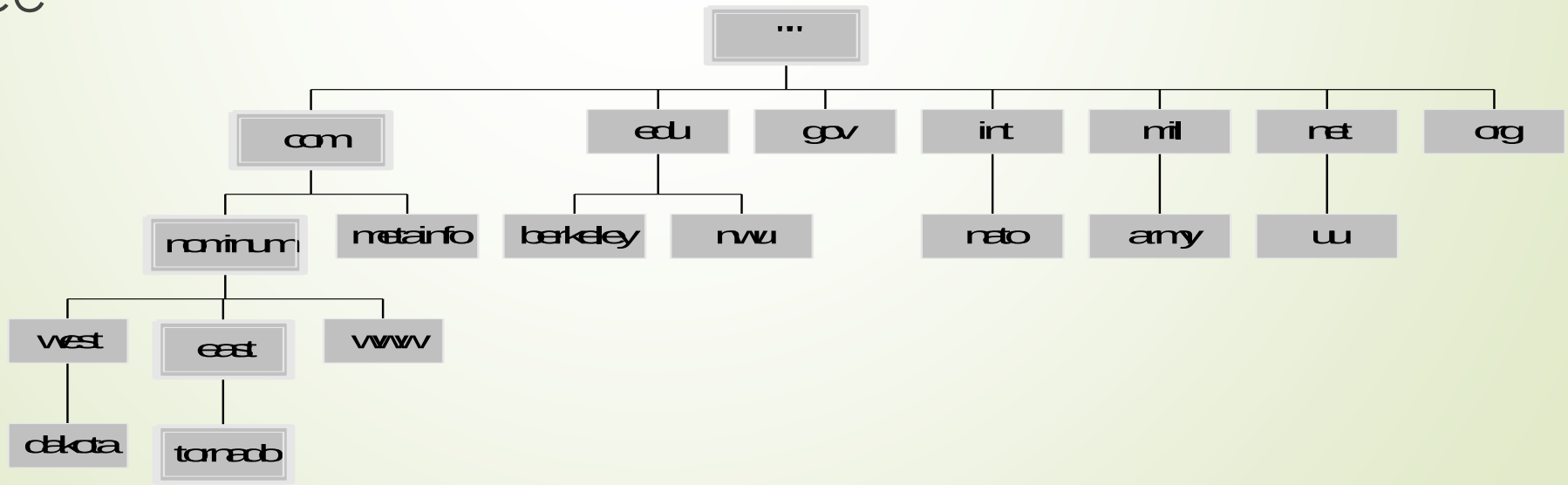
# The Name Space

- The ***name space*** is the structure of the DNS database
  - An **inverted tree** with the **root node** at the top
- Each node has a label
  - The root node has a null label, written as “”



# Domain Names

- A *domain name* is the sequence of labels from a node to the root, separated by dots (“.”s), read left to right
  - The name space has a maximum depth of 127 levels
  - Domain names are limited to 255 characters in length
- A node’s domain name identifies its position in the name space



# Subdomains

- One domain is a subdomain of another if its domain name ends in the other's domain name
  - So *sales.nominum.com* is a subdomain of
    - *nominum.com & com*
  - *nominum.com* is a subdomain of *com*

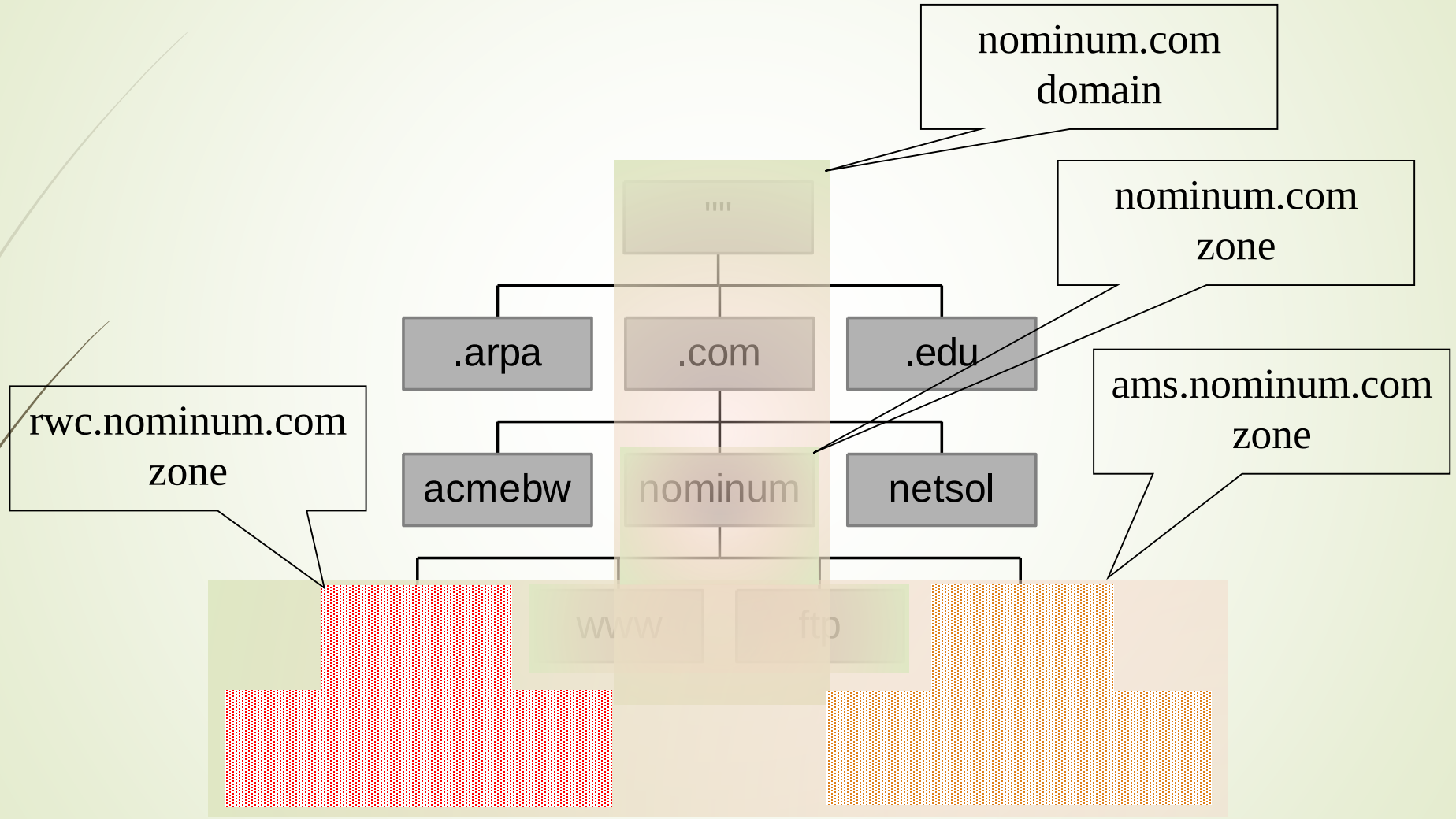
# Delegation

- Administrators can create subdomains to group hosts
  - According to geography, organizational affiliation etc.
- An administrator of a domain can delegate responsibility for managing a subdomain to someone else
- The parent domain retains links to the delegated subdomains

# Delegation Creates Zones

- Each time an administrator delegates a subdomain, a new unit of administration is created
  - The subdomain and its parent domain can now be administered independently
  - These units are called *zones*
  - The boundary between zones is a point of delegation in the name space
- Delegation is good: it is the key to scalability

