INFO-H-509 XML AND WEB TECHNOLOGIES

Lecture 1: Introduction and Web Architecture

Stijn Vansummeren

February 14, 2018

LECTURE OUTLINE

General course information

Definition of the Web

Constitutent 1: Resources and URIs

Constitutent 2: HTTP

Constitutent 3: HTML and CSS as representation format

GENERAL COURSE INFORMATION

COURSE RESPONSIBLES

Stijn Vansummeren
Campus Solbosch, UB4.125
svsummer@ulb.ac.be

Dhananjay Ipparthi
Campus Solbosch, UB4.131
dhananjay.ipparthi@ulb.ac.be

Personal meeting before or after class **upon request**

PREREQUISITES

Required

- Basic programming skills

Recommended

- Basic knowledge about computer networks
- Introductory course on (relational) databases.

COURSE OBJECTIVES

Objective:

- Get acquainted with the principles, architectures, and systems for producing, exchanging, consuming, and reasoning with data on the World Wide Web.

Content divided into 4 parts:

1. Introduction and basic web architecture.
2. The syntactic web (XML and related standards, JSON)
3. The Semantic Web and Linked Open Data (RDF + ontologies)
4. Web Services (REST, BIG-WS*)

COMPETENCES TO DEVELOP

After successful completion of this course you should be able to

1. construct simple HTML pages and simple CSS stylesheets.
2. construct well-formed XML documents; describe families of XML documents using DTDs and XML Schema's; access, transform and query XML documents with XPath, XSLT and XQuery.
3. interpret and construct RDF data; query RDF with SPARQL.
4. construct RDF ontologies (in both RDF Schema and OWL); as well as predict the inferences that can be made from them.
5. use RDF in practice: linked data.
6. interact with (consume) RESTfull web services and to design new RESTfull web services.
7. interact with (consume) BIG-WS* web services.
8. understand and explain the differences between RESTfull and BIG-WS* web services. Motivate which of the two is applicable in a given application scenario.

ORGANIZATION

The course is organized as a mixture of:

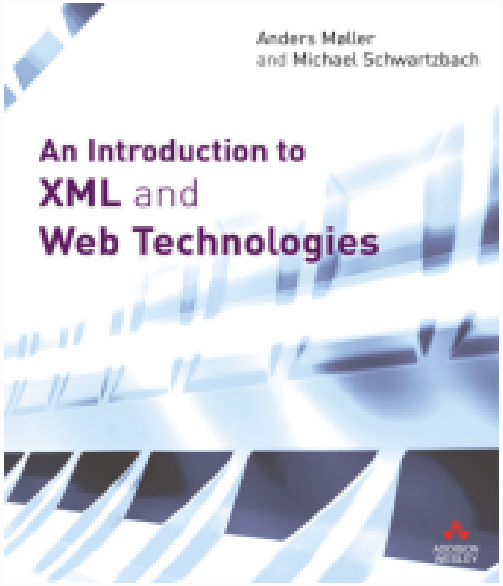
- Ex-cathedra lectures
- Self-study reading assignments
- Technical exercises
- Project work

Course material, exercises, reading assignments, project assignments are all published on the course website:

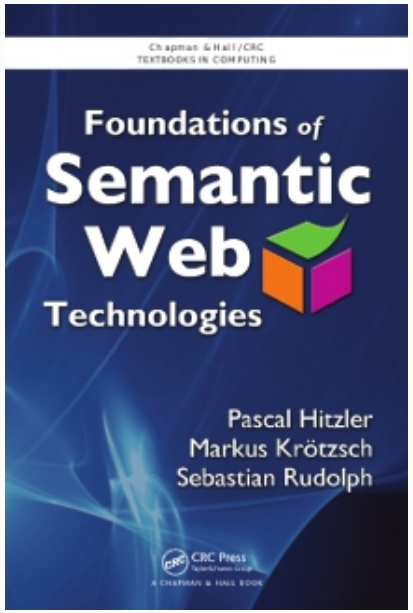
`http://cs.ulb.ac.be/public/teaching/infoh509`

Check regularly for updates!

SYLLABUS



A. Moller and M. Schwartzbach
Addison-Wesley, 2006
ISBN-13: 978-0-321-26966-9
Required.



P. Hitzler, M. Krötzsch, and S. Rudolph
CRC Press, 2009
ISBN-13: 978-1-420-09050-5
Optional.

NOTE: The course slides are supportive material only and do not cover everything that will be examinated! Check webpage for required reading assignments.

EVALUATION / EXAM

Written exam:

- contributes 14/20 (70%) to final score;
- questions in English;
- you are free to respond in Dutch, English, or French;

Project work:

- contributes 6/20 (30%) to final score;
- consists of 3 seperate mini-projects (2/20 each);
- to be done individually
- project assignments and deadlines become available on the course website once corresponding material has been studied.

DEFINITION OF THE WEB

WHAT IS THE WORLD WIDE WEB?

- **1989**: At the CERN physics laboratory, Tim Berners-Lee designs a simple global hypermedia system, know known as the World Wide web

Three constituents (with their **original meaning**):

- URIs as a means for locating files on servers
- HTTP as a protocol for transmitting HTML files over networks
- HTML as a markup language for describing information in hypertext form

WHAT IS THE WORLD WIDE WEB?

- **1989**: At the CERN physics laboratory, Tim Berners-Lee designs a simple global hypermedia system, know known as the World Wide web

Three constituents (with their current meaning):

- URIs as a means for identifying & locating resources
- HTTP as a protocol for transmitting information over networks
- A myriad of data formats for describing information (resource representations):
  - HTML for display in a browser
  - XML as a data exchange format
  - JSON as a data exchange format, alternative to XML
  - RDF as a machine-interpretable data model

SO: THE WEB ≠ THE INTERNET

- The internet is a global system of interconnected computer networks.

- The Web is a subset of the Internet. It is a collection of resources, linked by hyperlinks and URIs, transmitted in various formats by web clients (e.g., browsers) and web servers talking HTTP.

CONSTITUTENT 1: RESOURCES AND URIS

The W3C's view on the Web

"

The World Wide Web (WWW, or simply Web) is an information space in which the items of interest, referred to as **resources**, are identified by global identifiers called **Uniform Resource Identifiers (URI)**.

…

In order to communicate internally, a community agrees (to a reasonable extent) on a set of terms and their meanings. One goal of the Web, since its inception, has been to build a global community in which any party can share information with any other party. To achieve this goal, the Web makes use of a single global identification system: the URI.

--Architecture of the World Wide Web, Volume One
(http://www.w3.org/TR/2004/REC-webarch-20041215/)

URI = Uniform Resource Identifier

"

A Uniform Resource Identifier (URI) provides a simple and extensible means for **identifying** a **resource**.

--RFC 3986

- The purpose & syntax of URIs has been defined in a number of standards, evolving over the years.
  - 1994: "Universal Resource Identifiers in WWW" - RFC 1630
  - 1998: "Uniform Resource Identifiers (URI): Generic Syntax" – RFC 2396
  - 2005: "Uniform Resource Identifier (URI): Generic Syntax" – RFC 3986

Current version, see http://tools.ietf.org/html/rfc3986

URI = Uniform Resource Identifier

“

A Uniform Resource Identifier (URI) provides a simple and extensible means for **identifying** a **resource**.

Originally:

A statically addressable document or file on the network

- Examples:
  - Example: the logo.jpg file on the google.com server

URI = Uniform Resource Identifier

"

A Uniform Resource Identifier (URI) provides a simple and extensible means for **identifying** a **resource**.

Currently:

***Any entity,* both physical or abstract, both network-accessible or not**

- Examples from RFC 3986:
  - Electronic documents and images, as before
  - Sources of information (e.g., "today's weather report for Los Angeles")
  - Services (e.g., an HTTP-to-SMS gateway)
  - Human beings, animals, corporations, and bound books in a library
  - Abstract concepts, such as mathematical operators, the types of a relationship (e.g., "parent" or "employee"), or numeric values (e.g., zero, one, and infinity).

URI = Uniform Resource Identifier

"

A Uniform Resource Identifier (URI) provides a simple and extensible means for **identifying** a **resource**.

Currently:

***Any entity,* both physical or abstract, both network-accessible or not**

Examples of **Digital Resources**

- Examples from RFC 3986:
  - An HTML document, PDF document, JPEG image (as before)
  - Sources of information (e.g., "today's weather report for Los Angeles")
  - Services (e.g., an HTTP-to-SMS gateway)
  - Human beings, animals, corporations, and bound books in a library
  - Abstract concepts, such as mathematical operators, the types of a relationship (e.g., "parent" or "employee"), or numeric values (e.g., zero, one, and infinity).

URI = Uniform Resource Identifier

"

A Uniform Resource Identifier (URI) provides a simple and extensible means for **identifying** a **resource**.

Currently:

***Any entity,* both physical or abstract, both network-accessible or not**

Examples of **Non-digital Resources**

- Examples from RFC 3986:
  - An HTML document, PDF document, JPEG image (as before)
  - Sources of information (e.g., "today's weather report for Los Angeles")
  - Services (e.g., an HTTP-to-SMS gateway)
  - Human beings, animals, corporations, and bound books in a library
  - Abstract concepts, such as mathematical operators, the types of a relationship (e.g., "parent" or "employee"), or numeric values (e.g., zero, one, and infinity).

URI = Uniform Resource Identifier

"

A Uniform Resource Identifier (URI) provides a simple and extensible means for **identifying** a **resource**.

"

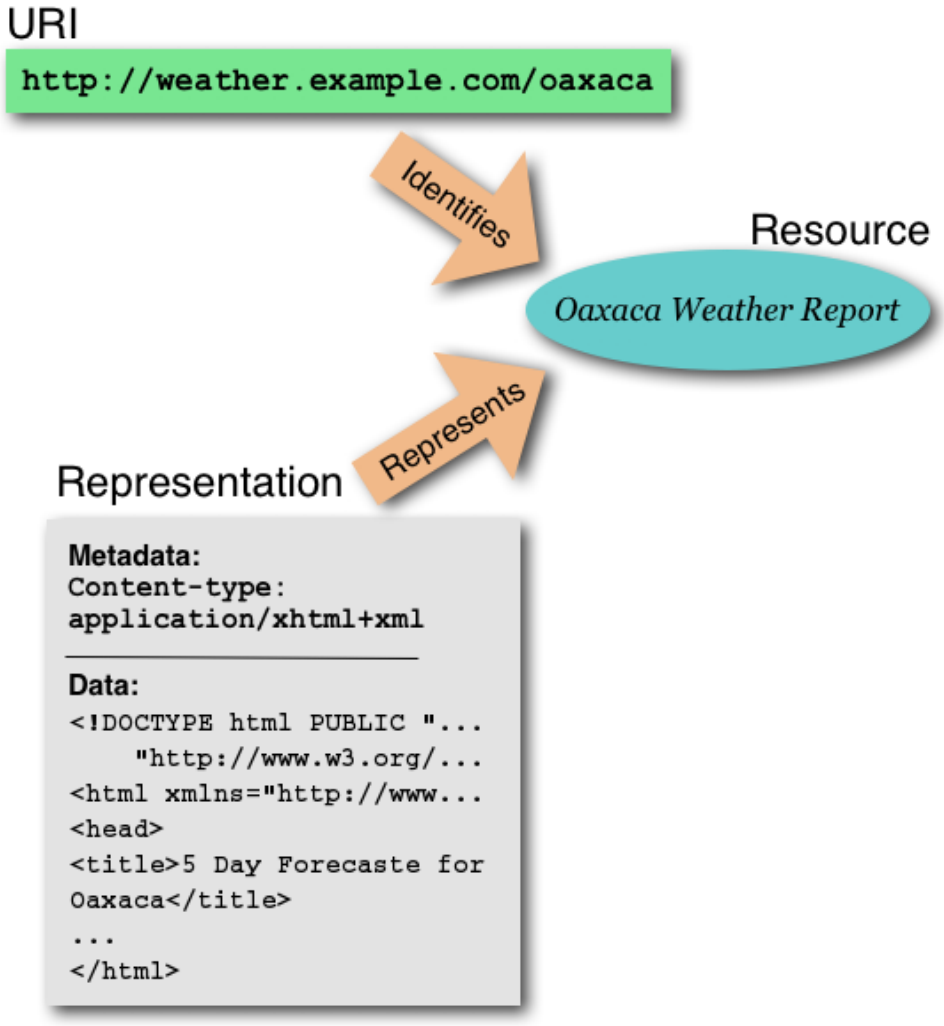
Identifier

Our use of the terms "identify" and "identifying" refer to the purpose of distinguishing one resource from all other resources, regardless of how that purpose is accomplished (e.g., by name, address, or context).

…

It should not be assumed that a system using URIs will access the resource identified: in many cases, URIs are used to denote resources without any intention that they be accessed.

The W3C's view on the Web



URIs

- A URI takes the form

scheme:scheme-specific-part

- Examples:
  - ftp://ftp.is.co.za/rfc/rfc1808.txt
  - http://www.ietf.org/rfc/rfc2396.txt
  - ldap://[2001:db8::7]/c=GB?objectClass?one
  - mailto:John.Doe@example.com
  - news:comp.infosystems.www.servers.unix
  - tel:+1-816-555-1212
  - telnet://192.0.2.16:80/
  - urn:oasis:names:specification:docbook:dtd:xml:4.1.2
  - doi:10.1000/182

Scheme:
ftp
http
ldap
mailto
news
tel
telnet
urn
doi

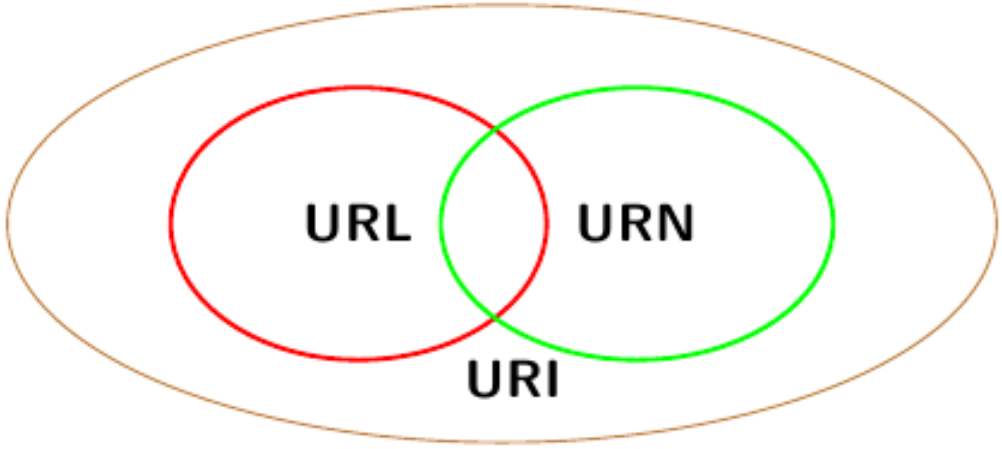
URIs

- A URI takes the form

scheme:scheme-specific-part

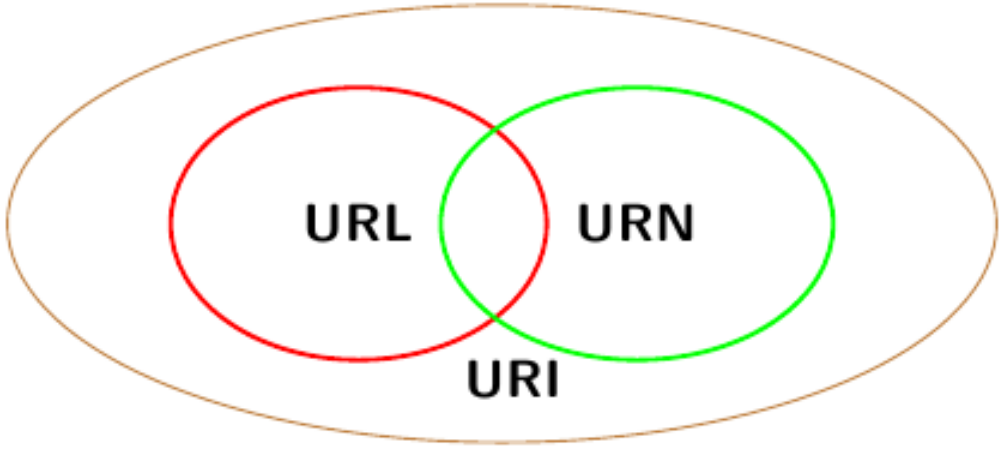
- Each scheme is free to define the syntax of the scheme-specific part. This makes URIs extensible
- There are conventions about the use of /, #, and ?

A note on URIs



- A Uniform Resource Identifier (URI) is either:
  - a Uniform Resource Locator (URL)
  - a Uniform Resource Name (URN),
  - Or both

A note on URIs



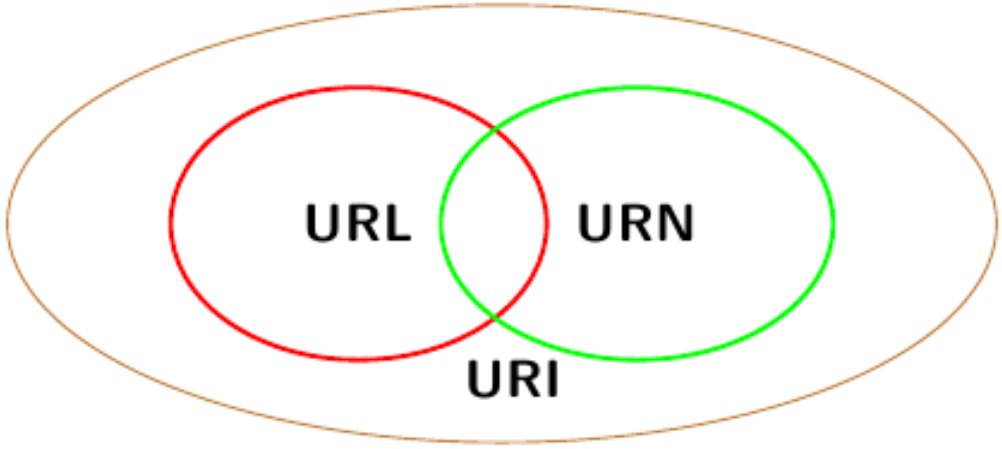
- A Uniform Resource Name functions like a person's name

urn:isbn:0-486-27557-4

That is: it gives a globally unique and persistent identifier, but not the item's location.