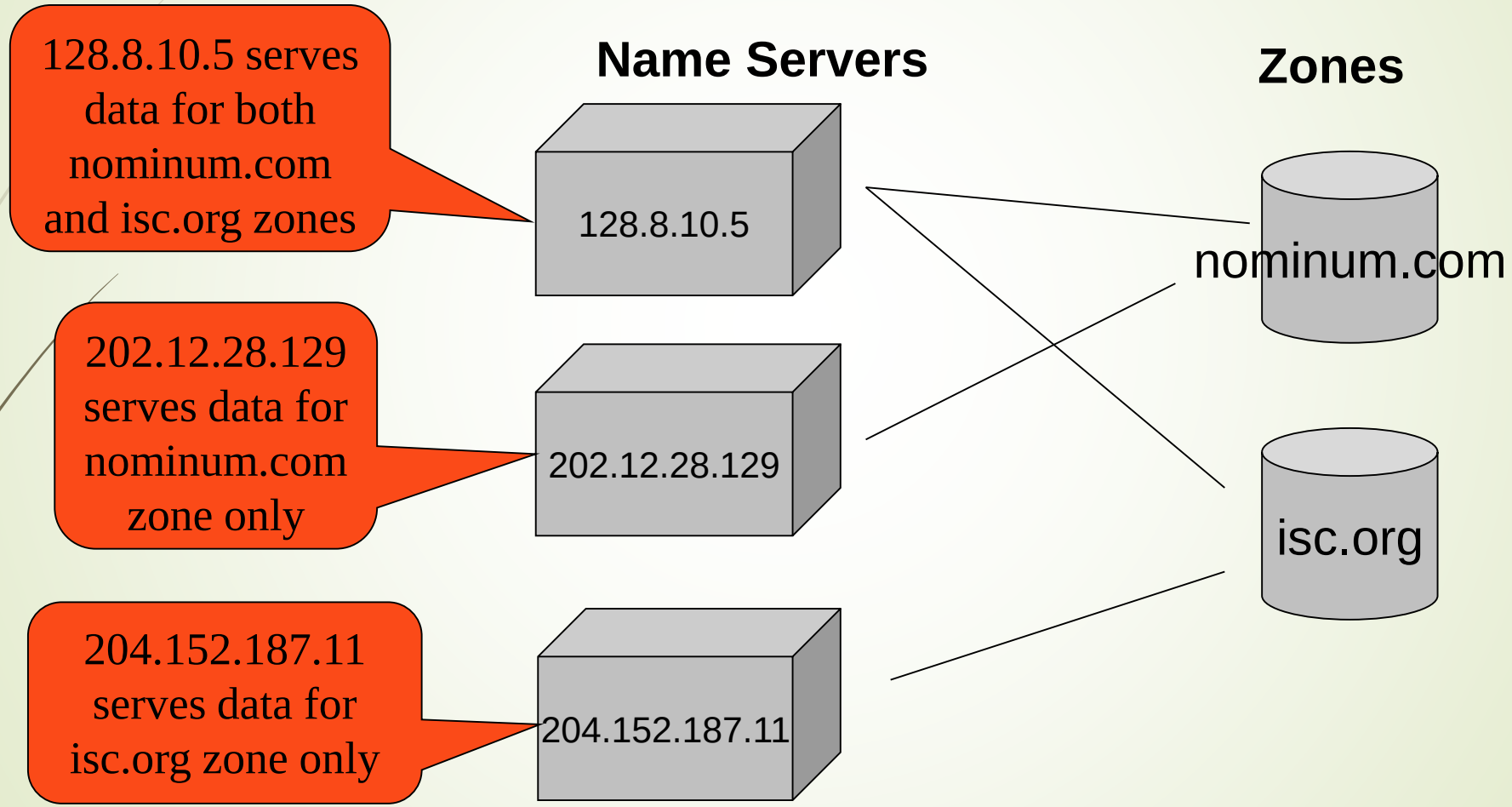
# Dividing a Domain into Zones



# Name Servers

- Name servers store information about the name space in units called “zones”
  - The name servers that load a complete zone are said to “have authority for” or “be authoritative for” the zone
- Usually, more than one name server are **authoritative** for the same zone
  - This ensures redundancy and spreads the load
- Also, a single name server may be authoritative for many zones

# Name Servers and Zones



# Types of Name Servers

- Two main types of servers
  - Authoritative – maintains the data
    - Master – where the data is edited
    - Slave – where data is replicated to
  - Caching – stores data obtained from an authoritative server
- No special hardware necessary

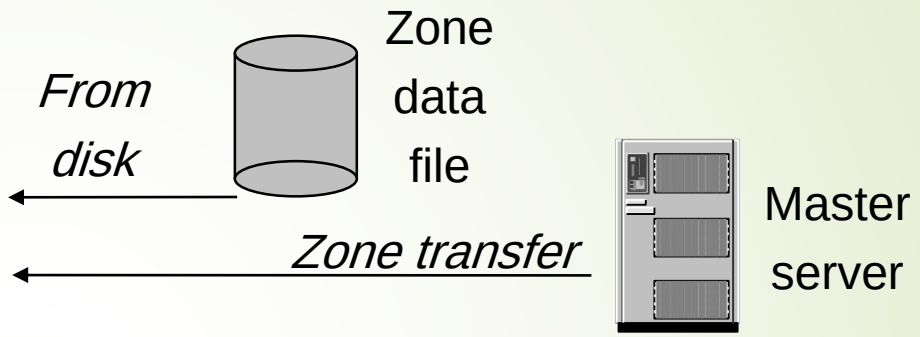
# Name Server Architecture

- You can think of a name server as part of:
  - *database server,* answering queries about the parts of the name space it knows about (i.e., is authoritative for),
  - *cache,* temporarily storing data it learns from other name servers, and
  - *agent,* helping resolvers and other name servers find data

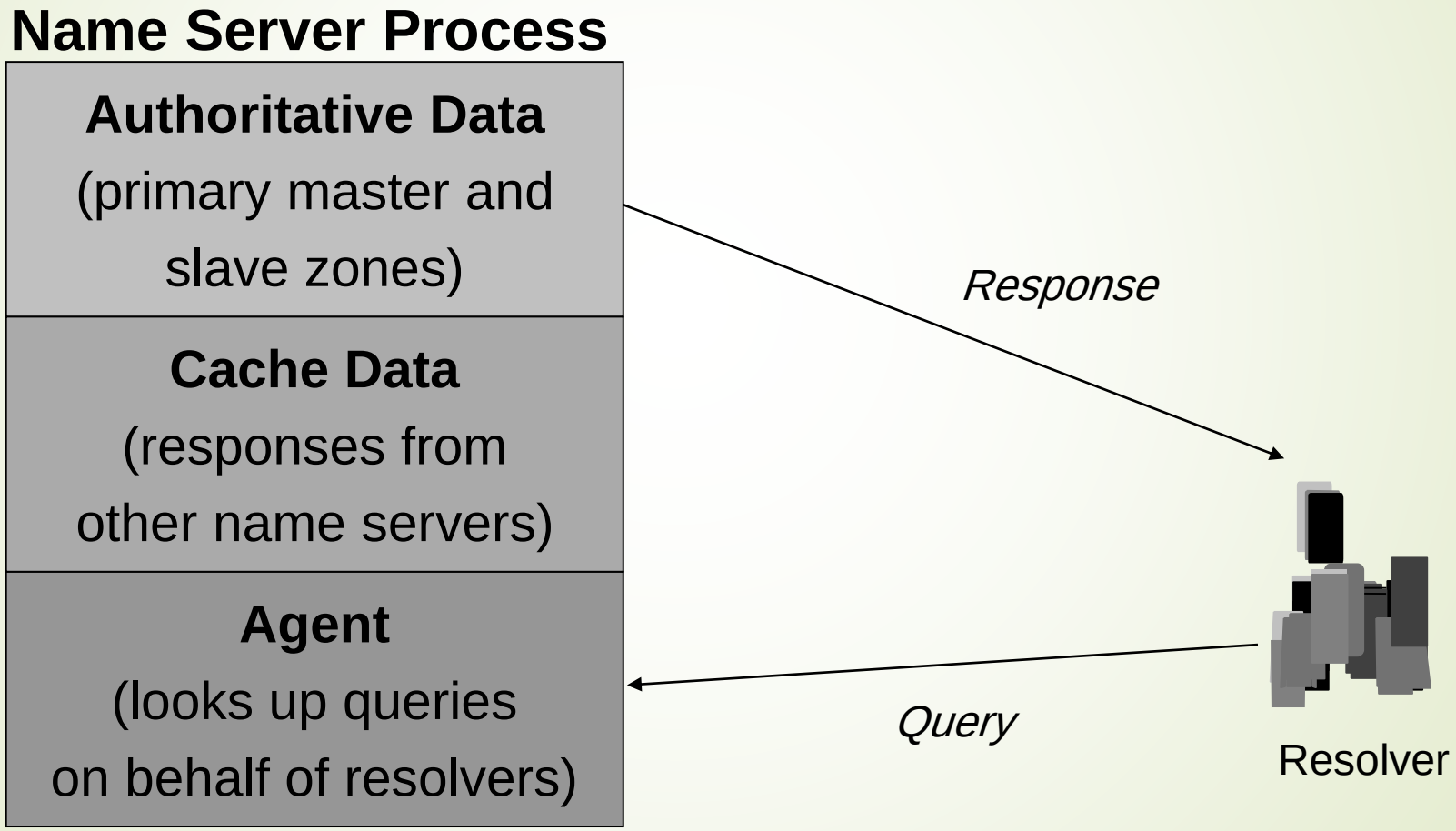
# Name Server Architecture

**Name Server Process**

| |
|---|
| **Authoritative Data** (primary master and slave zones) |
| **Cache Data** (responses from other name servers) |
| **Agent** (looks up queries on behalf of resolvers) |



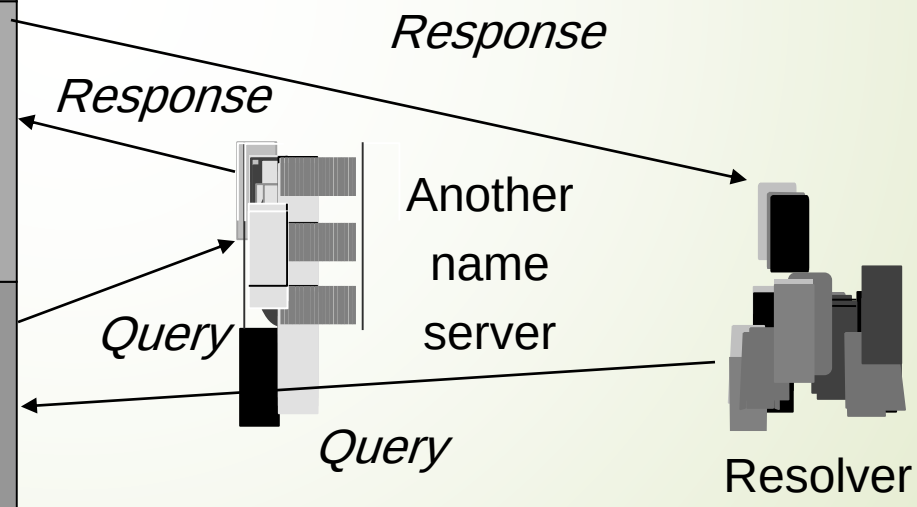
# Authoritative Data



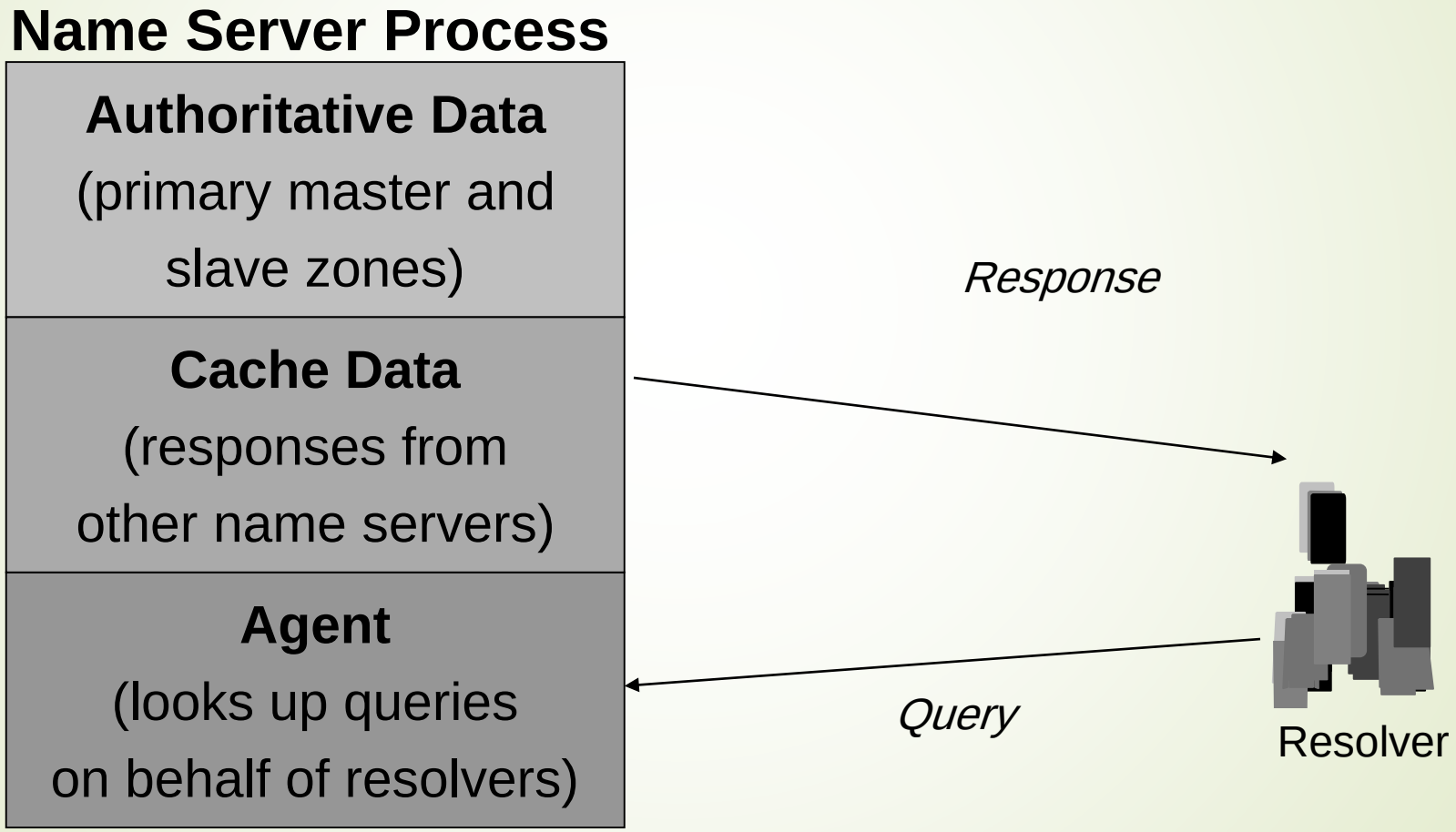
# Using Other Name Servers

**Name Server Process**

| |
|---|
| **Authoritative Data** (primary master and slave zones) |
| **Cache Data** (responses from other name servers) |
| **Agent** (looks up queries on behalf of resolvers) |



# Cached Data


Name Server Process
Authoritative Data
(primary master and
slave zones)
Cache Data
(responses from
other name servers)
Agent
(looks up queries
on behalf of resolvers)
Response
Query
Resolver

# Name Resolution

- *Name resolution* is the process by which resolvers and name servers cooperate to find data in the name space
- Closure mechanism for DNS?
  - Starting point: the names and IP addresses of the name servers for the root zone (the "root name servers")
  - The root name servers know about the top-level zones and can tell name servers whom to contact for all TLDs

# Name Resolution

- A DNS query has three parameters:
  1. A domain name (e.g., *www.nominum.com*),
     - Remember, every node has a domain name!
  2. A class (e.g., *IN*), and
  3. A type (e.g., *A*)

  http://network-tools.com/nslook/
- Upon receiving a query from a resolver, a name server
  1. looks for the answer in its authoritative data and its cache
  2. If step 1 fails, the answer must be looked up

# The Resolution Process

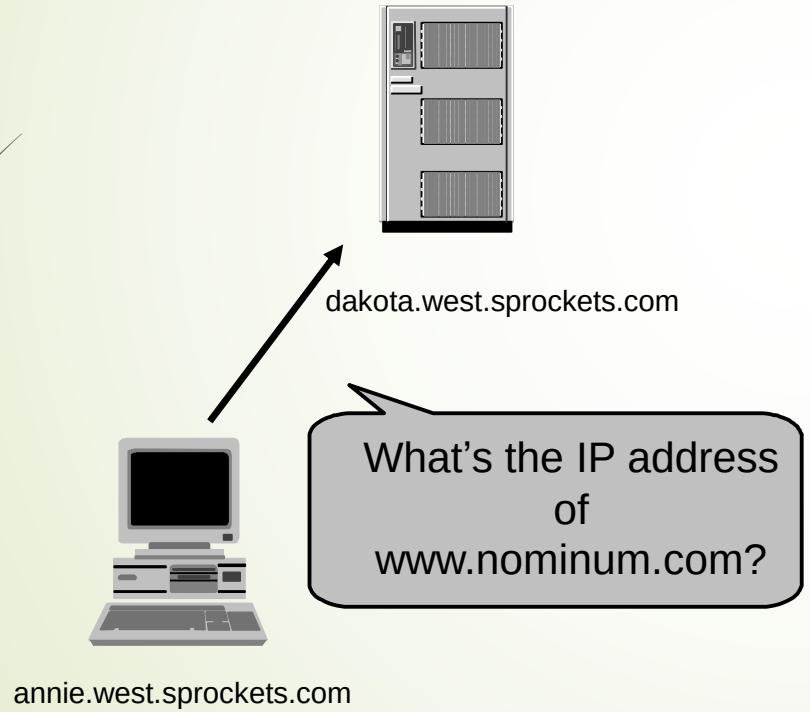
- Let's look at the resolution process step-by-step:

annie.west.sprockets.com

```
ping www.nominum.com.
```

# The Resolution Process

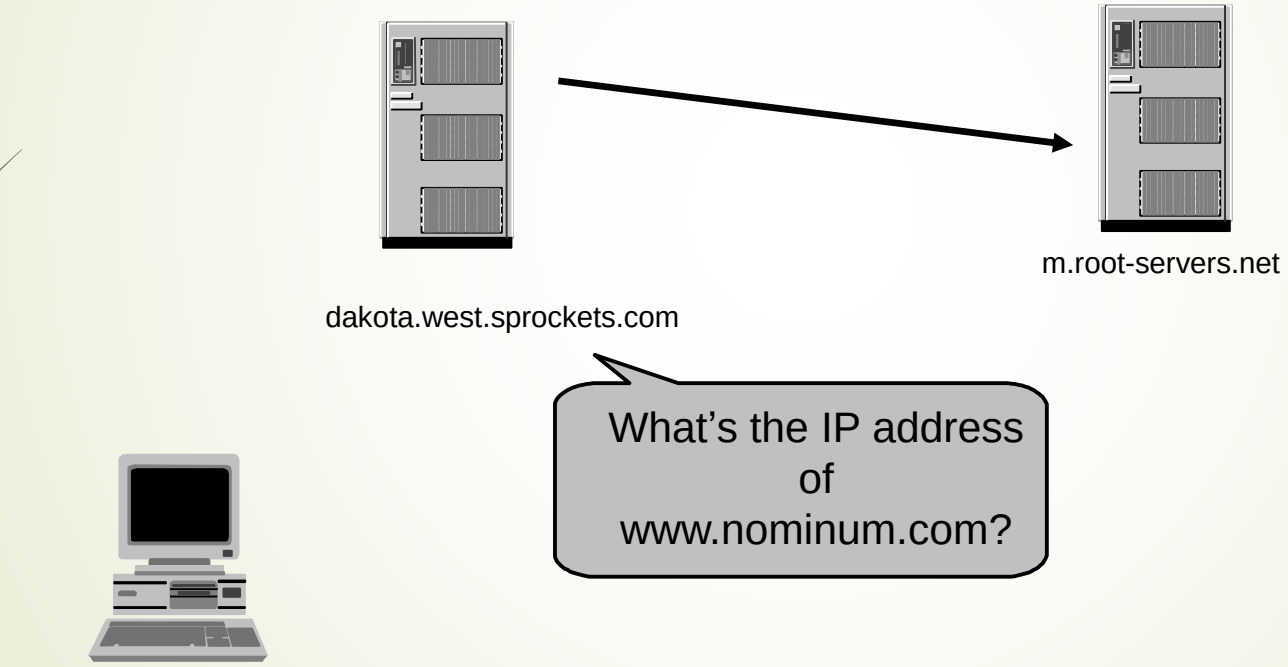
- The workstation *annie* asks its configured name server, *dakota,* for *www.nominum.com's* address




```
ping www.nominum.com.
```