(Historically speaking, URNs used to begin with the urn scheme, but that is no longer required.)

A note on URIs

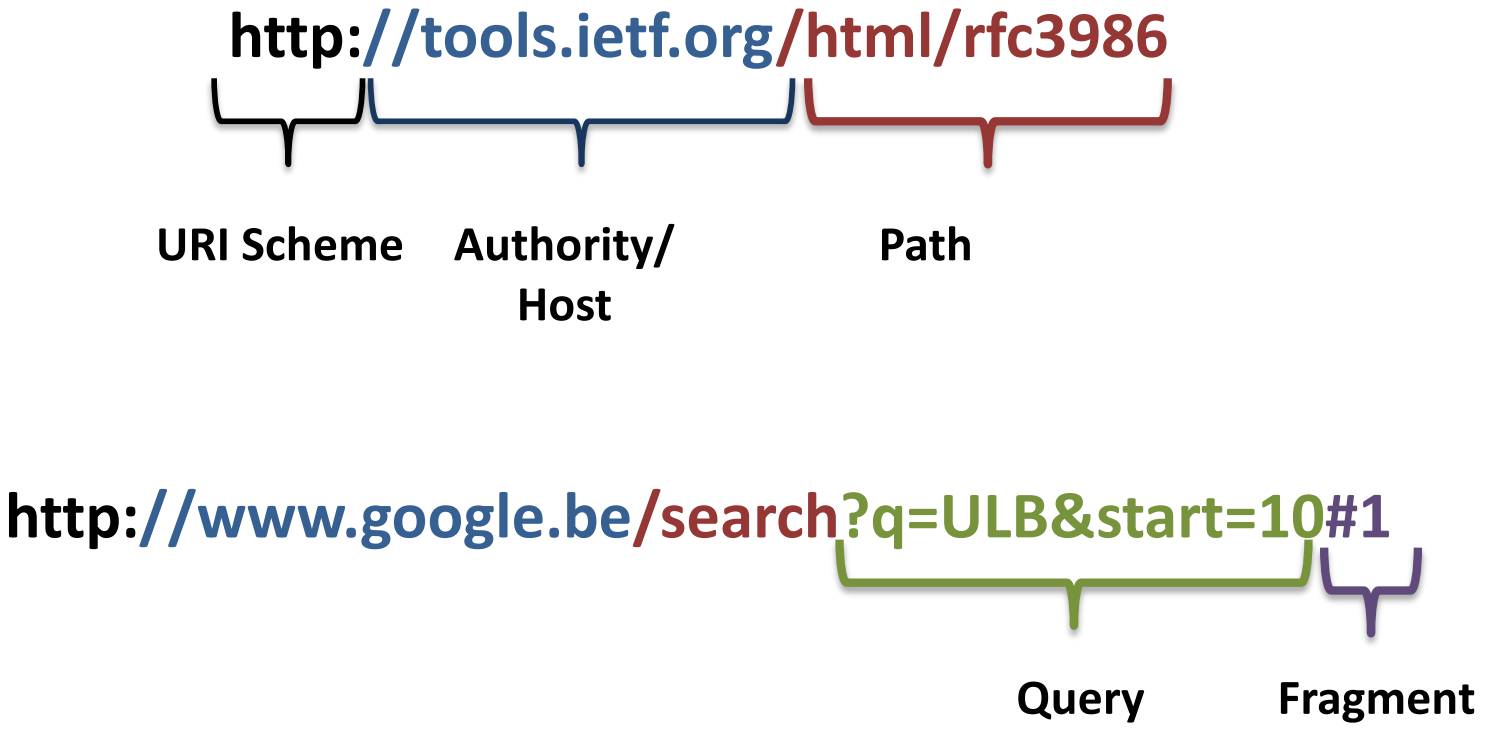


- A Uniform Resource Locator functions like a street address:

  http://www.google.com

  That is: in addition to identifying an item, it provides a means of locating the resource (e.g., its network "location"), although it need not be persistent

The anatomy of a URL



URIs are identifiers

- By design a URI identifies one (and only one) resource.
  - Ex. mailto:stijn.vansummeren@ulb.ac.be identifies my mailbox (it does **not** identify me!)

- Yet, the same resource may be identified by multiple URIs (these are called **URI aliases**)
  - Ex. mailto:svsummer@ulb.ac.be also identifies my mailbox.

URIs: scope

- URIs have global scope

  – When I write

    http://www.google.com

    or

    urn:isbn:0-486-27557-4

    it means the same thing as when someone else (any where in the world) does this.

URIs: ownership & allocation

"

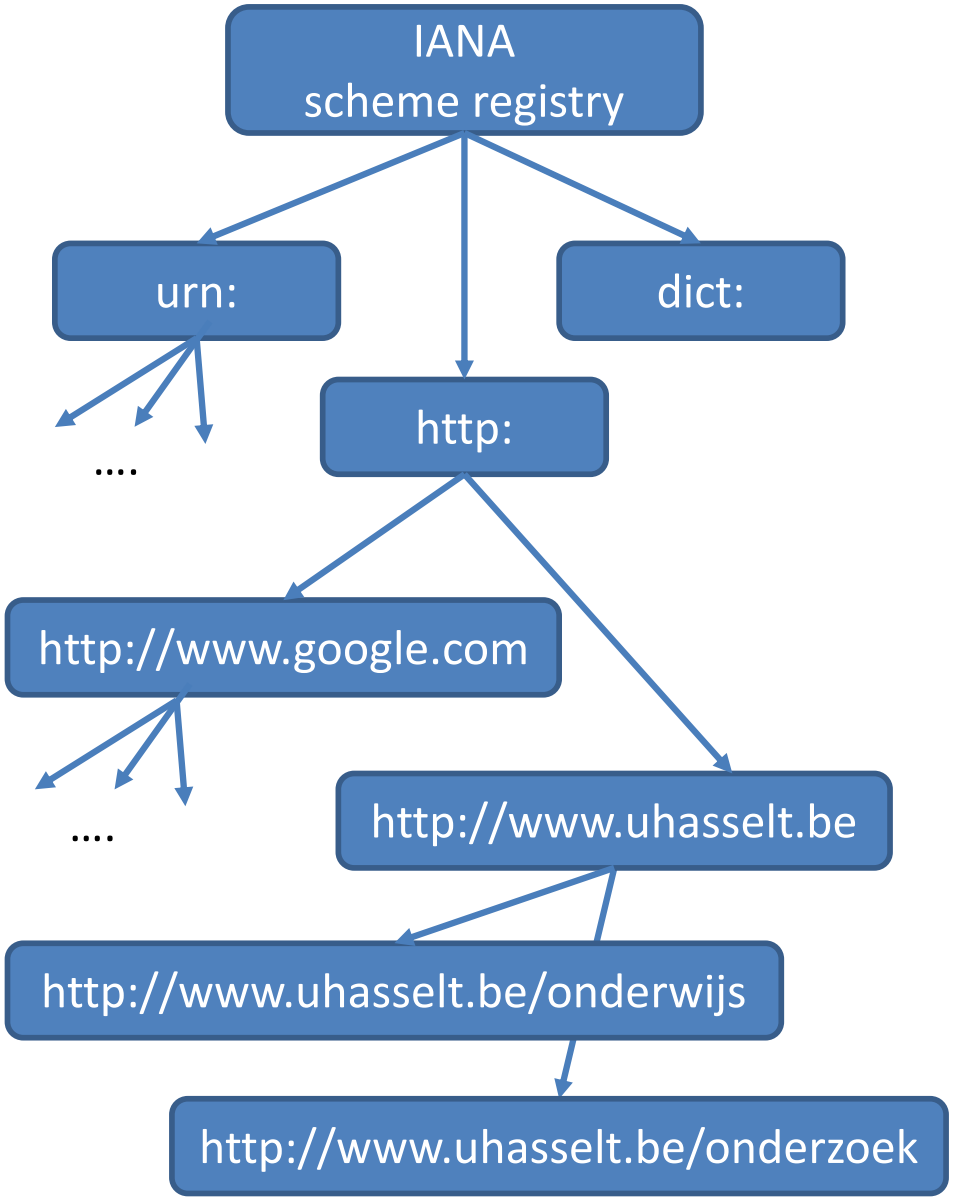
URI ownership is a relation between a URI and a social entity, such as a person, organization, or specification. URI ownership gives the relevant social entity certain rights, including:

1. to pass on ownership of some or all owned URIs to another owner—delegation; and
2. to associate a resource with an owned **URI—URI allocation**.

--Architecture of the World Wide Web, Volume One (http://www.w3.org/TR/2004/REC-webarch-20041215/)

- **CONSEQUENCE:** You should only allocate ("mint") URIs that you own.

URI ownership is hierarchical



- The root of all URI ownership lies with the IANA scheme registry, itself a social entity (http://www.iana.org/assignments/uri-schemes),
- It delegates, for each registered scheme, ownership to some scheme-specific owner.
- Each scheme-specific owner is free to delegate further in a way that it sees fit. (Or not delegate at all)
- For example, HTTP delegates to DNS: only if you own the host should you allocate HTTP URIs starting with that host
- Hosts can delegate further

URIs: ownership & allocation

"

URI ownership is a relation between a URI and a social entity, such as a person, organization, or specification. URI ownership gives the relevant social entity certain rights, including:

1. to pass on ownership of some or all owned URIs to another owner—delegation; and
2. to associate a resource with an owned **URI—URI allocation**.

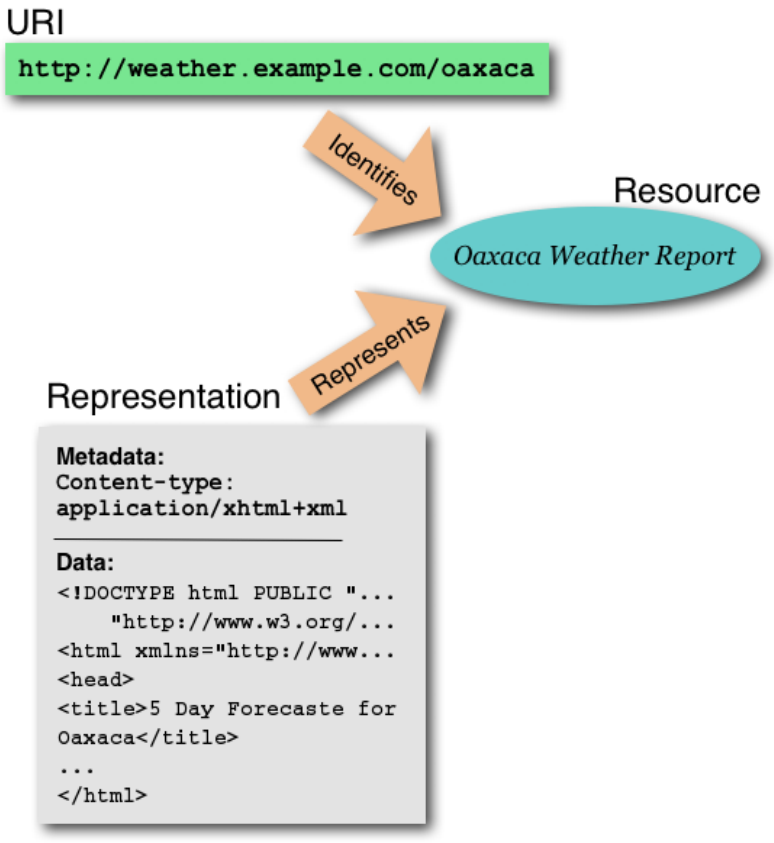
--Architecture of the World Wide Web, Volume One (http://www.w3.org/TR/2004/REC-webarch-20041215/)

- **SO:**
  - **You are free to use & re-use (i.e., link to) existing URIs**
  - **But only create URIs that you own!**

IRIs: internationalized URIs

- URIs, by design are only allowed to contain US ASCII characters (A-Z), not international characters like é, à, ç, ã etc.
- IRIs (or international resource identifiers) are identifiers in which such characters can occur.
- RFC 3987 defines how IRIs can be transformed “behind the scenes” to normal URIs.

The W3C's view on the Web (cont.)



"

A representation is data that encodes information about resource state. Representations do not necessarily describe the resource, or portray a likeness of the resource, or represent the resource in other senses of the word "represent".

--Architecture of the World Wide Web, Volume One (http://www.w3.org/TR/2004/REC-webarch-20041215/)

- Representations may be transmitted through various protocols.
- Most schemes (except URNs) come with default protocol.
- On the web, the predominant protocol is HTTP.

CONSTITUTENT 2: HTTP

HTTP: Hypertext Transfer Protocol

- A protocol that runs on top of the Internet's lower-lying protocols to provide **end-to-end** communication.

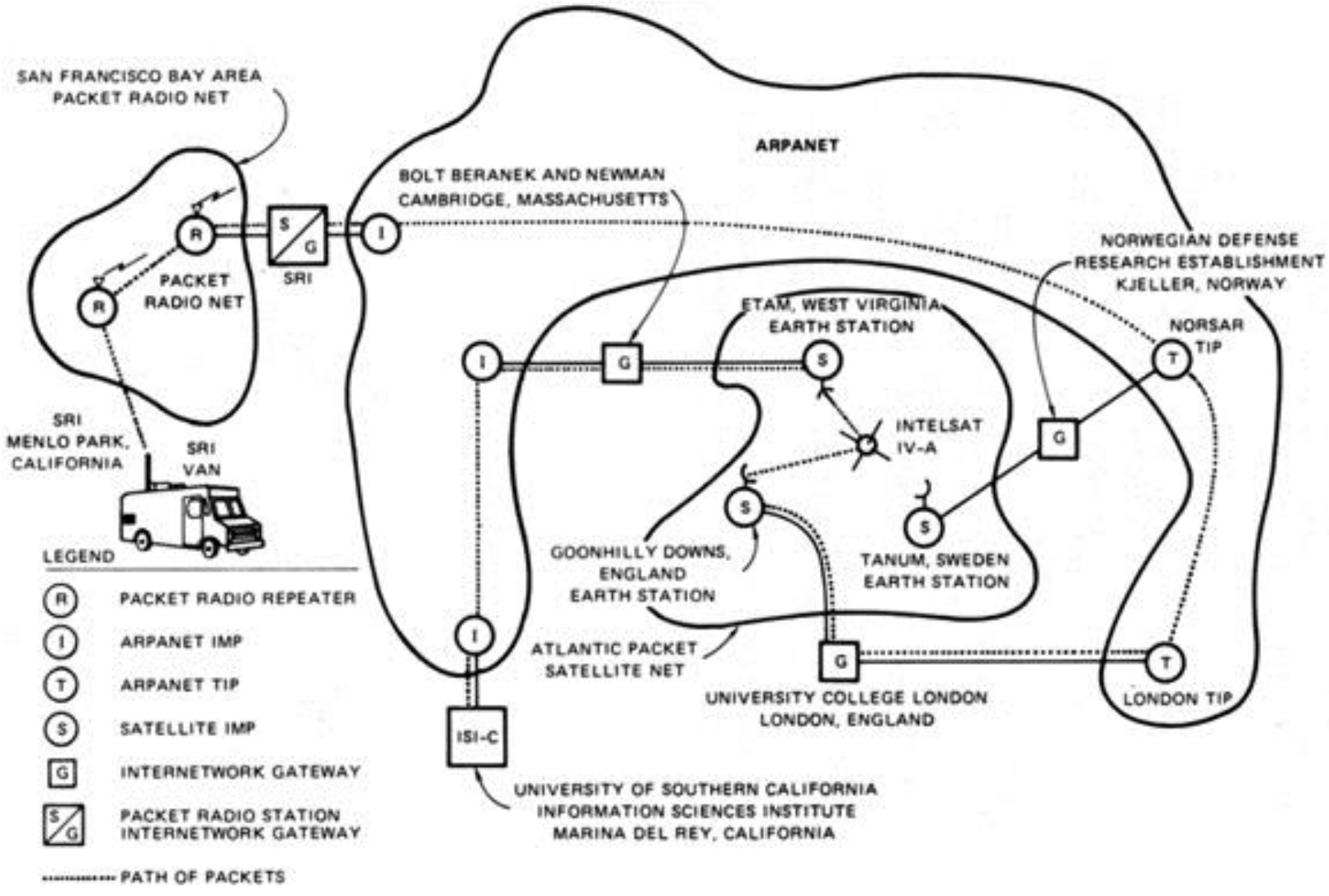
- Let's see what this means …

The Internet: some history

- **ARPANET (1969):**
  Advanced **R**esearch **P**roject **A**gency **NET**work
  - Created by the US Department of Defense
  - First operational **packet switching** network
  - Early ARPANET applications
    - Email, SMTP (1971), Ray Tomlinson
    - File Transfer Protocol, FTP (1973)