# The Resolution Process

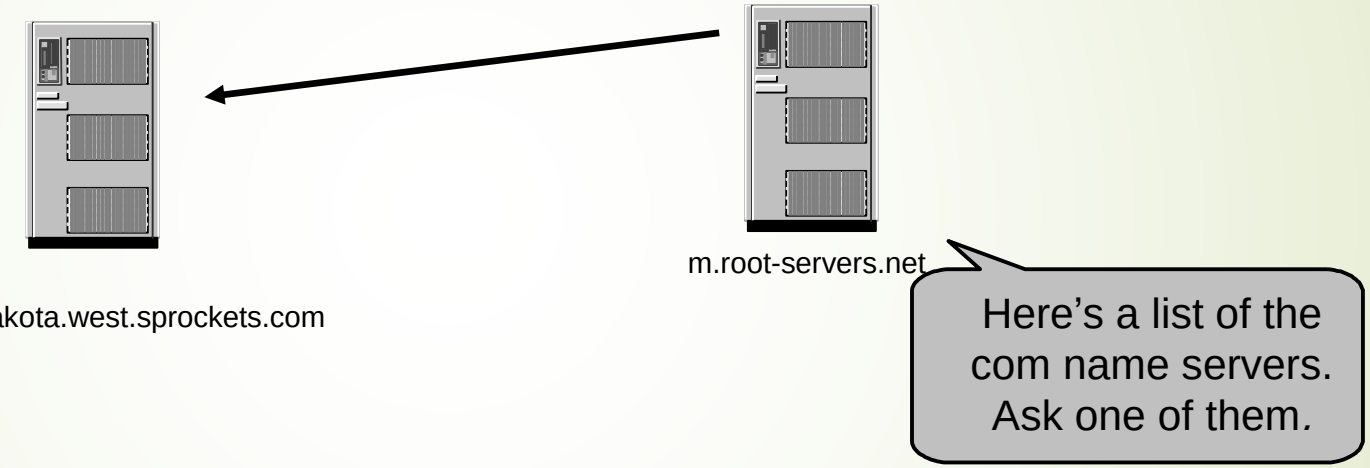
- The name server *dakota* asks a root name server, *m*, for *www.nominum.com's* address




```
ping www.nominum.com.
```

# The Resolution Process

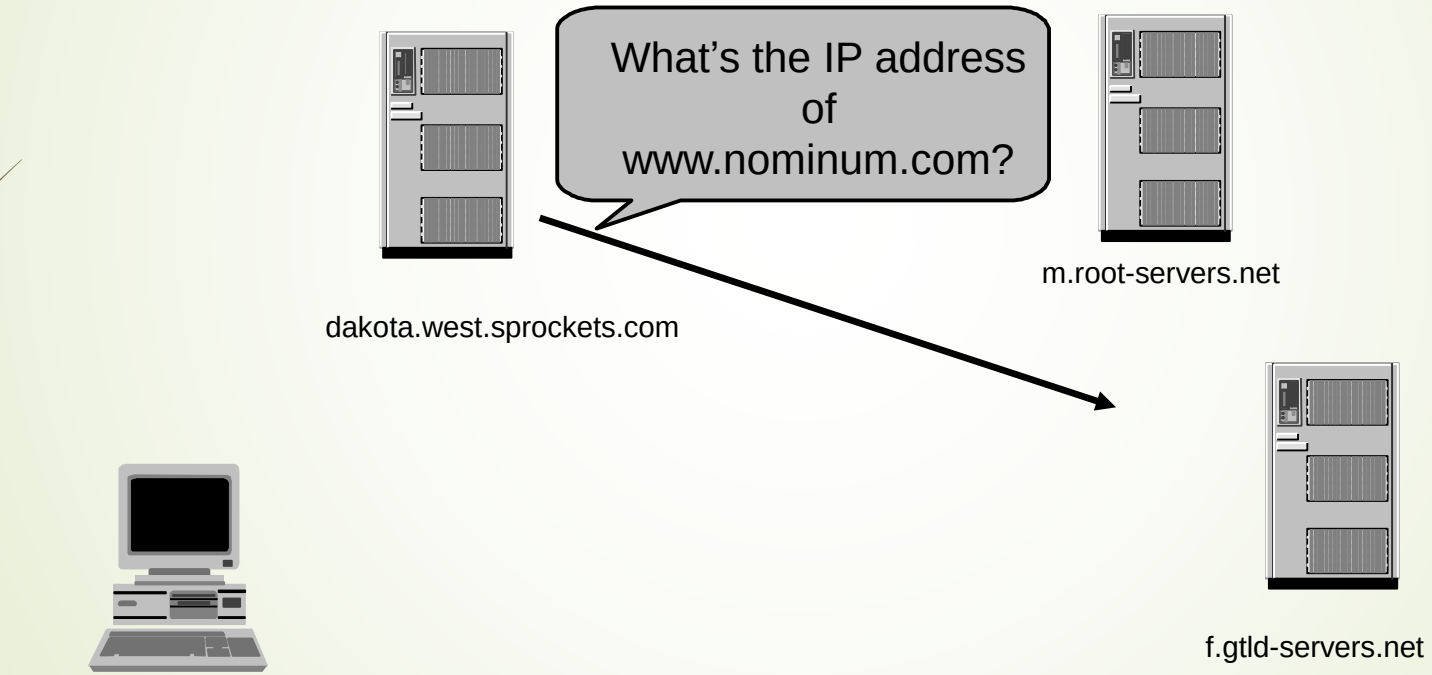
- The root server *m* refers *dakota* to the *com* name servers
- This type of response is called a "referral"




```
ping www.nominum.com.
```

# The Resolution Process

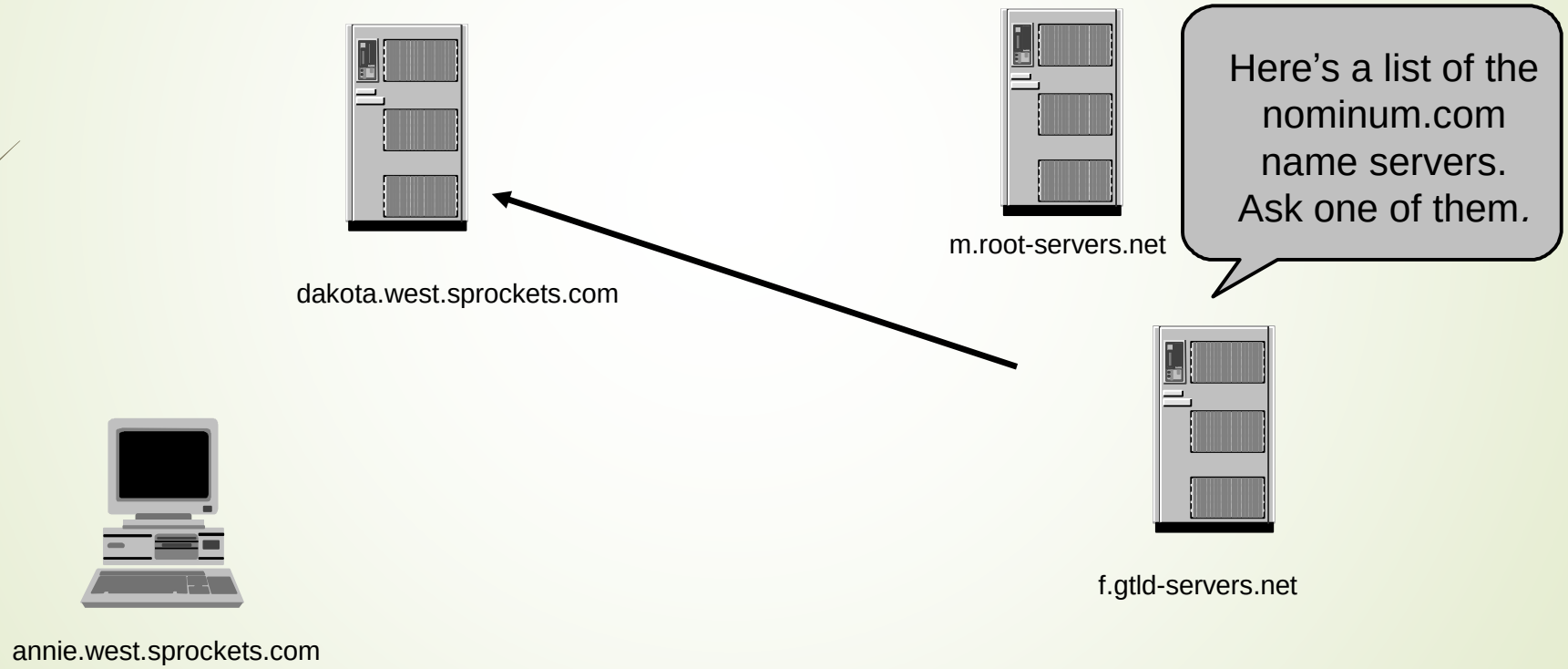
- The name server *dakota* asks a *com* name server, *f*, for *www.nominum.com's* address




```
ping www.nominum.com.
```

# The Resolution Process

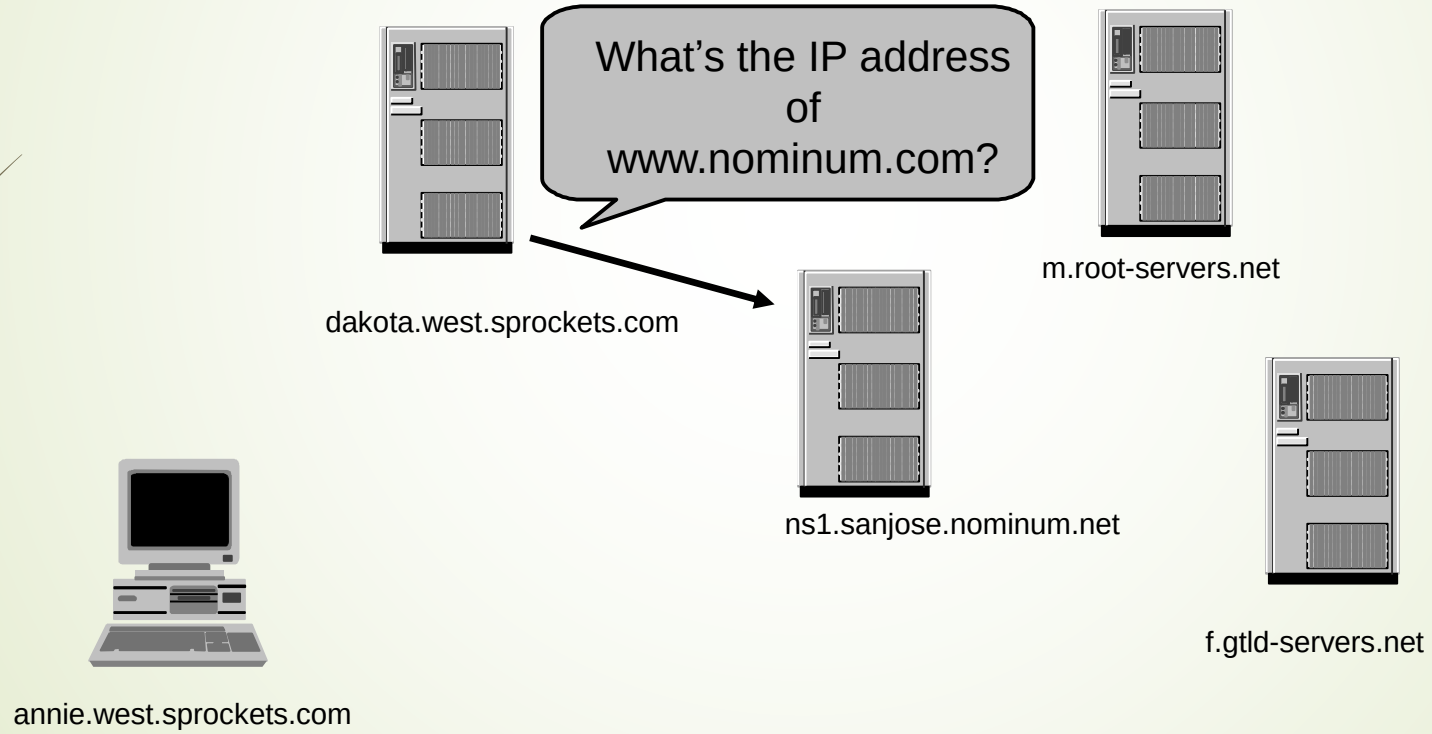
- The *com* name server *f* refers *dakota* to the *nominum.com* name servers




```
ping www.nominum.com.
```

# The Resolution Process

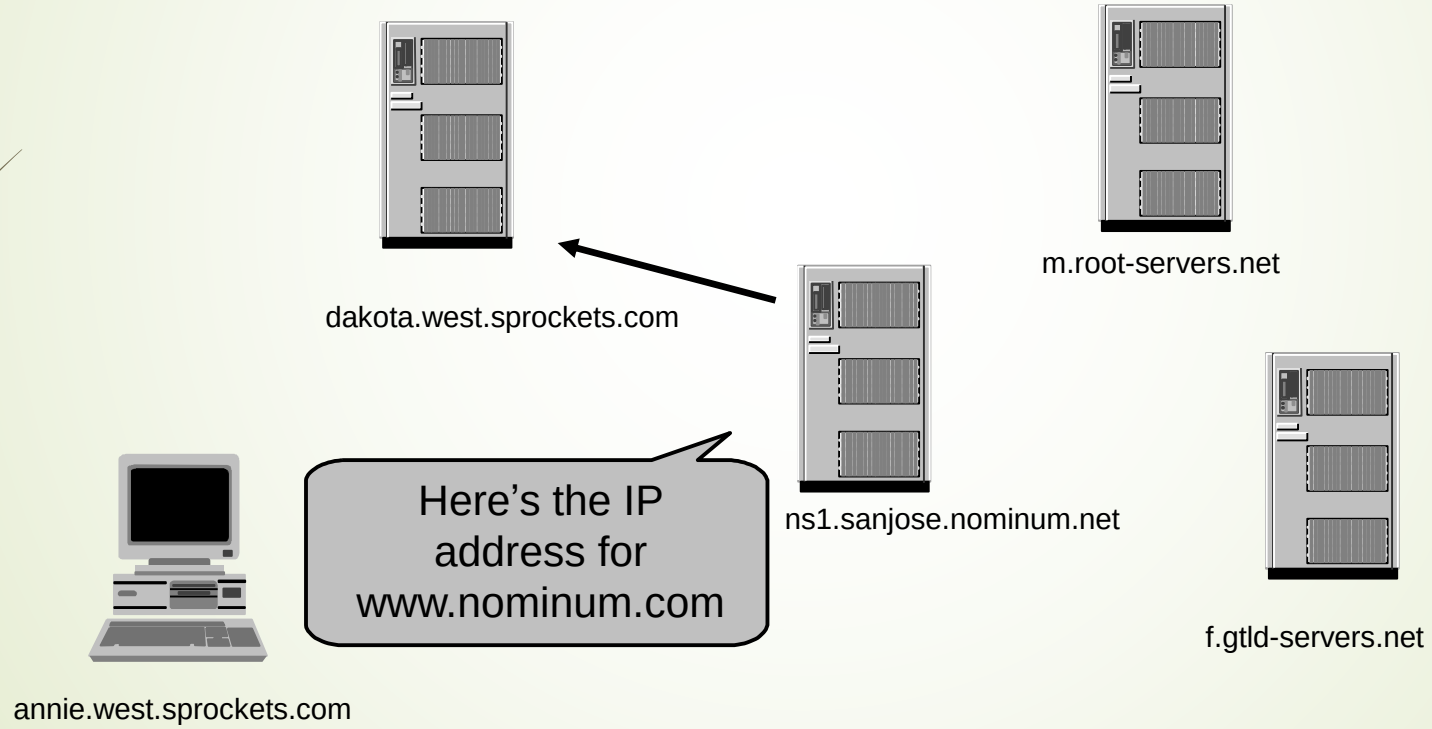
- The name server *dakota* asks a *nominum.com* name server, *ns1.sanjose*, for *www.nominum.com's* address




```
ping www.nominum.com.
```

# The Resolution Process

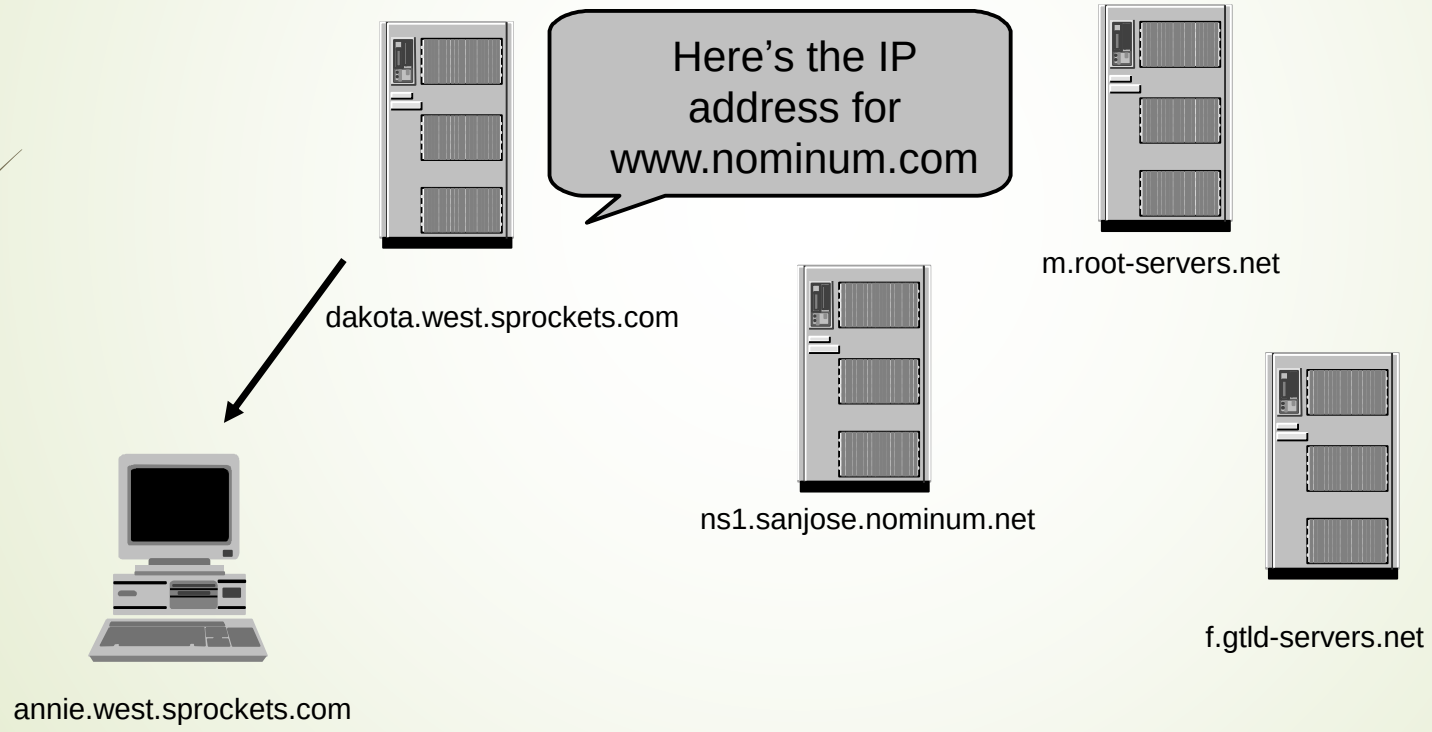
- The *nominum.com* name server *ns1.sanjose* responds with *www.nominum.com's* address




```
ping www.nominum.com.
```