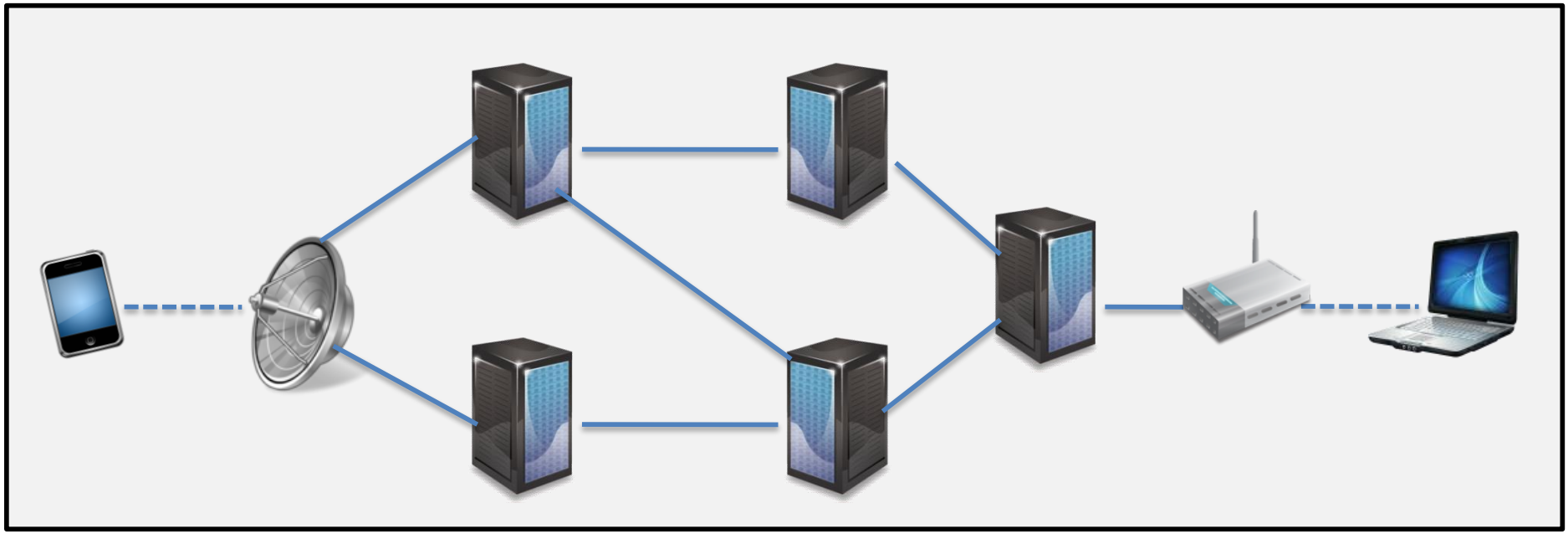
- Standardized **Internet Protocol Suite** (**TCP/IP**, 1983)

First ARPANET Logical Map



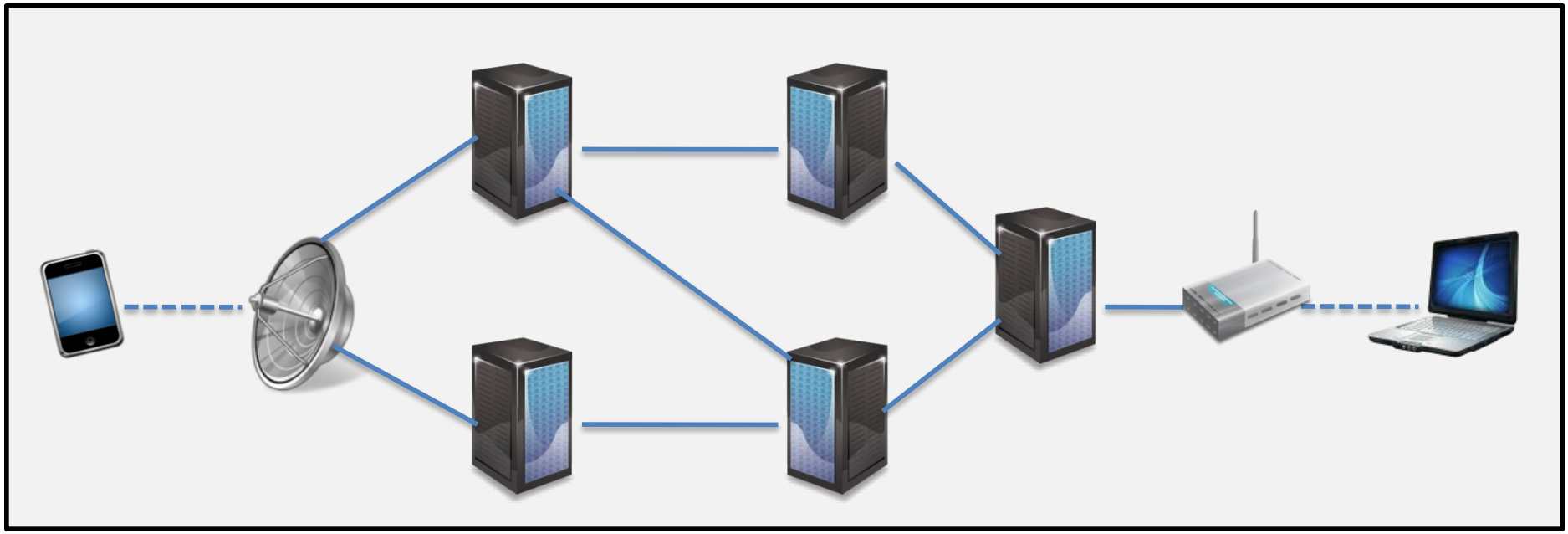
The Internet: simplified operation

- The Internet is a global system of interconnected computer networks.
- Computers that have a “physical link” (ethernet wire, WIFI, satelite, …) can talk directly to each other.
- Messages between non-adjacent computers are routed through adjacent computers to the destination.



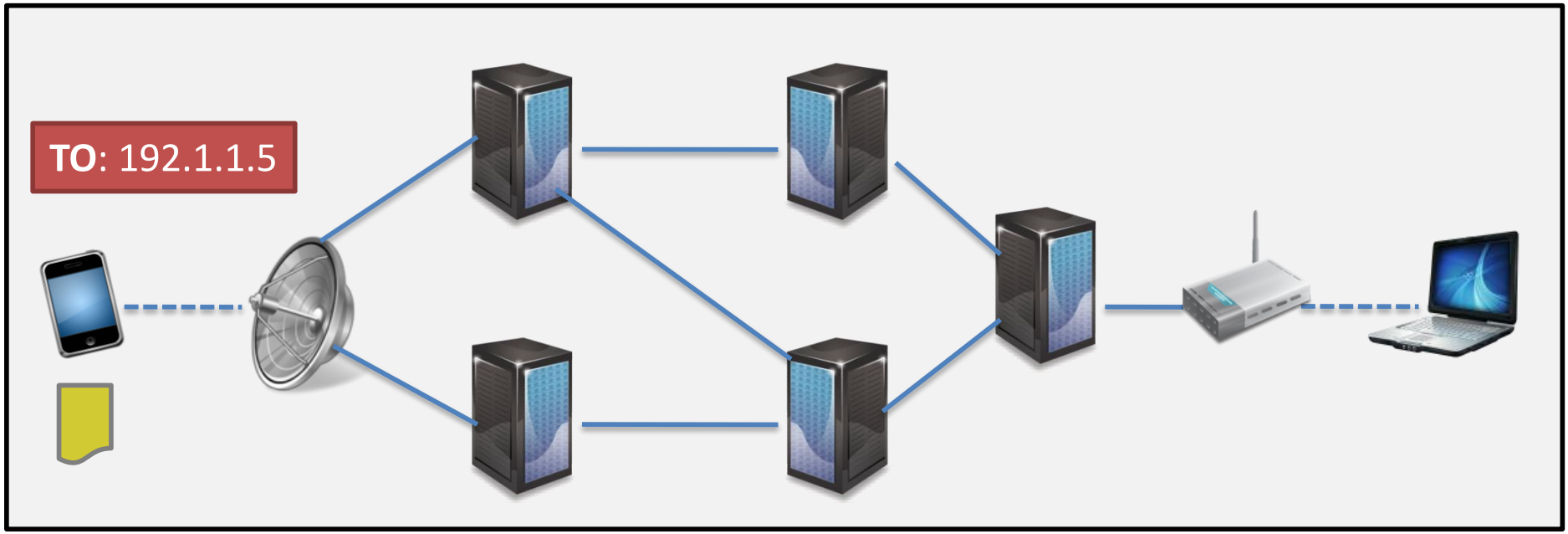
IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.



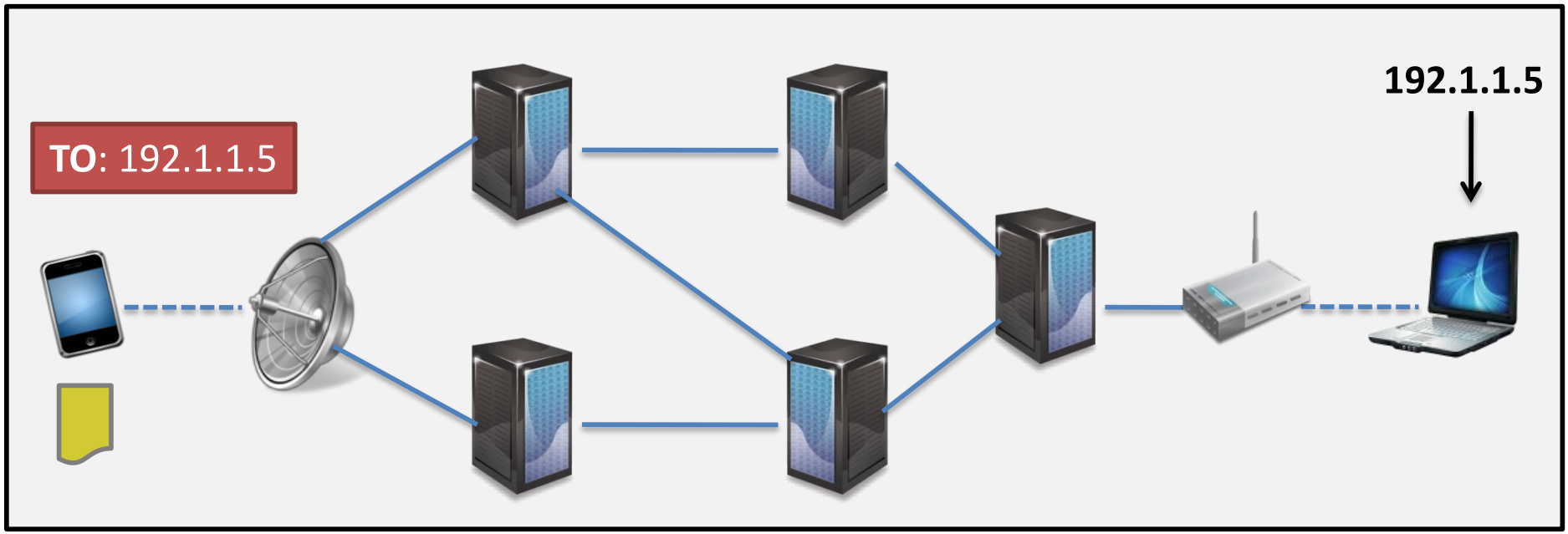
IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.



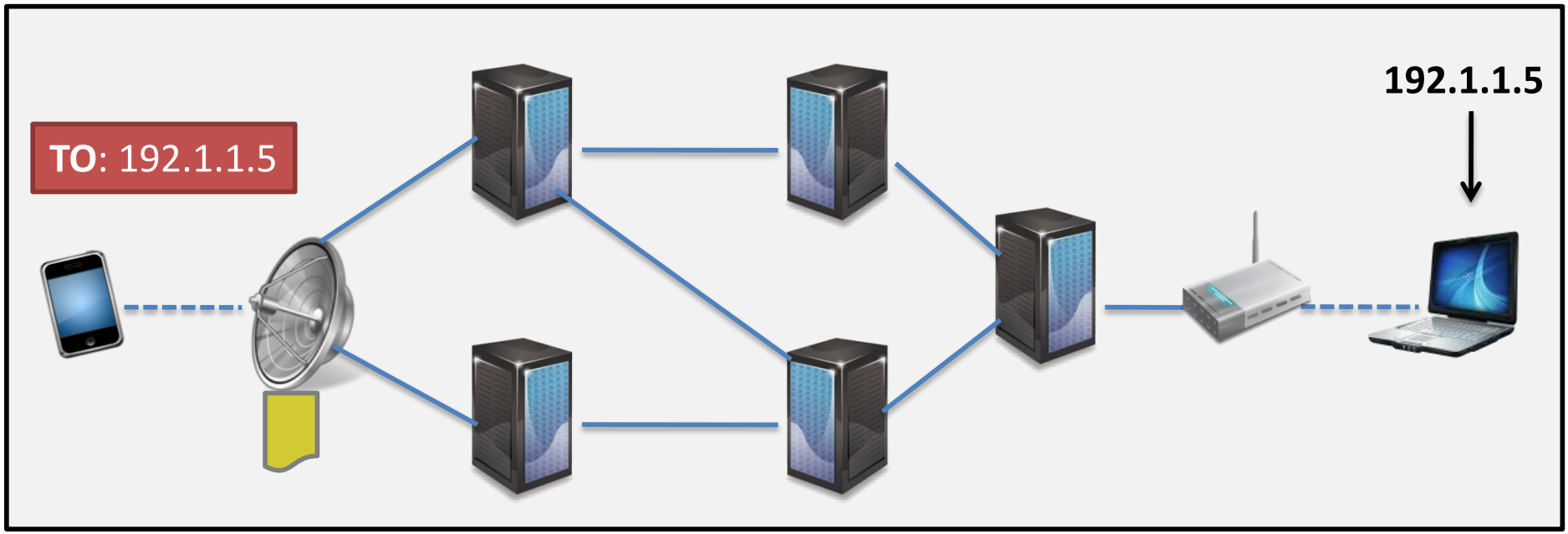
IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.



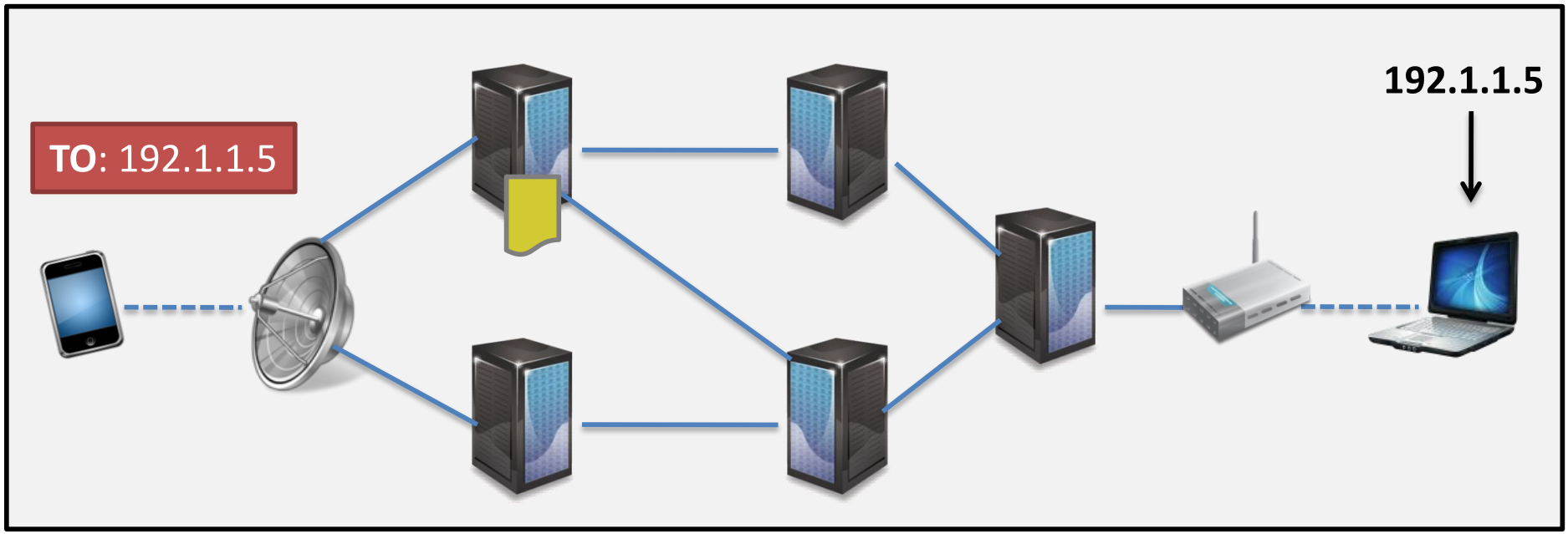
IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.