# The Resolution Process

- The name server *dakota* responds to *annie* with *www.nominum.com's* address




```
ping www.nominum.com.
```

# Resolution Process (Caching)

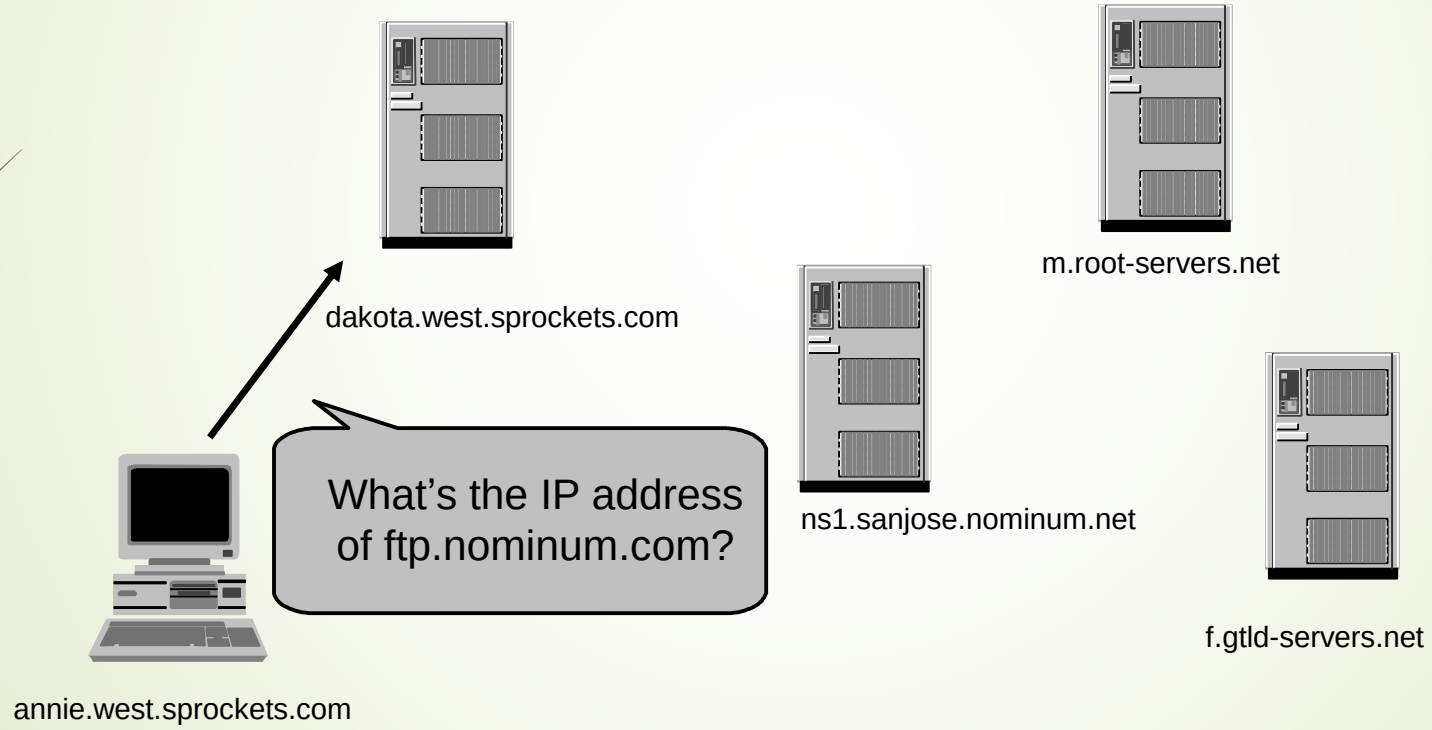
- After the previous query, the name server *dakota* now knows:
  - The names and IP addresses of the *com* name servers
  - The names and IP addresses of the *nominum.com* name servers
  - The IP address of *www.nominum.com*
- Let's look at the resolution process again

annie.west.sprockets.com

```
ping ftp.nominum.com.
```

# Resolution Process (Caching)

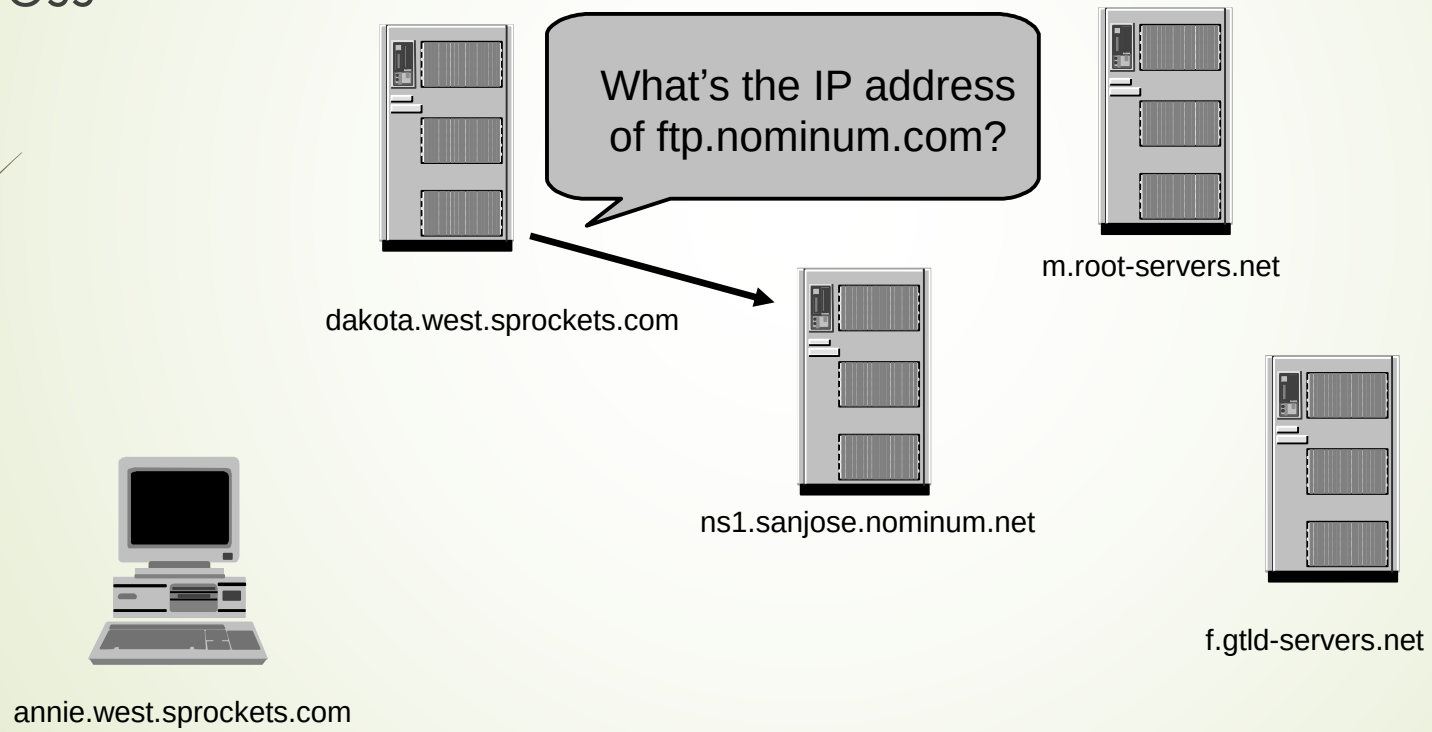
- The workstation *annie* asks its configured name server, *dakota,* for *ftp.nominum.com's* address




```
ping ftp.nominum.com.
```