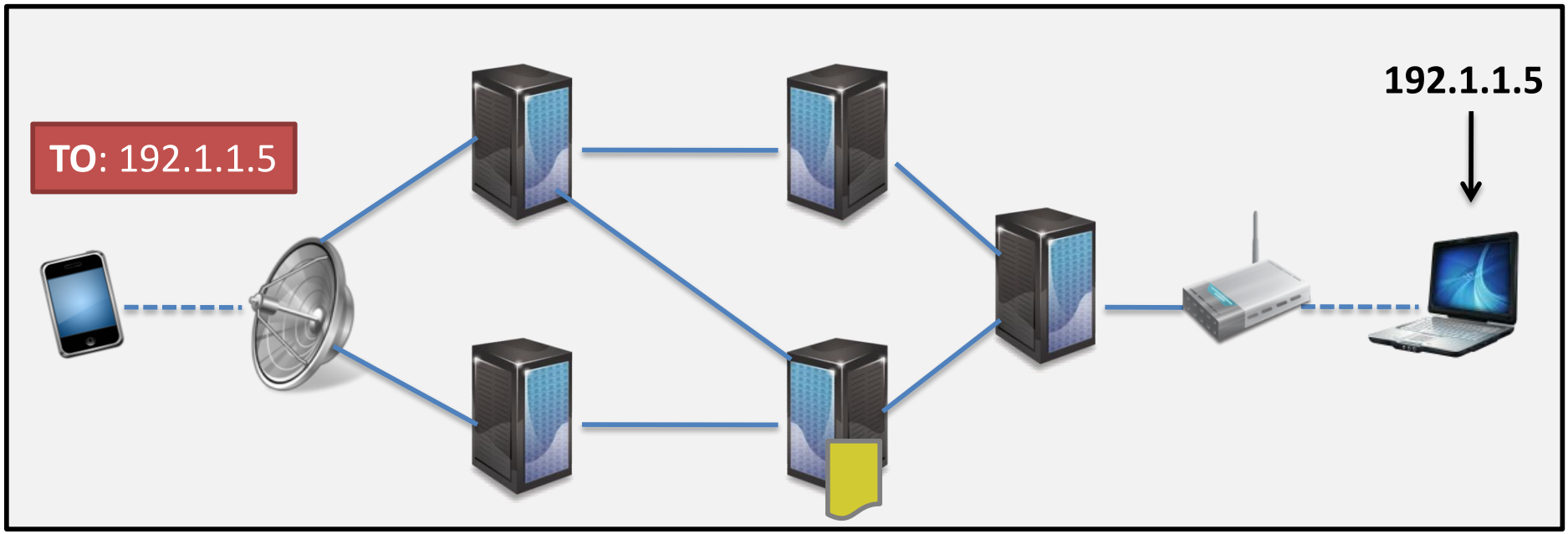
IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.



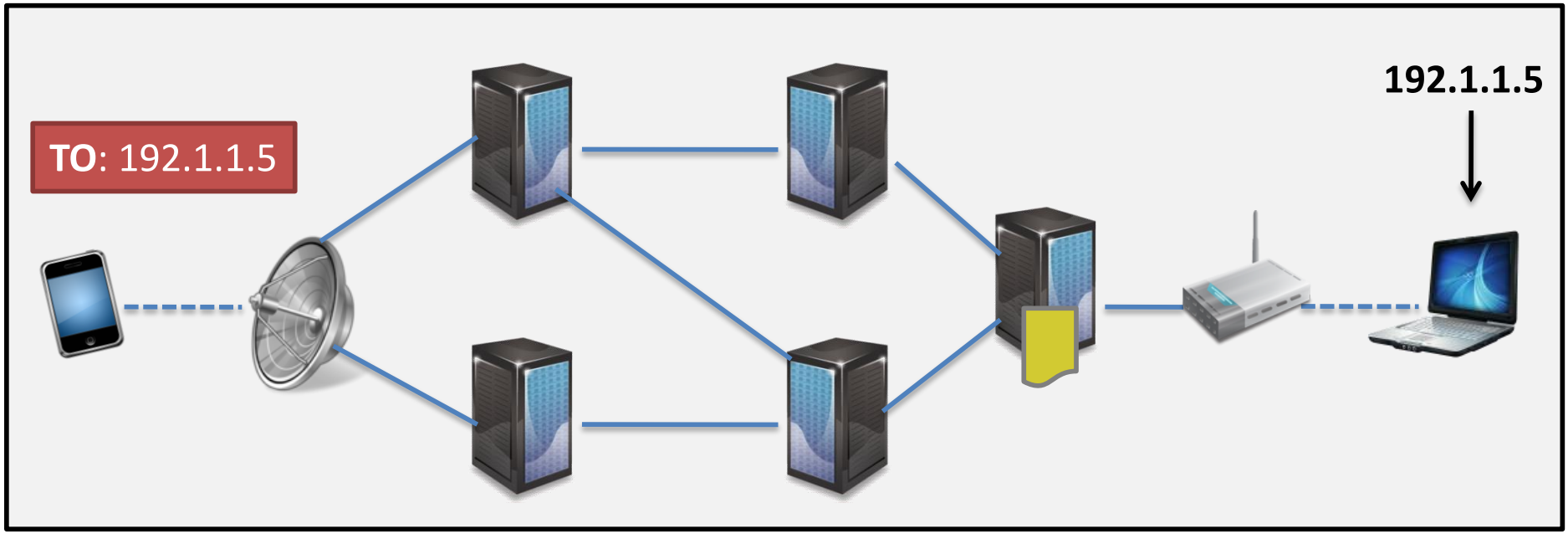
IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.



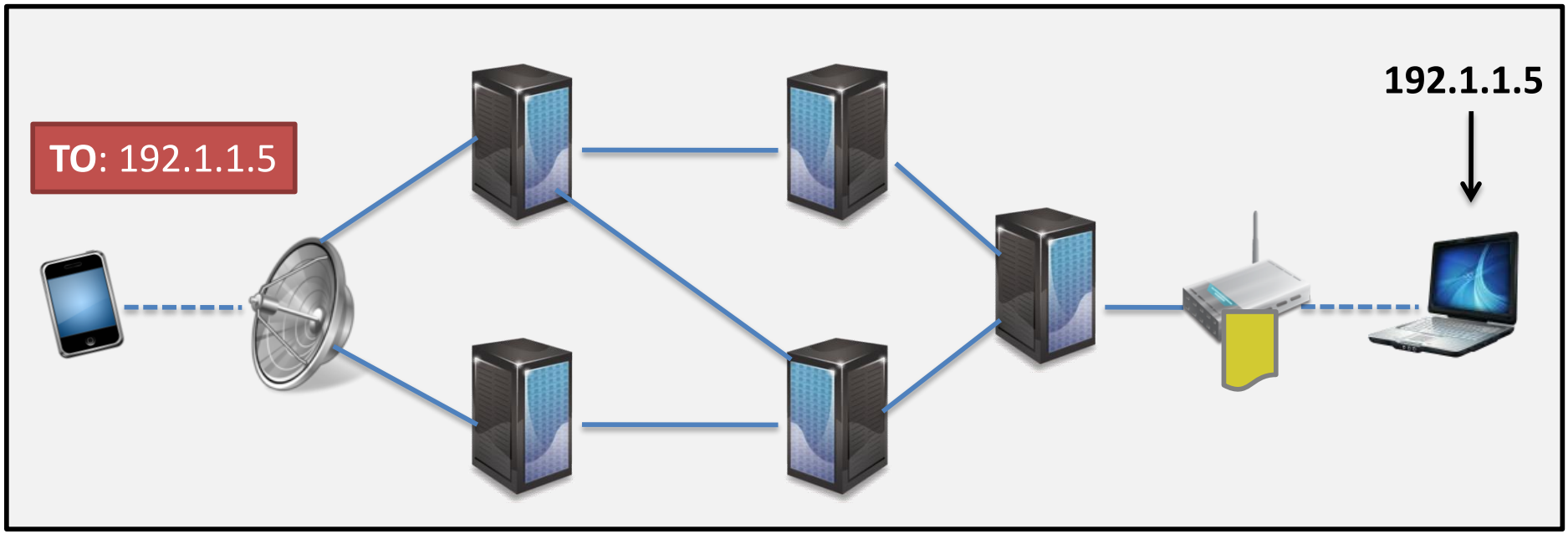
IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.



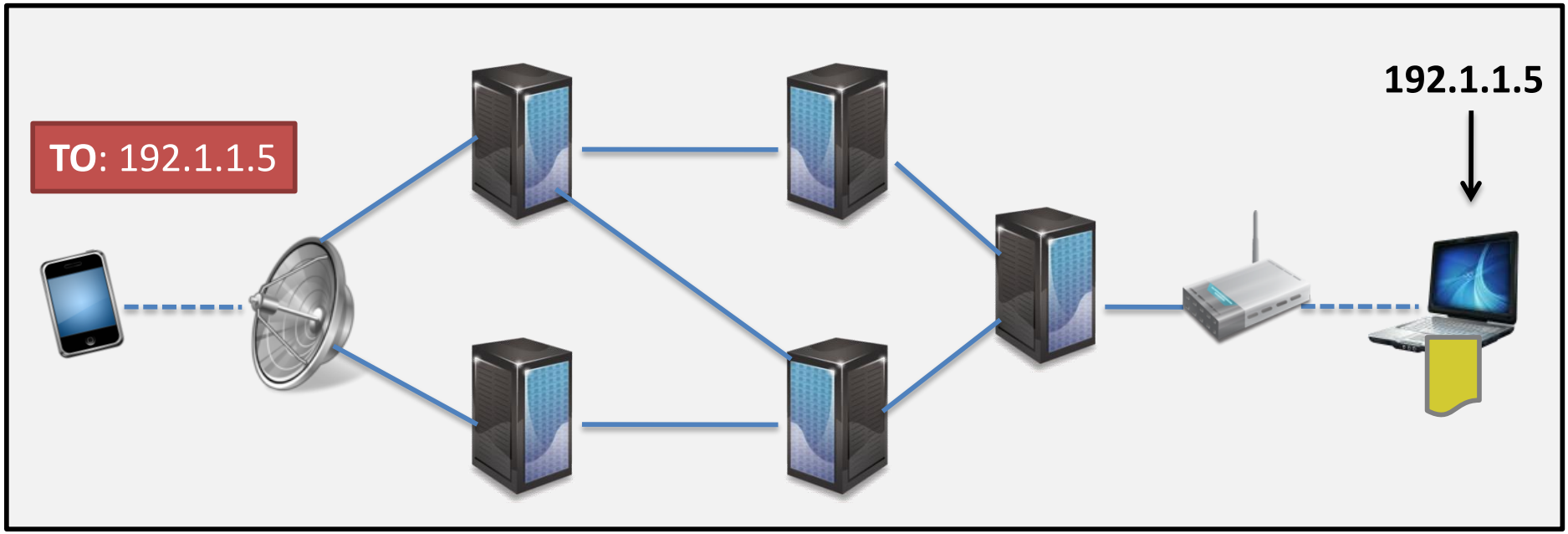
IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.

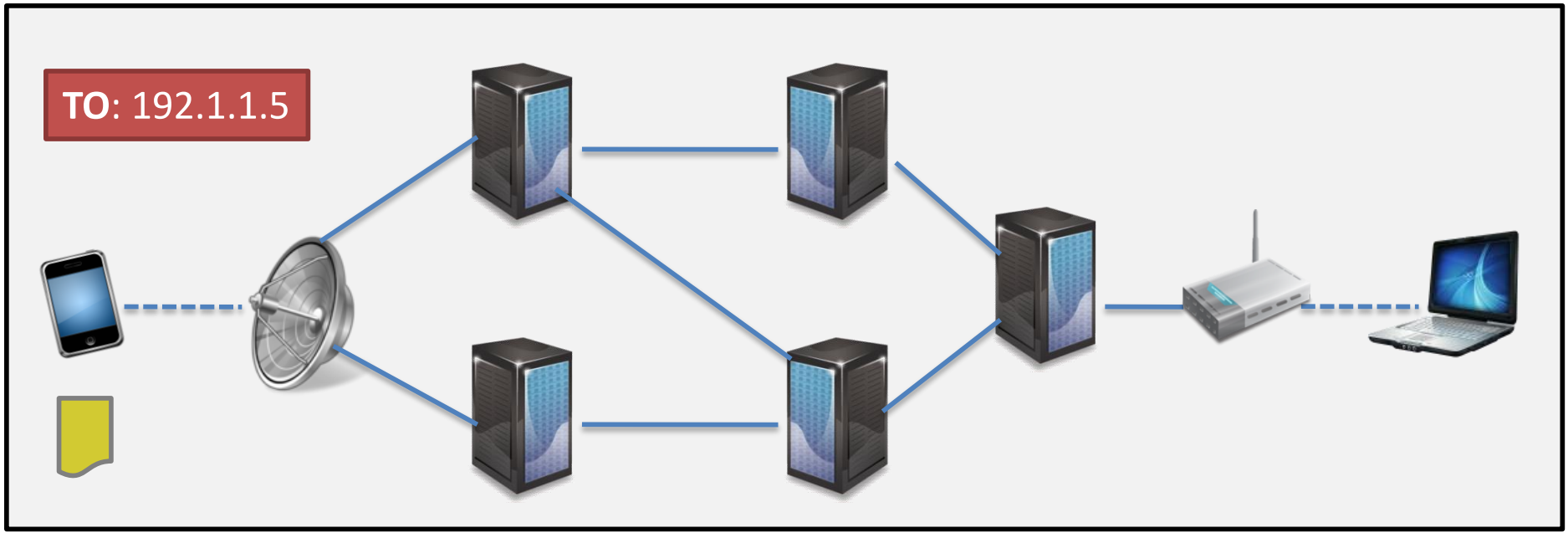


IP: Internet Protocol

- IP allows unreliable communication of *limited size data packets (called datagrams)* between machines identified by *IP addresses* (e.g., 164.15.59.215)
- Each datagram is sent and routed independently (and may arrive out of order). Routing is done transparently by the protocol.

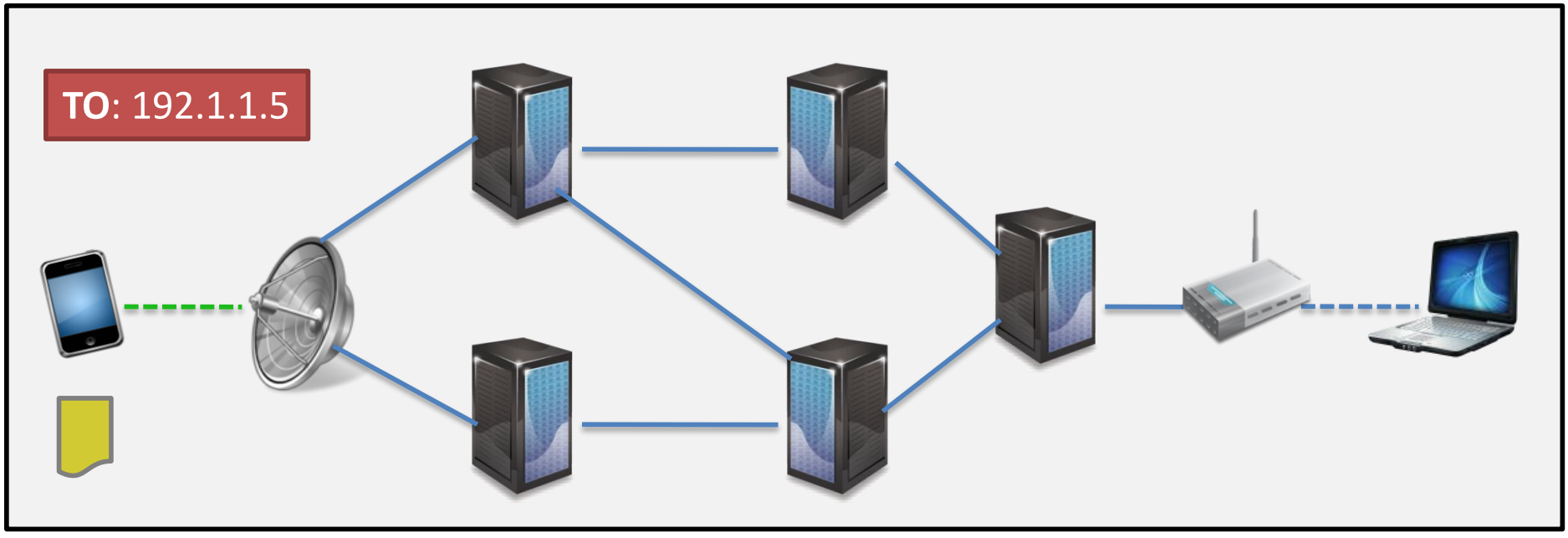


IP: Internet Protocol [Dataflow]



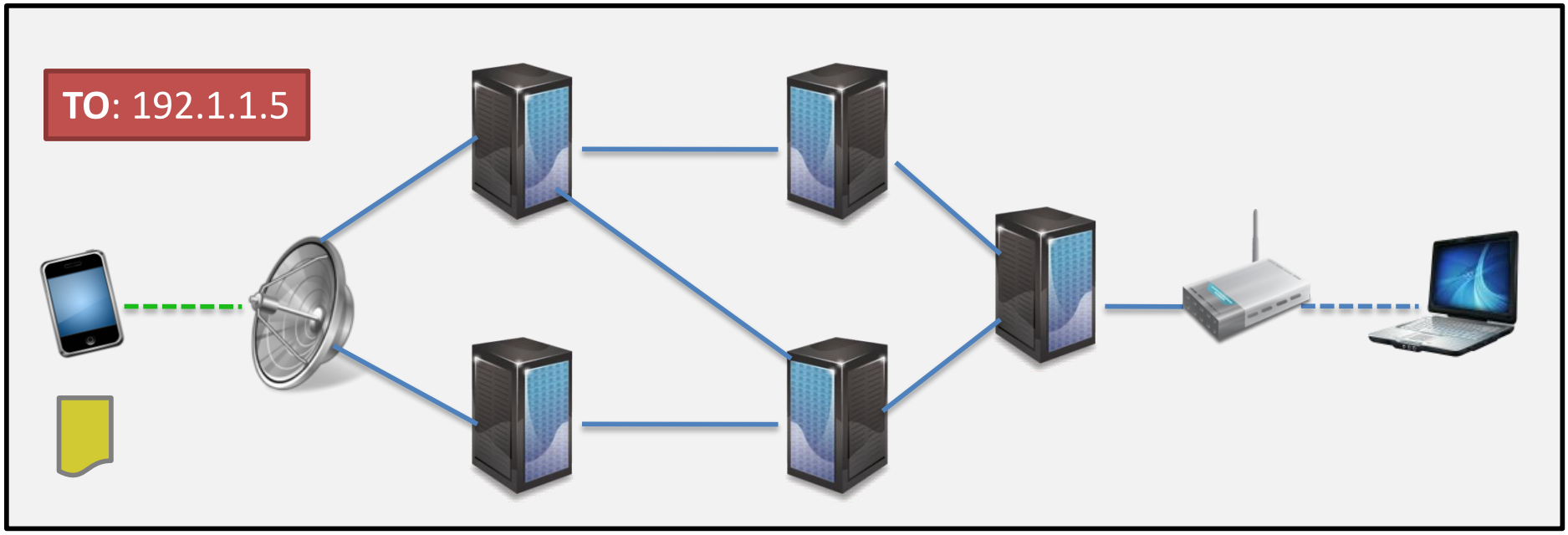
INTERNET LAYER

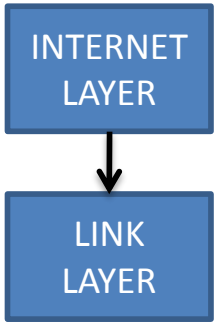
IP: Internet Protocol [Dataflow]