# Resolution Process (Caching)

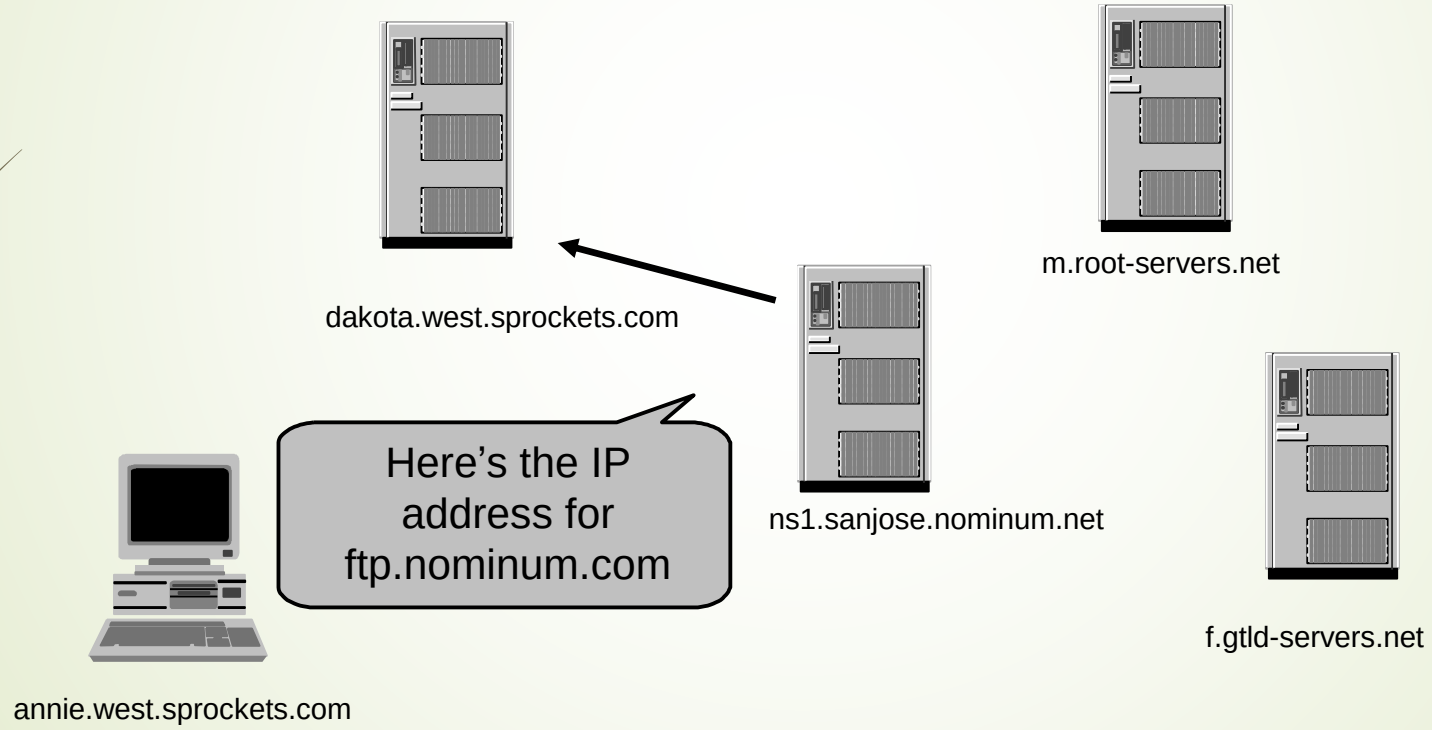
- *dakota* has **cached** a NS record indicating *ns1.sanjose* is an *nominum.com* name server, so it asks it for *ftp.nominum.com's* address




```
ping ftp.nominum.com.
```

# Resolution Process (Caching)

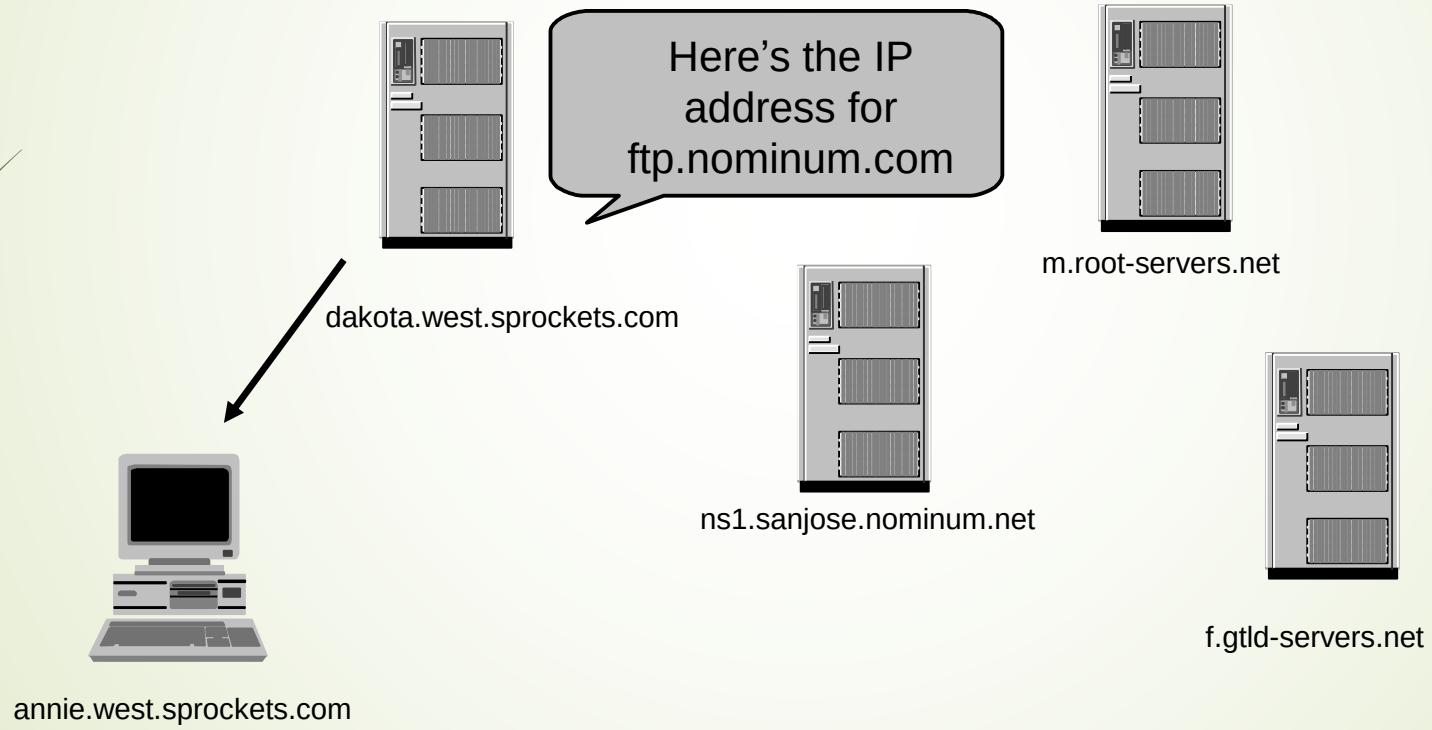
- The *nominum.com* name server *ns1.sanjose* responds with *ftp.nominum.com's* address




```
ping ftp.nominum.com.
```

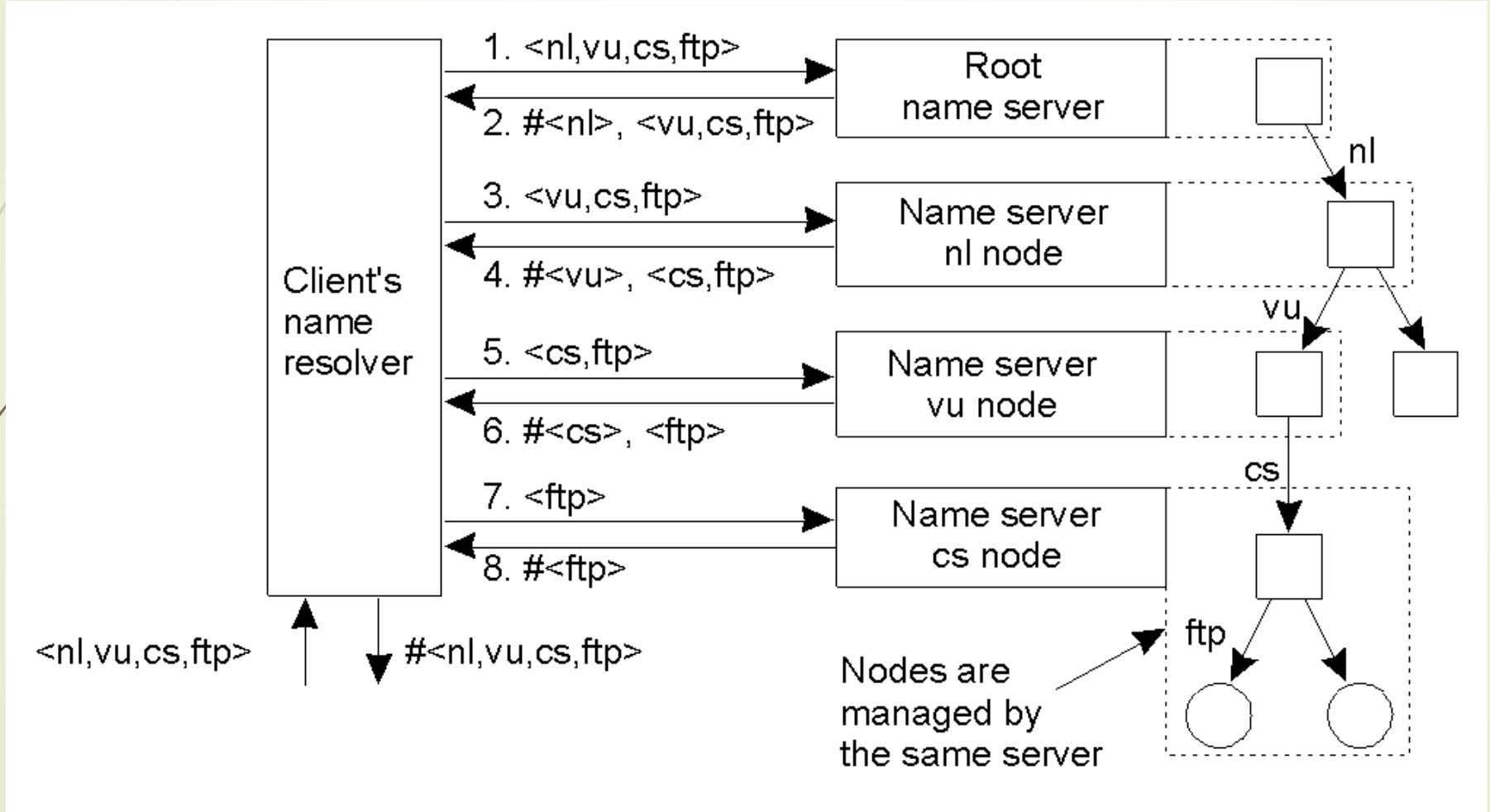
# Resolution Process (Caching)

- The name server *dakota* responds to *annie* with *ftp.nominum.com's* address

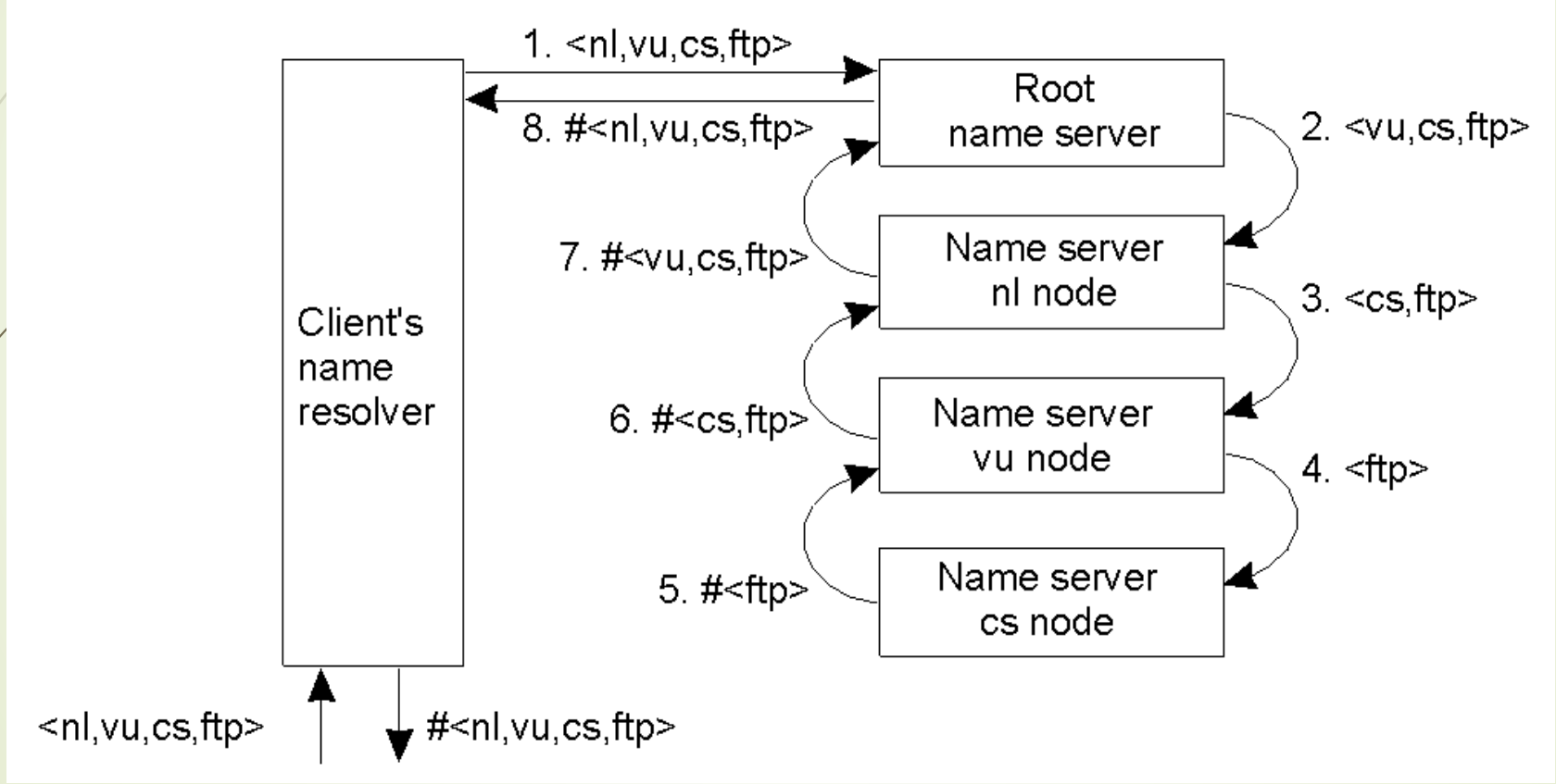



```
ping ftp.nominum.com.
```

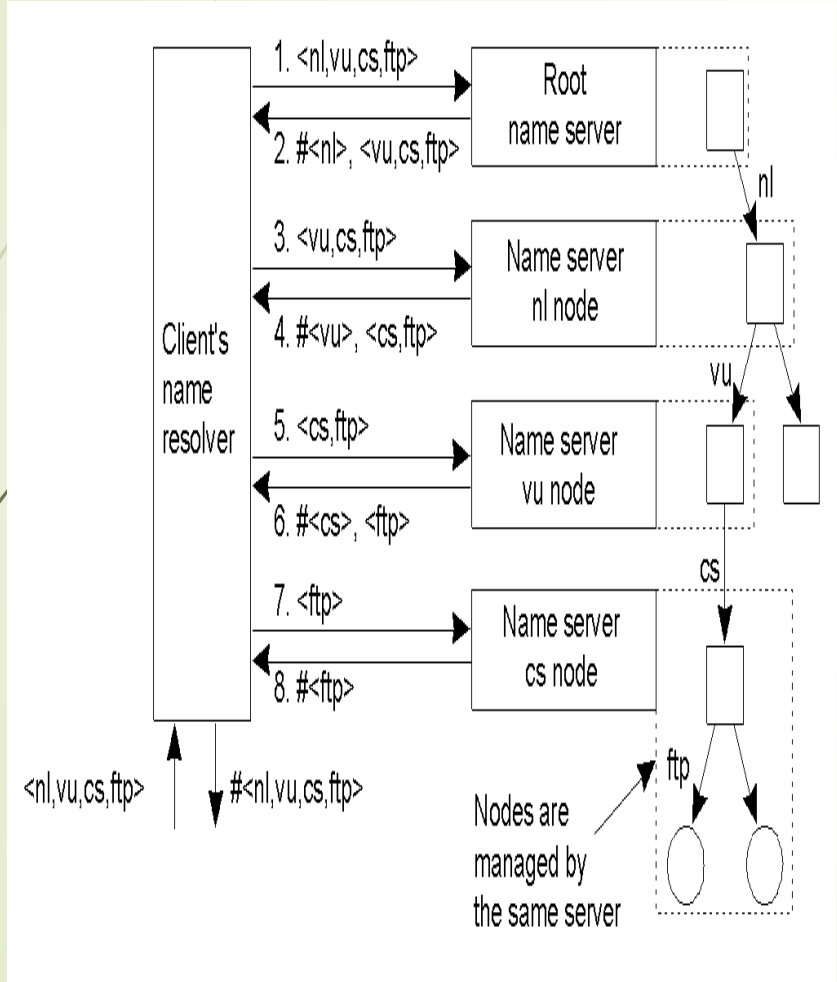
# Iterative Name Resolution



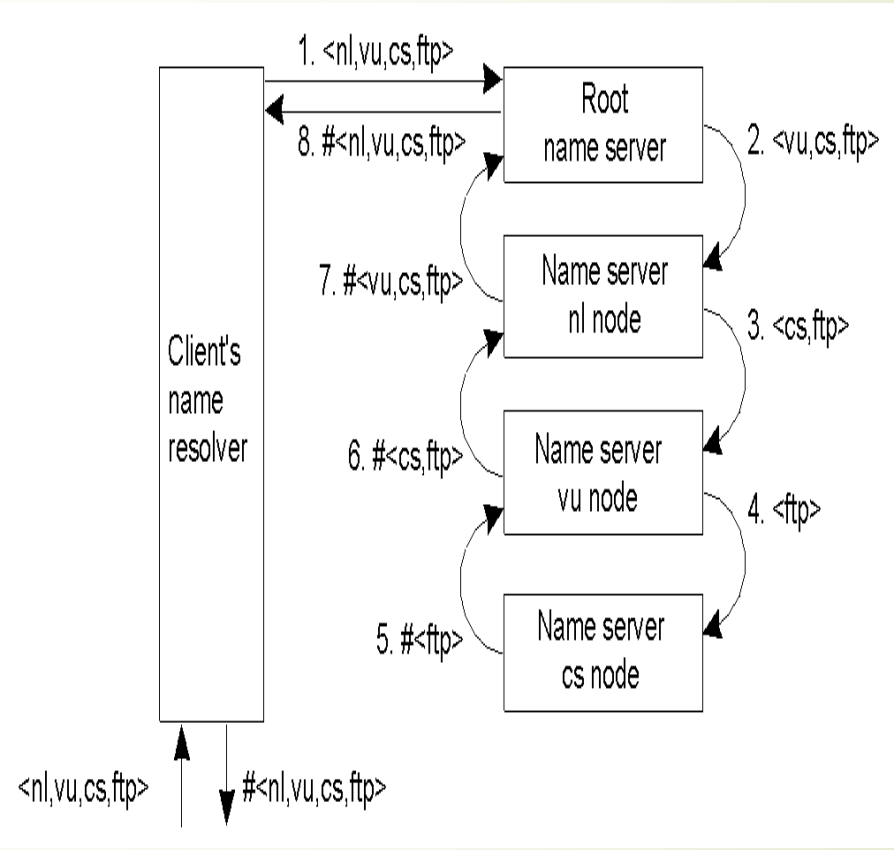
# Recursive Name Resolution (1)



# Iterative versus Recursive Resolution (1)




How about communication cost?




Performance-wise, which is better?

Which works better with caching?

# Iterative versus Recursive Resolution (2)

- Performance-wise, which is better?
  - Recursive method puts higher performance demand on each name server
- Which works better with caching?
  - Recursive method works better with caching
- How about communication cost?
  - Recursive method can reduce communication cost

# Questions??