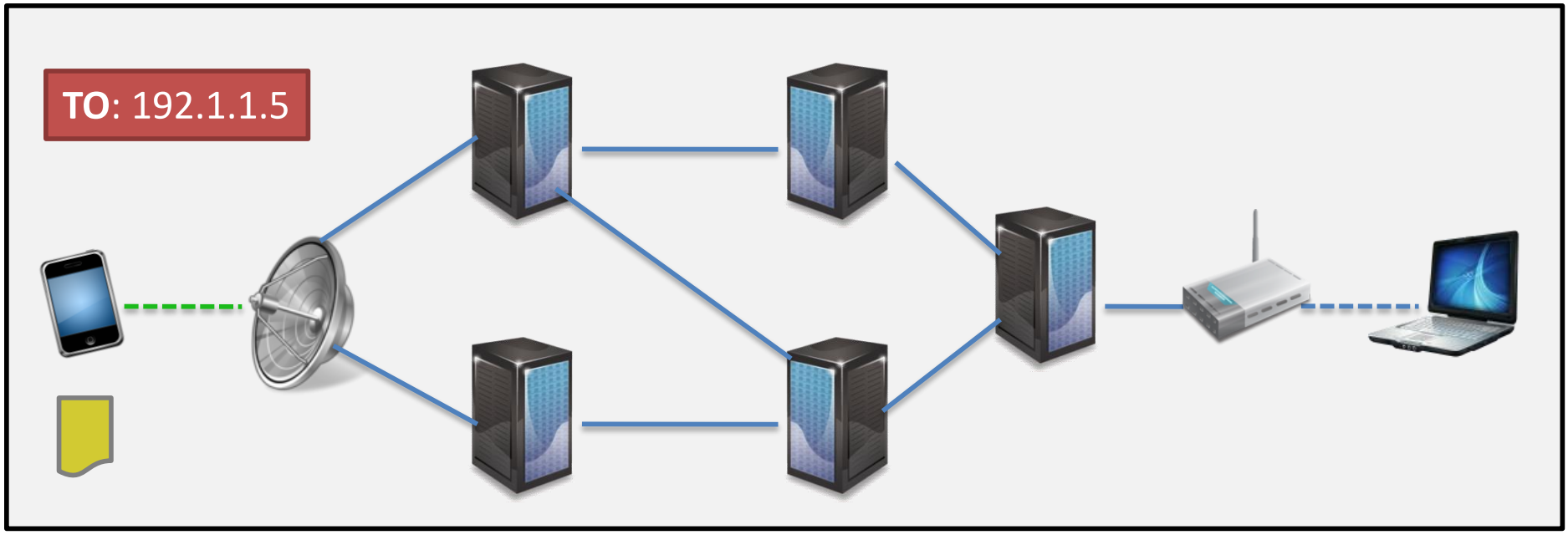
INTERNET LAYER

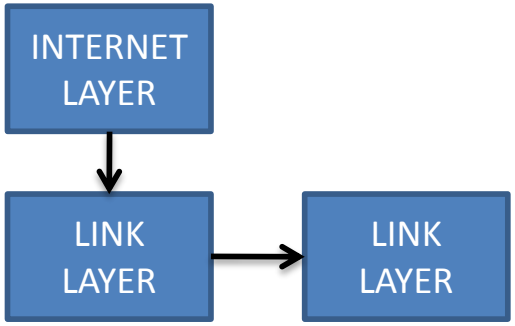
IP: Internet Protocol [Dataflow]



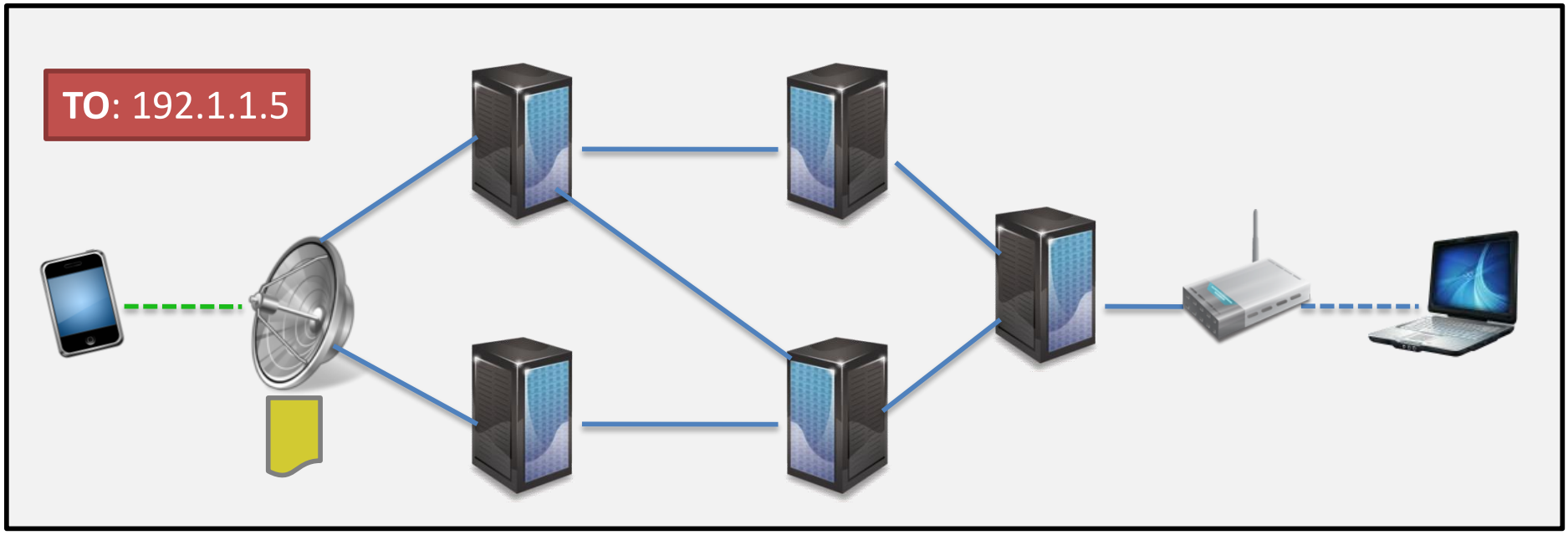


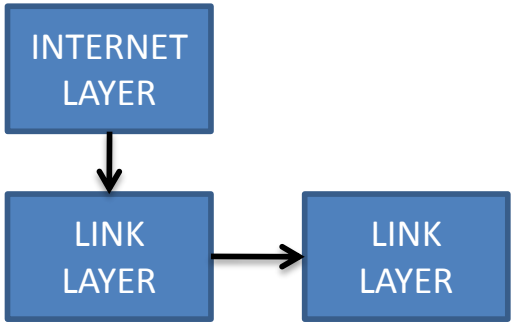
IP: Internet Protocol [Dataflow]



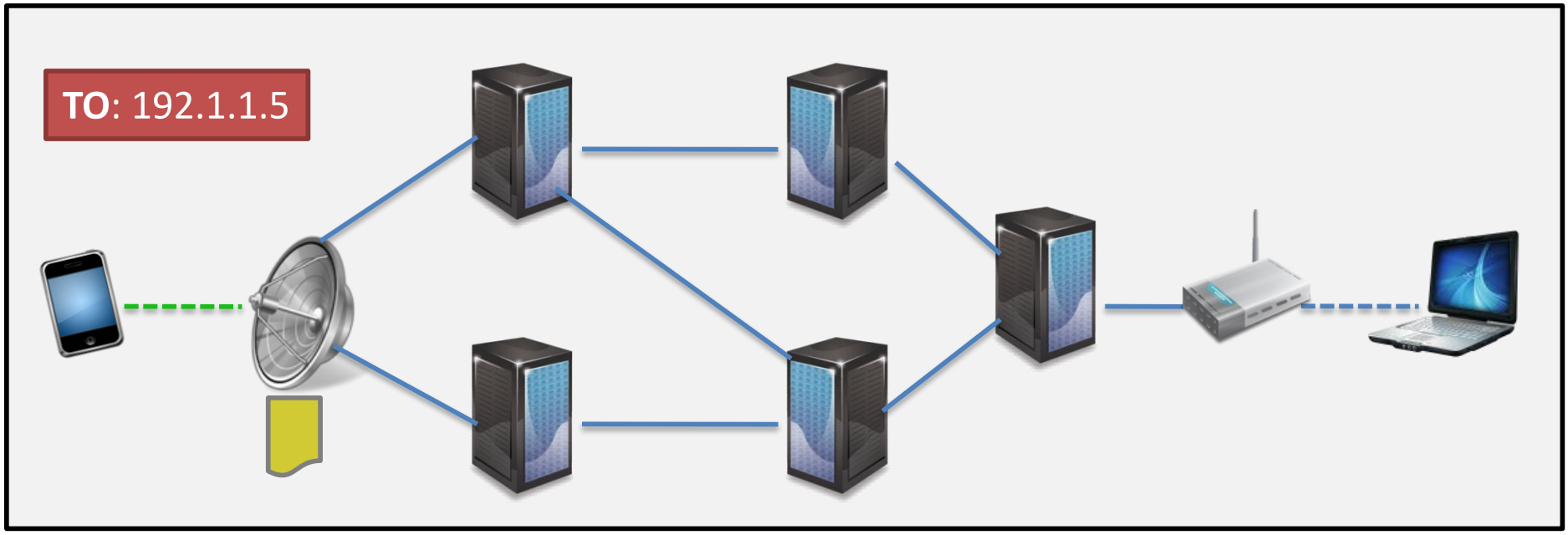


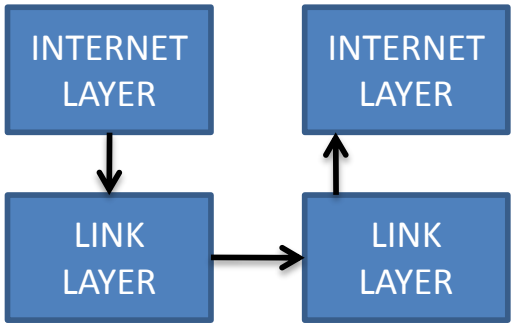
IP: Internet Protocol [Dataflow]



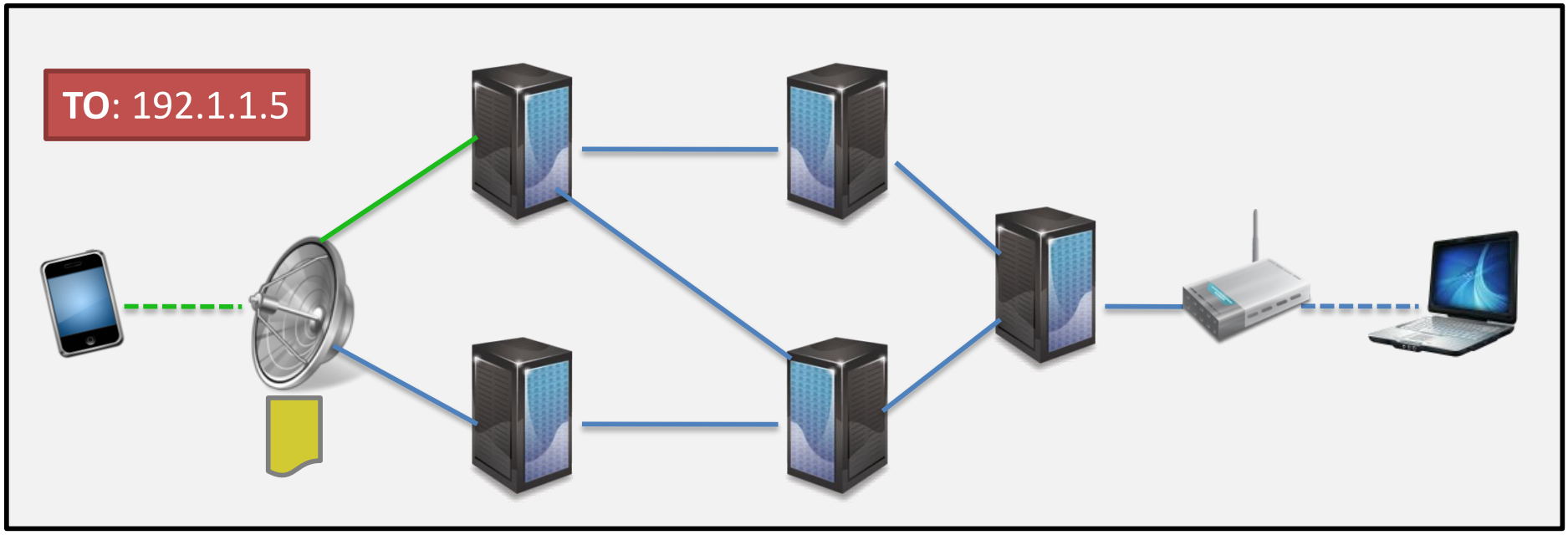


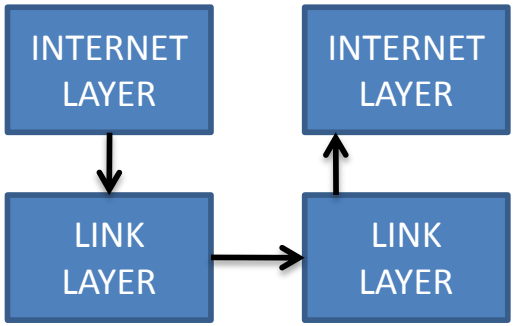
IP: Internet Protocol [Dataflow]



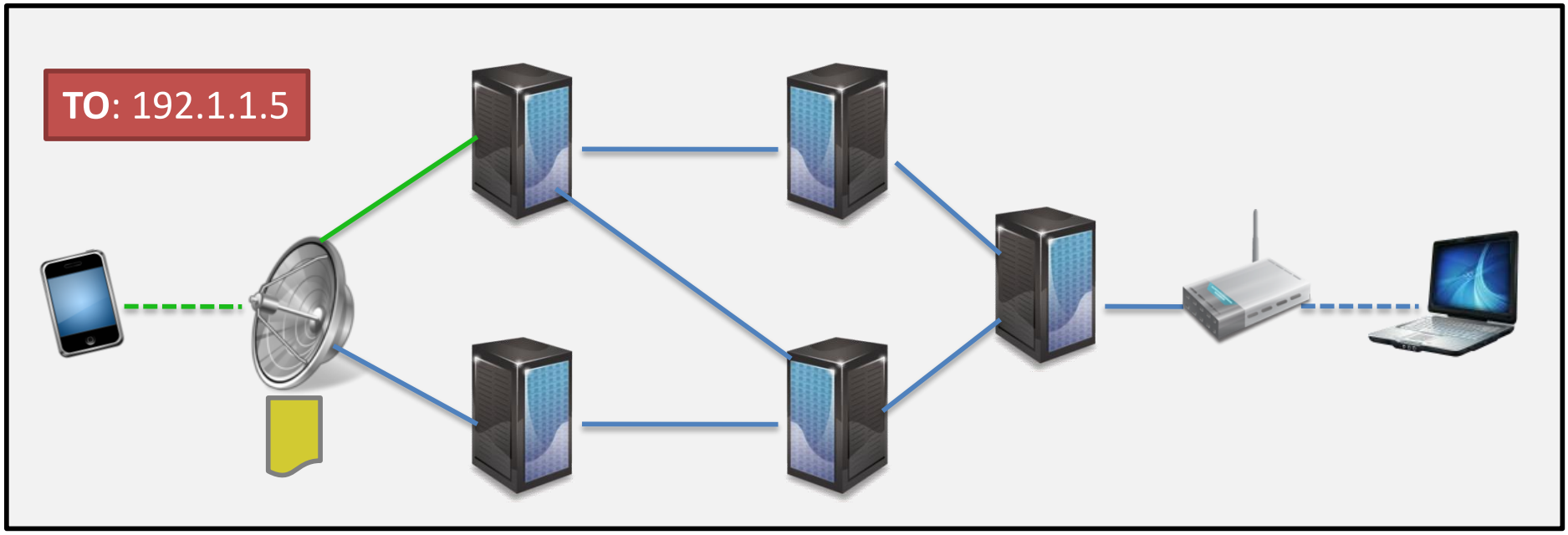


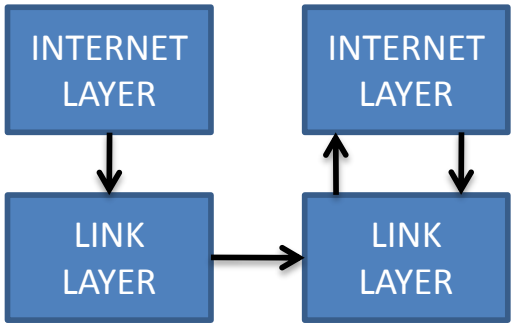
IP: Internet Protocol [Dataflow]



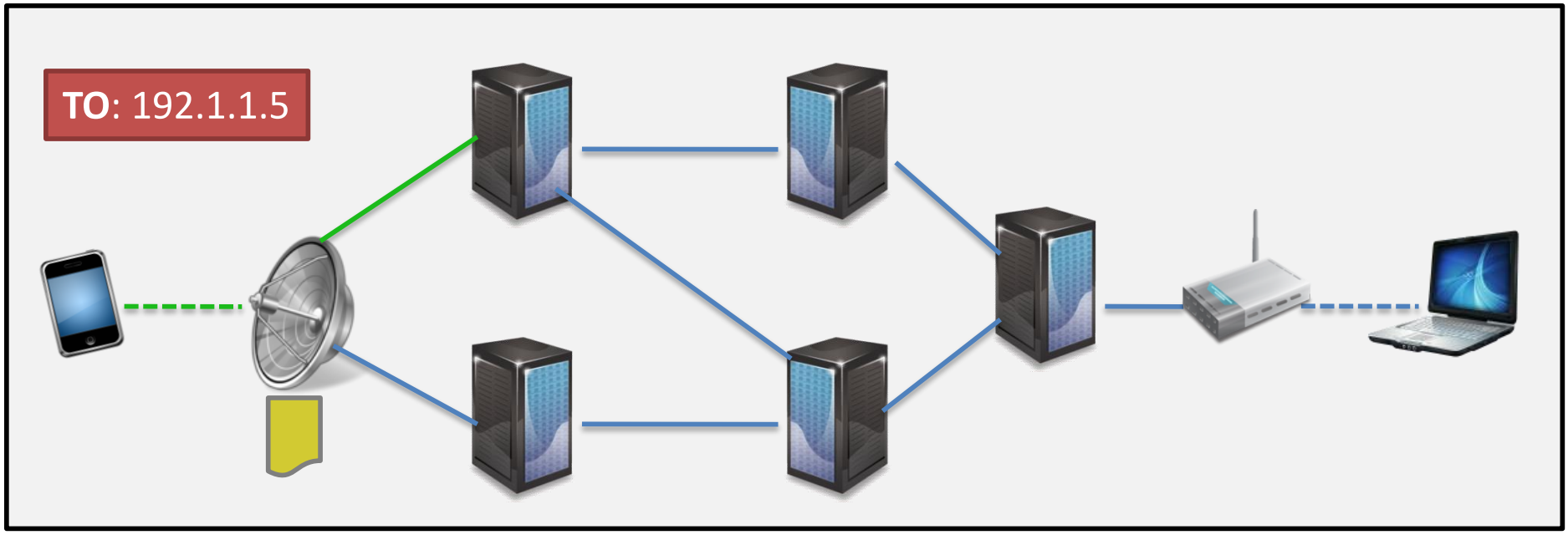


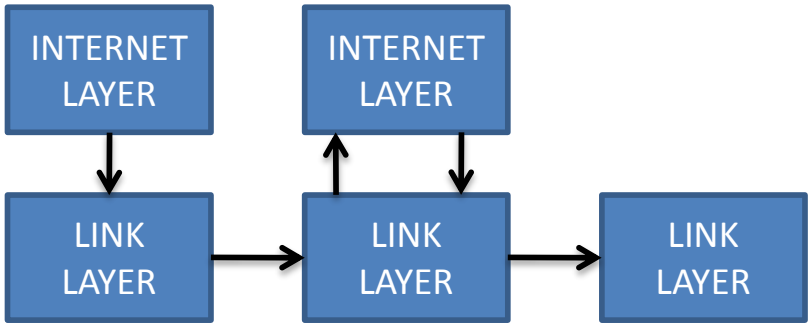
IP: Internet Protocol [Dataflow]



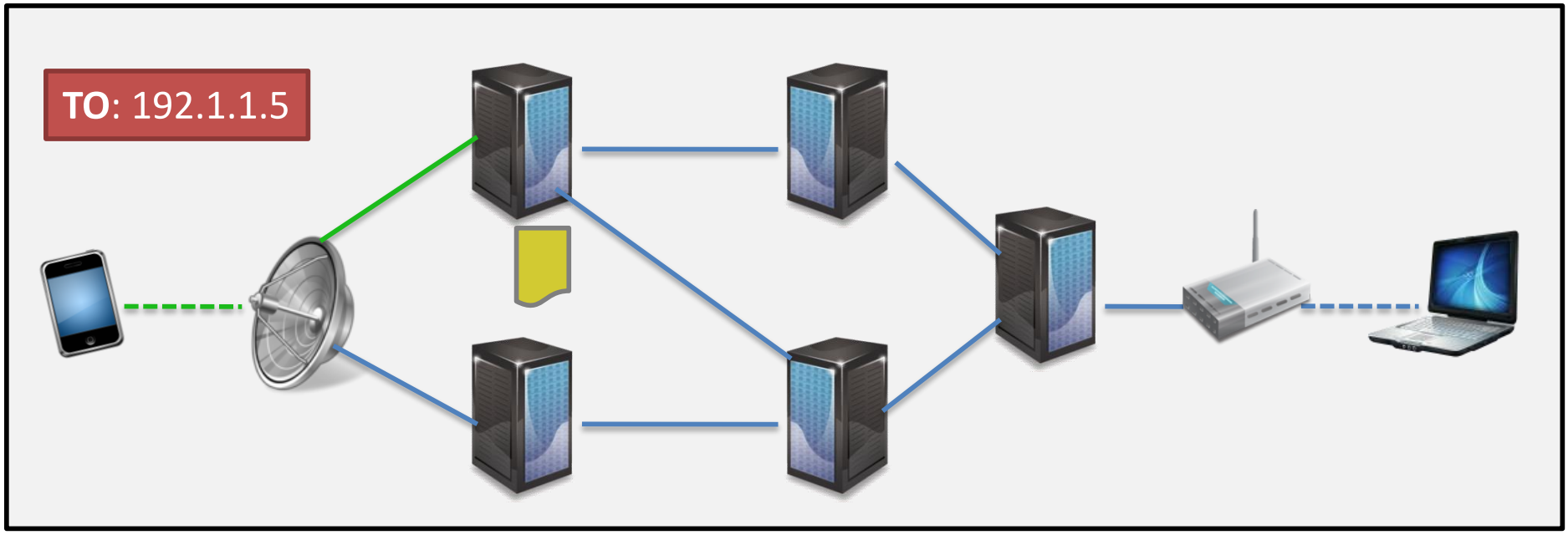


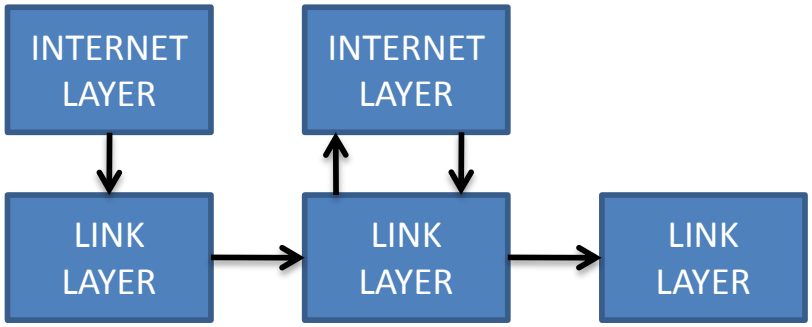
IP: Internet Protocol [Dataflow]



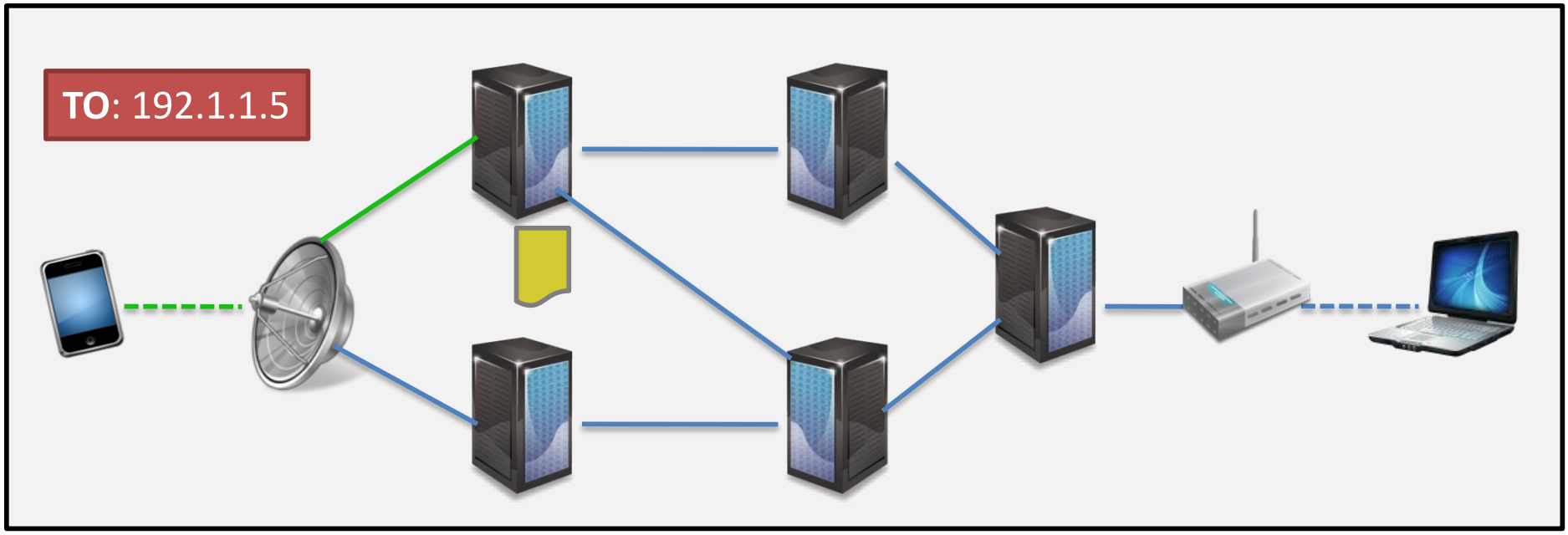


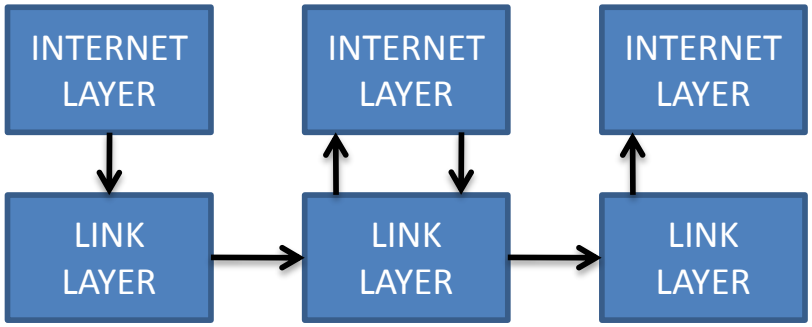
IP: Internet Protocol [Dataflow]



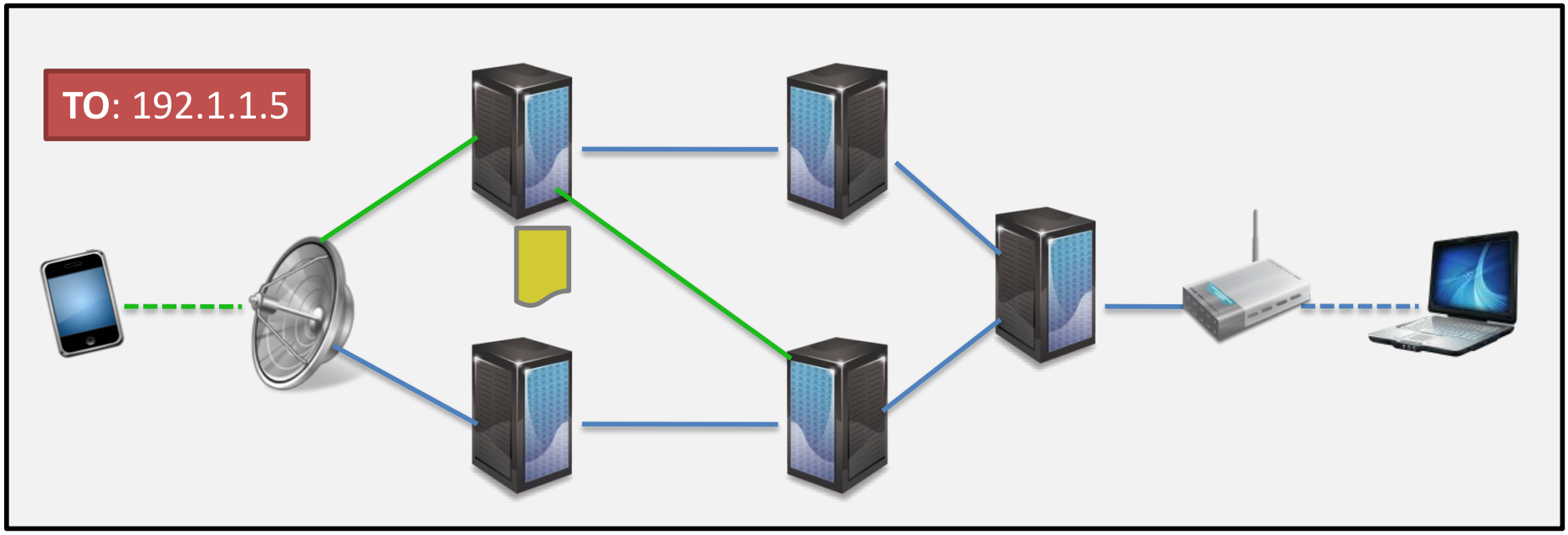


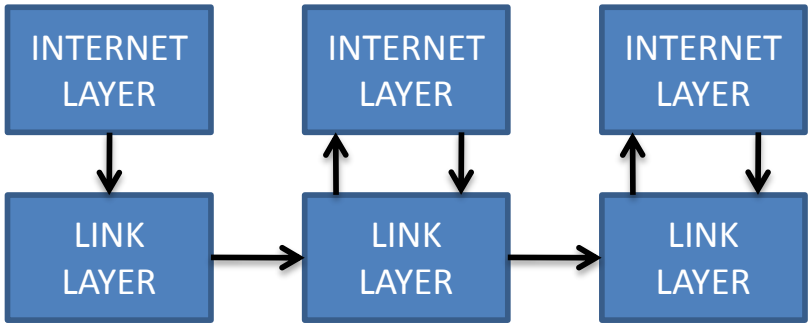
IP: Internet Protocol [Dataflow]



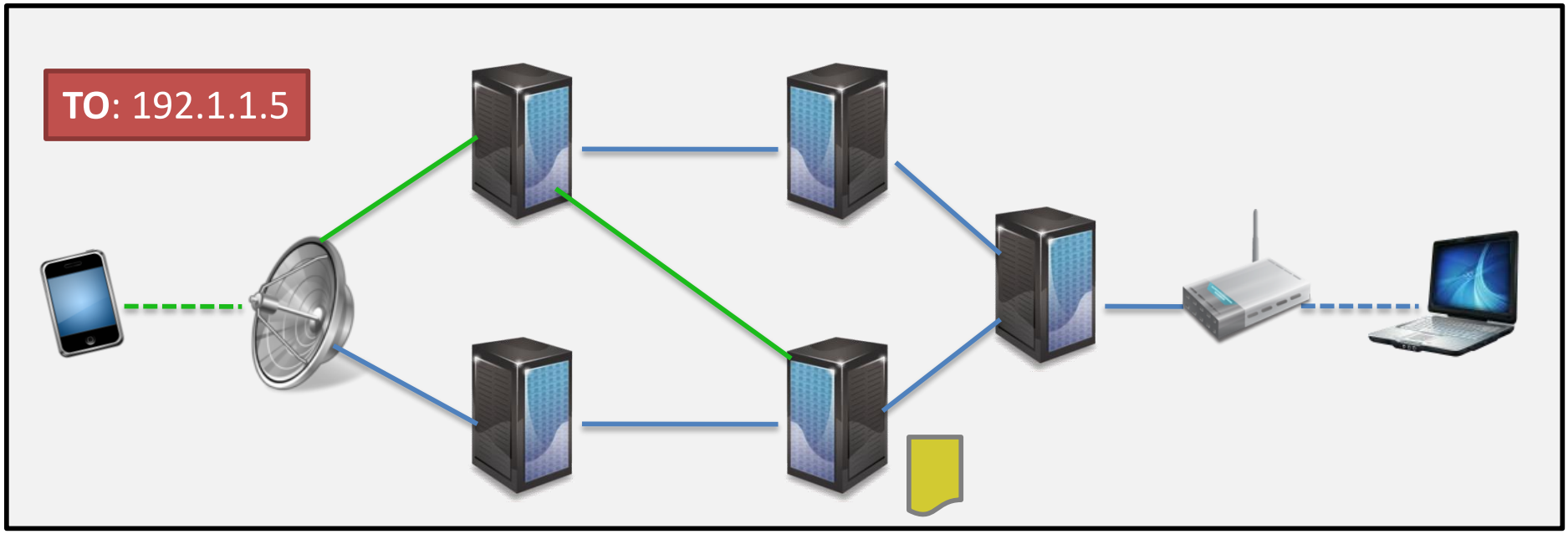


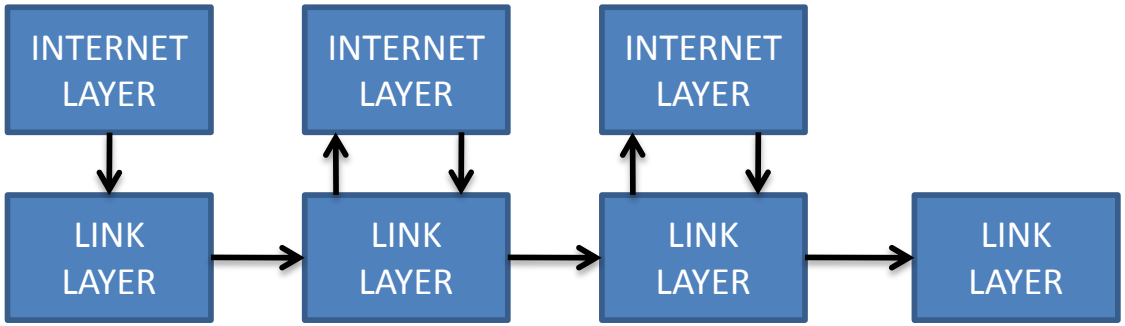
IP: Internet Protocol [Dataflow]



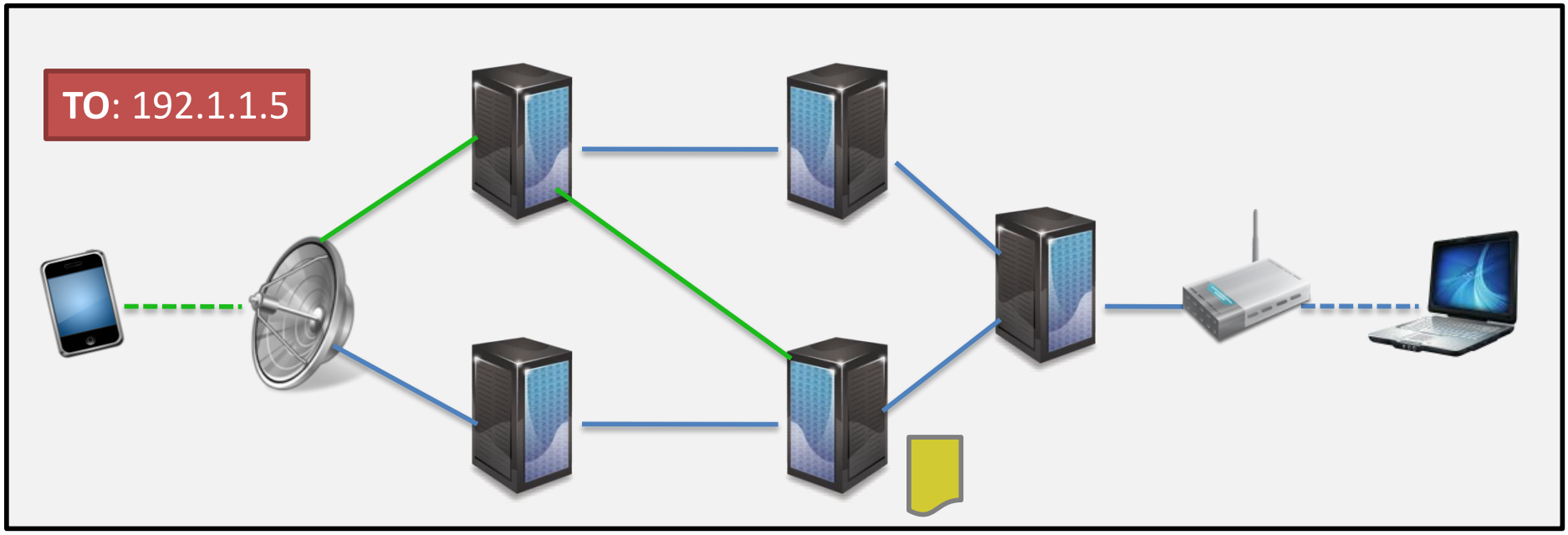


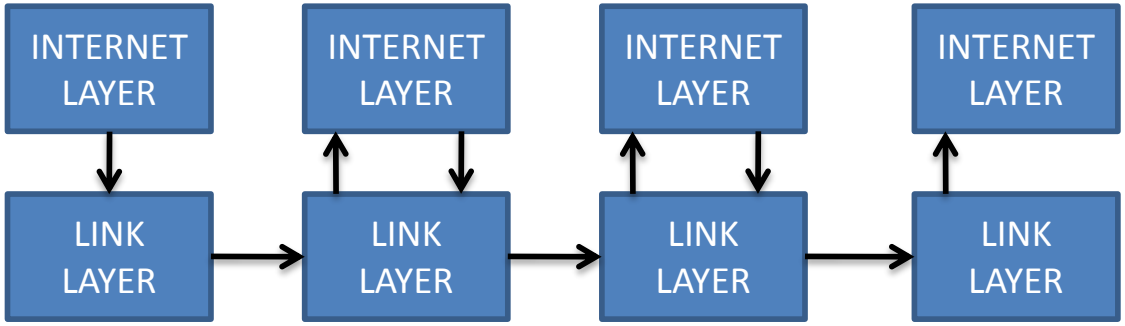
IP: Internet Protocol [Dataflow]



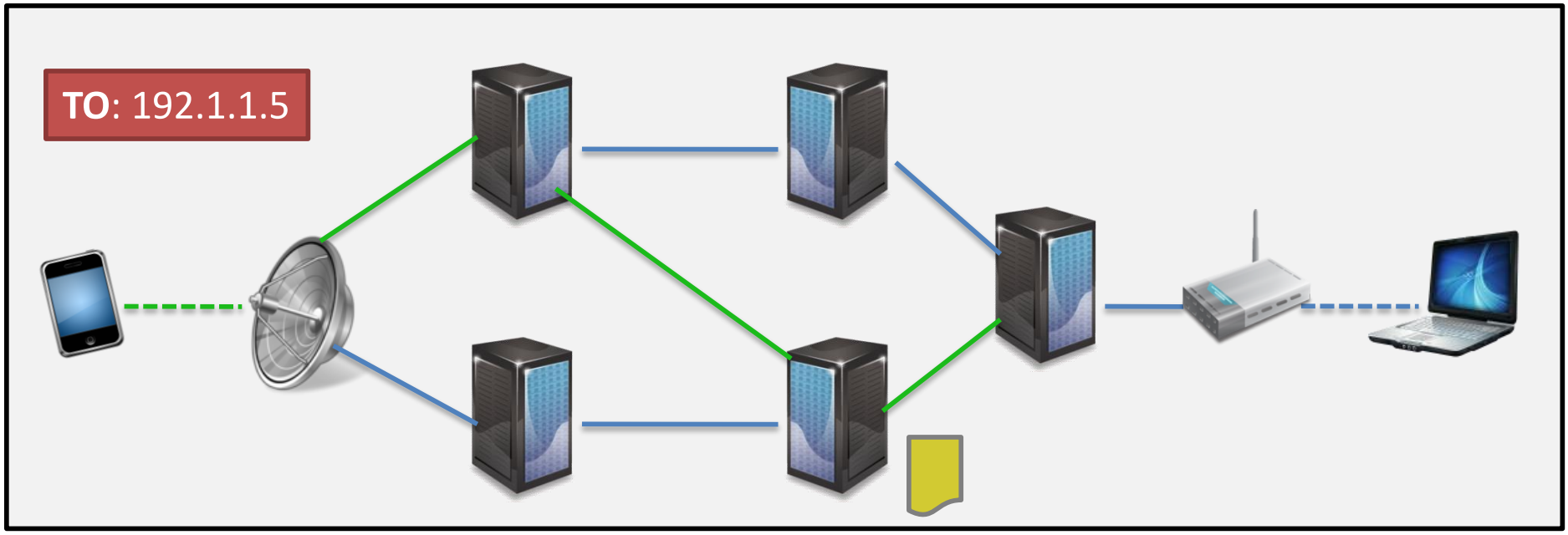


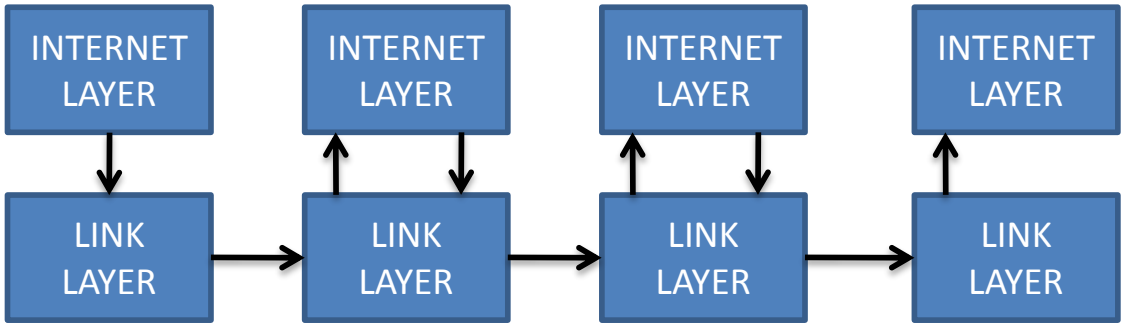
IP: Internet Protocol [Dataflow]



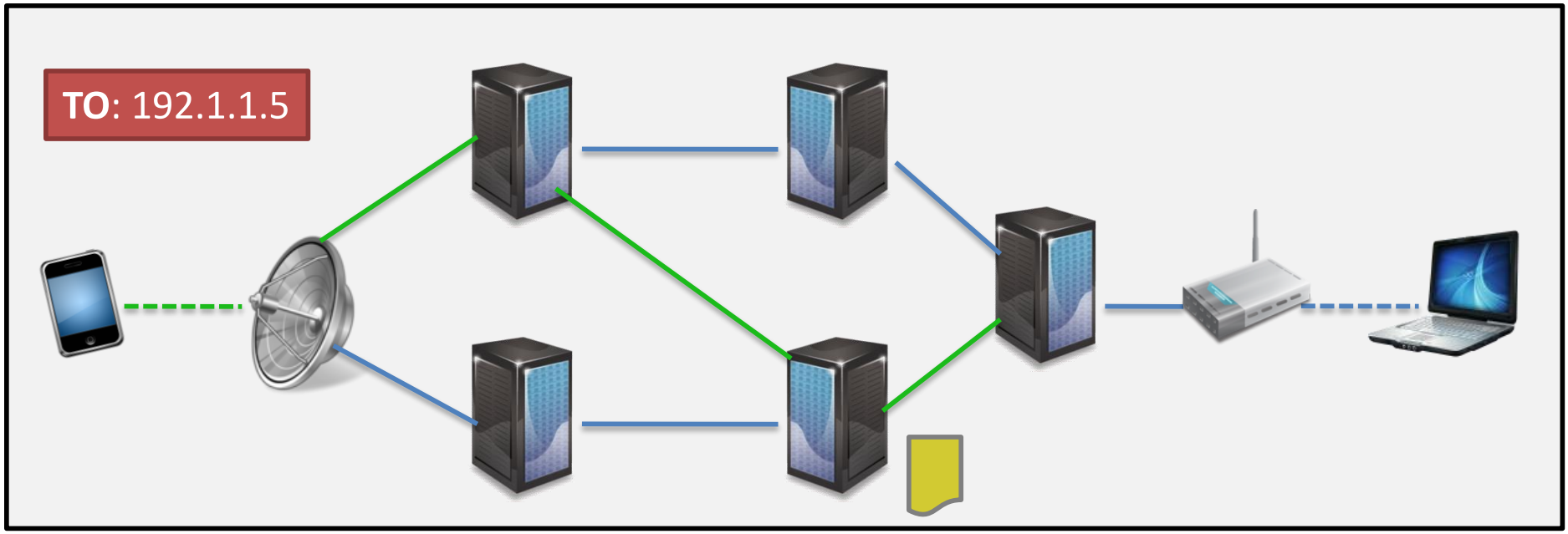


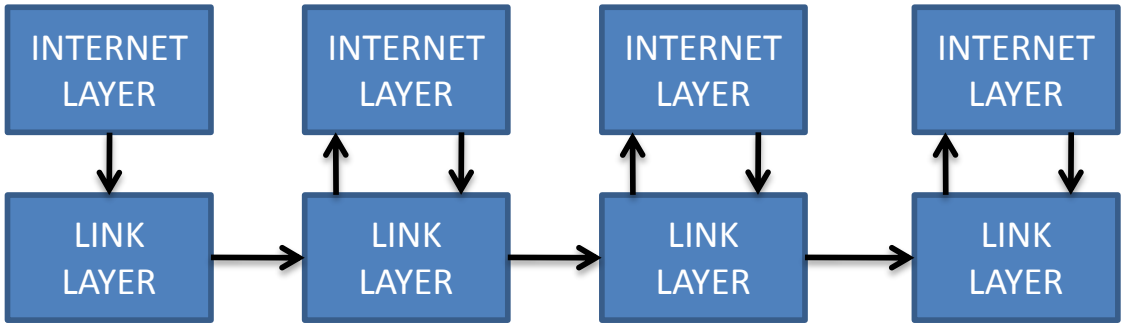
IP: Internet Protocol [Dataflow]



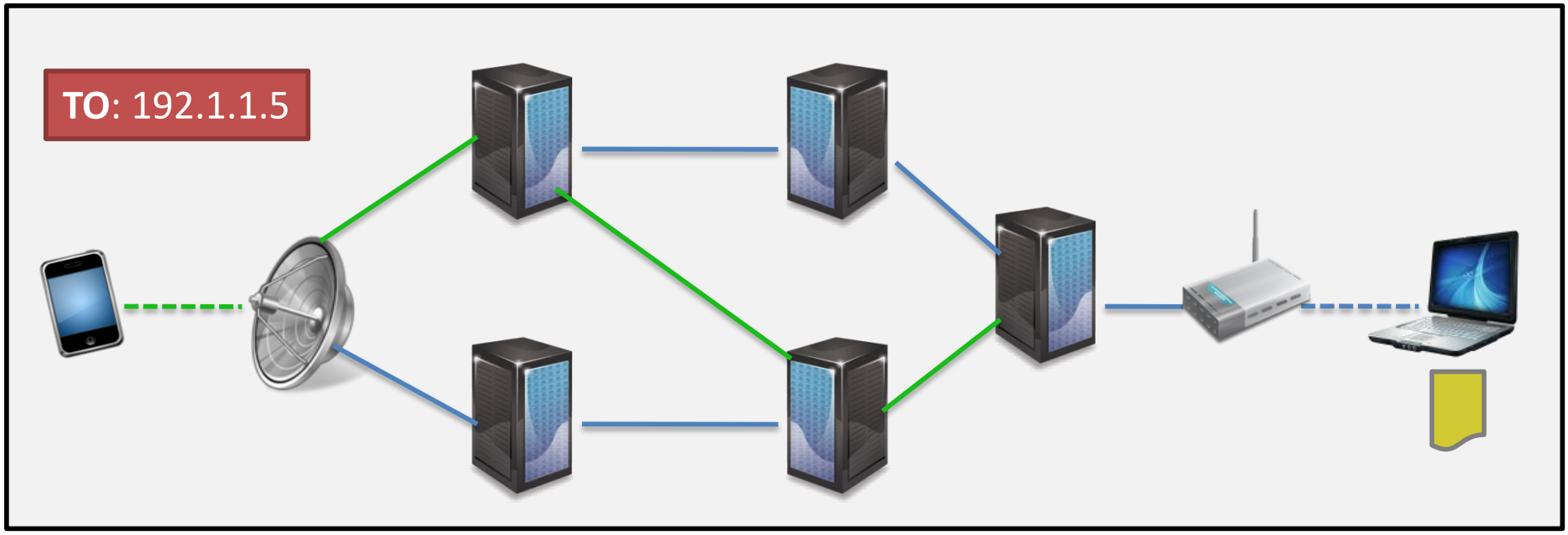


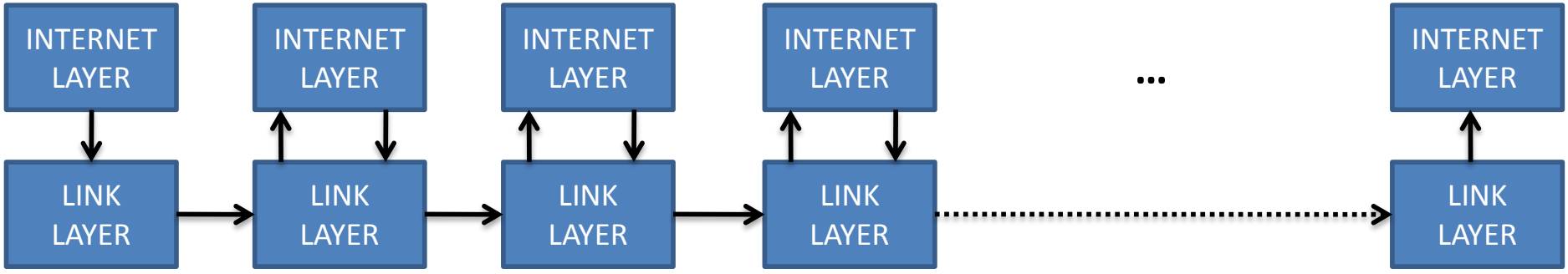
IP: Internet Protocol [Dataflow]





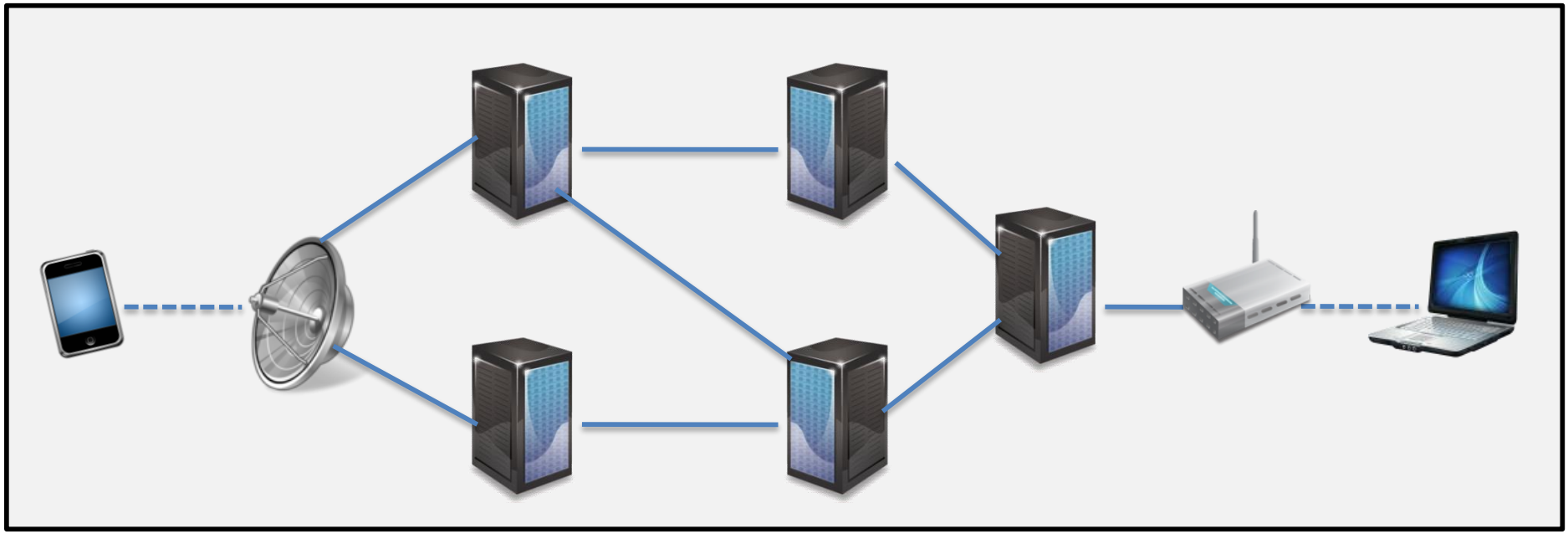
IP: Internet Protocol [Dataflow]





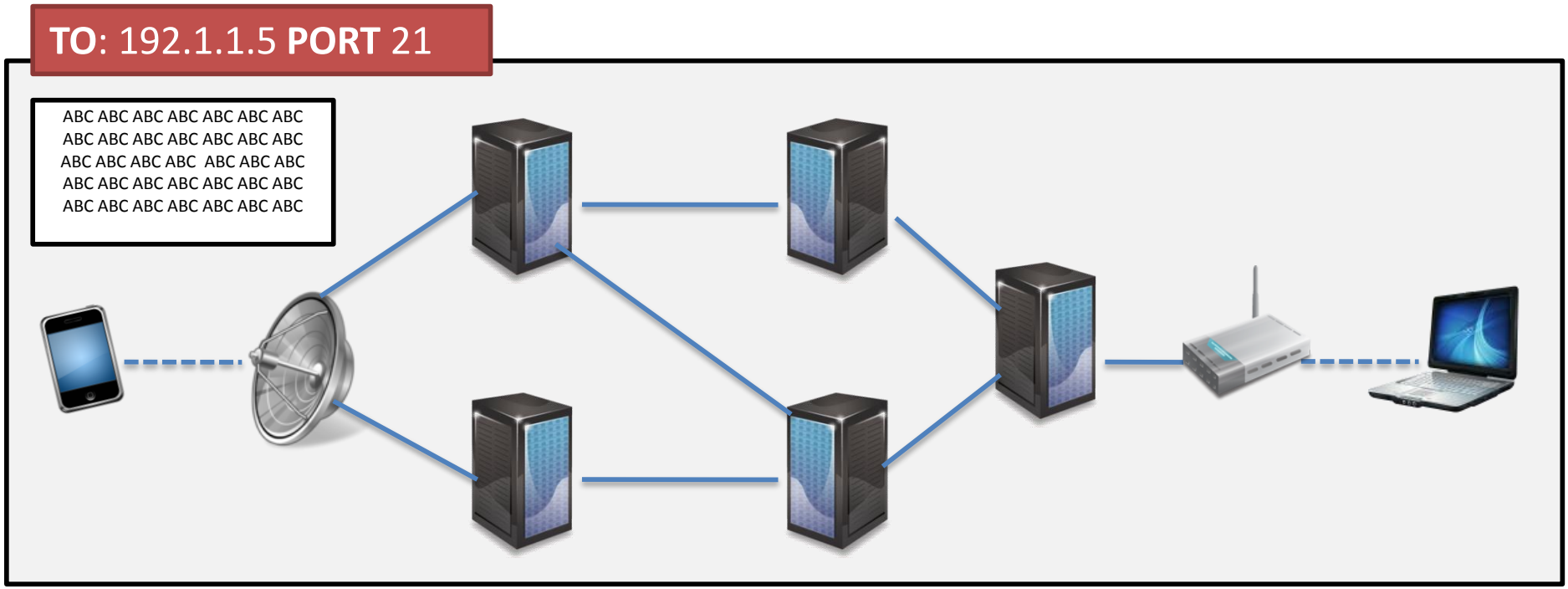
TCP: Transmission Control Protocol

- Transmission of arbitrary-length data in streams
- Transparently cuts up streams into fixed-size IP datagrams, and uses IP to send them to destination.
- Builds a reliable bi-directional communication channel on top of IP by retransmitting lost datagrams, reordering, etc.



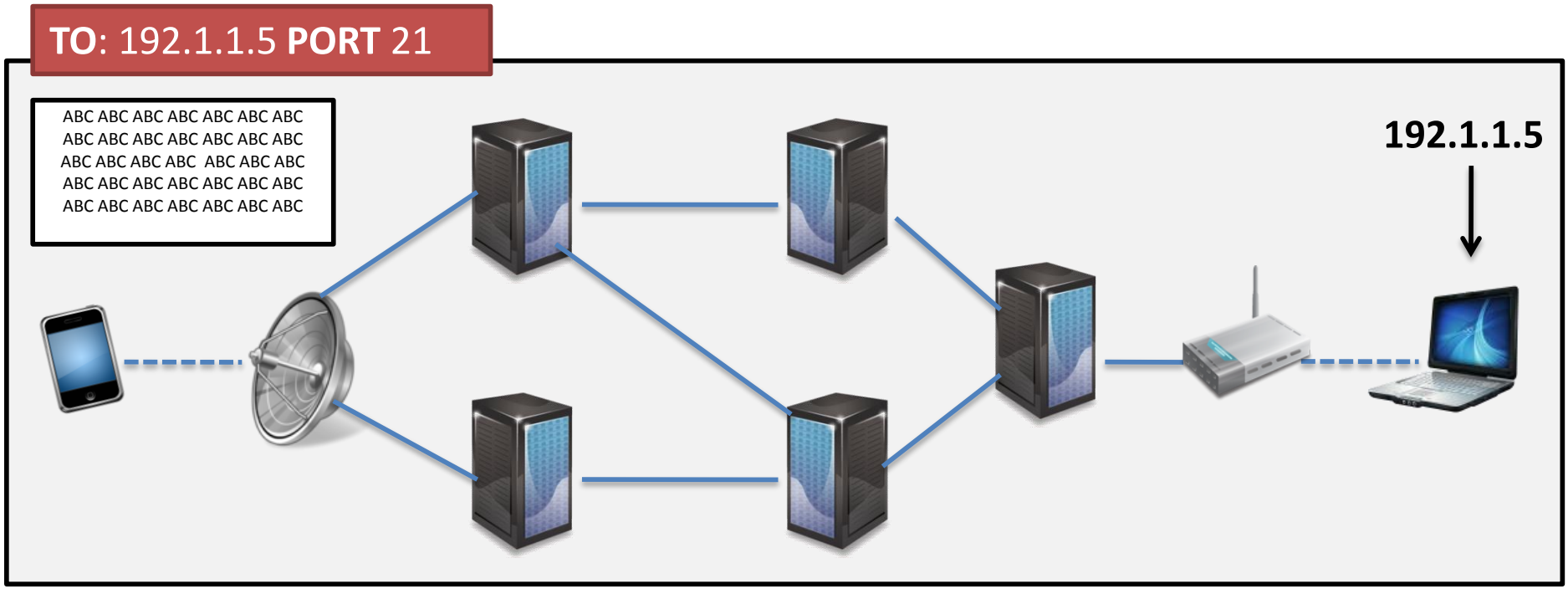
TCP: Transmission Control Protocol

- Transmission of arbitrary-length data in streams
- Transparently cuts up streams into fixed-size IP datagrams, and uses IP to send them to destination.
- Builds a reliable bi-directional communication channel on top of IP by retransmitting lost datagrams, reordering, etc.



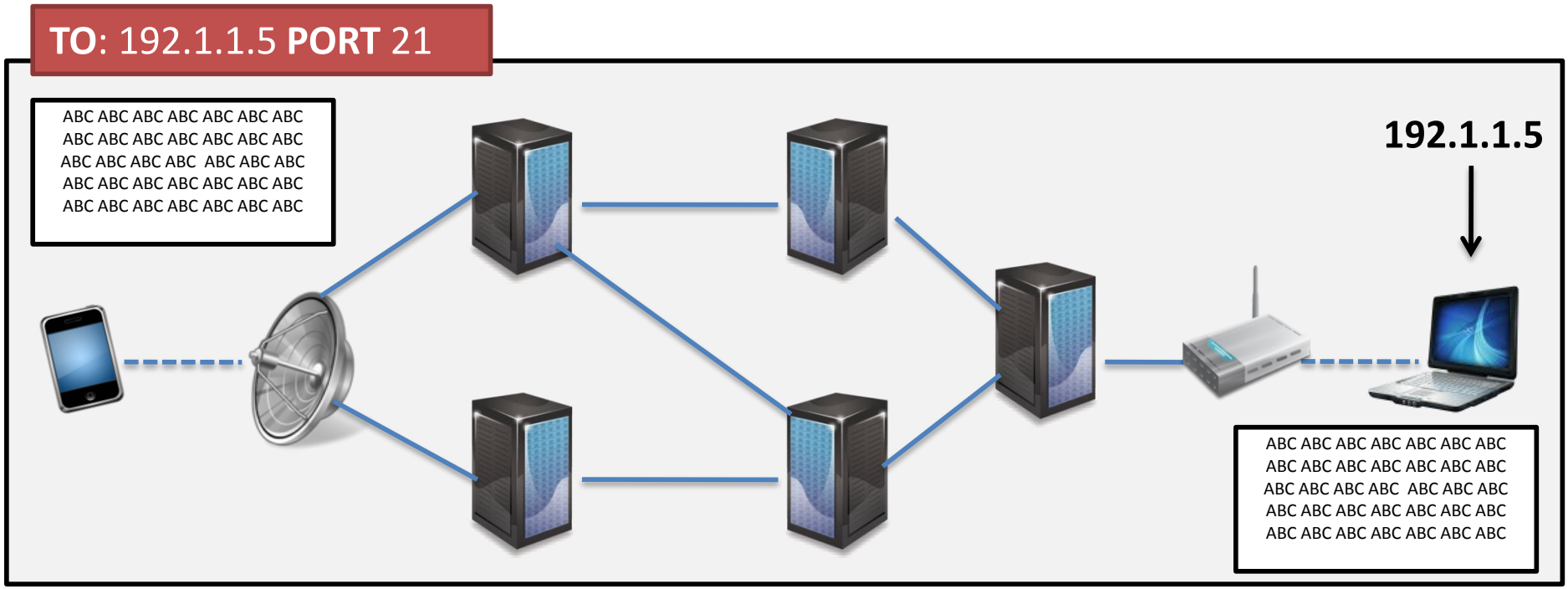
TCP: Transmission Control Protocol

- Transmission of arbitrary-length data in streams
- Transparently cuts up streams into fixed-size IP datagrams, and uses IP to send them to destination.
- Builds a reliable bi-directional communication channel on top of IP by retransmitting lost datagrams, reordering, etc.



TCP: Transmission Control Protocol

- Transmission of arbitrary-length data in streams
- Transparently cuts up streams into fixed-size IP datagrams, and uses IP to send them to destination.
- Builds a reliable bi-directional communication channel on top of IP by retransmitting lost datagrams, reordering, etc.



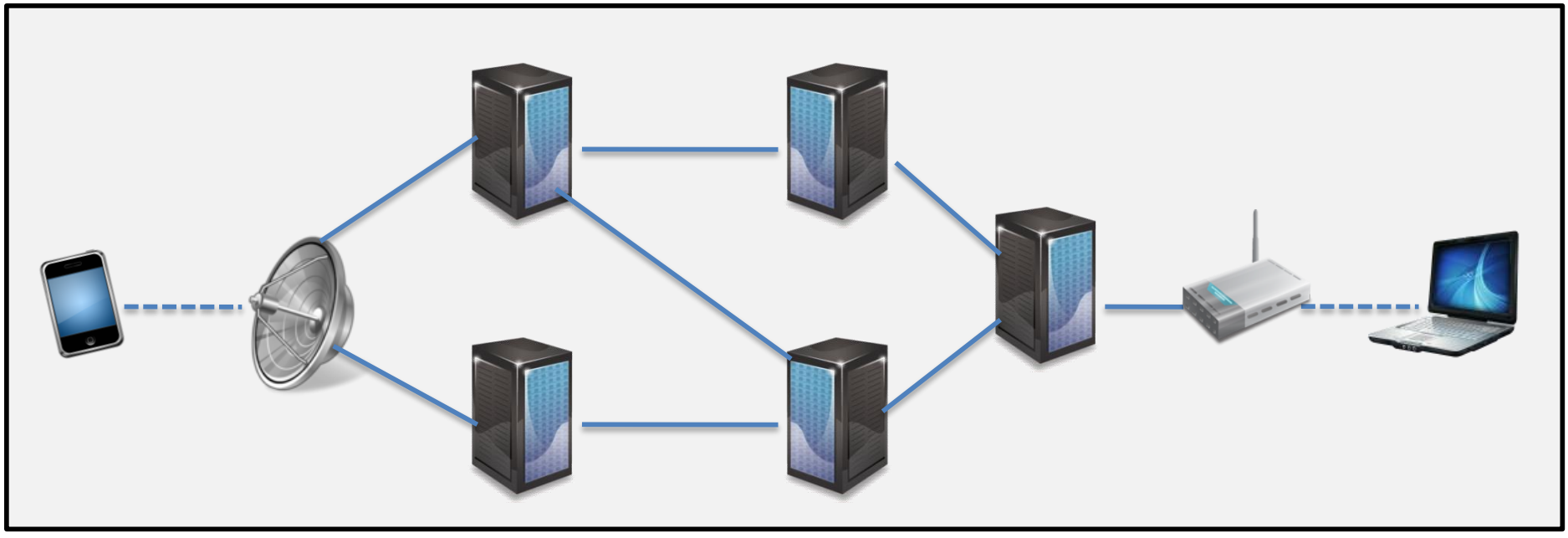
TCP: Transmission Control Protocol

- **TCP is Connection-oriented**
  - Establish connection between client and server process
  - Transmit data in both directions
  - Close connection
- End point of connection given by a pair

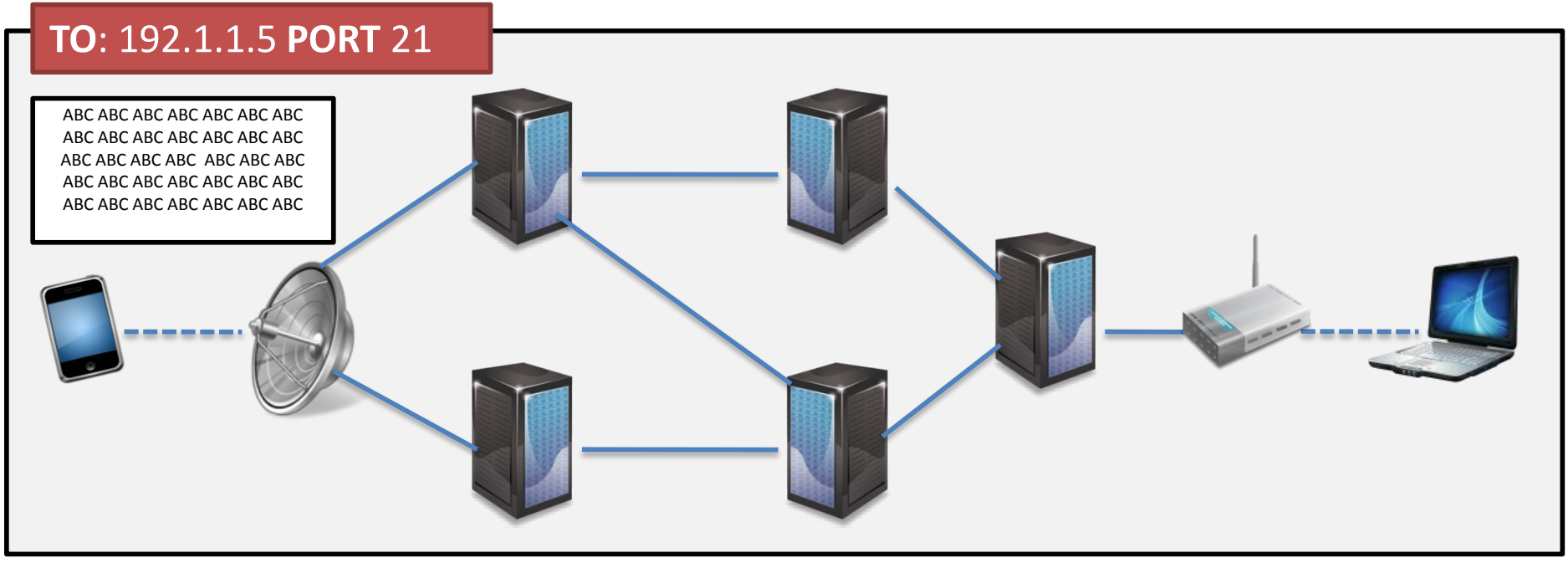
IP address : port number

- A port numbers identifies the server/client *process* for which the data is intended
- Standard services (email, web browsing, ftp) have a fixed port number (see http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml)

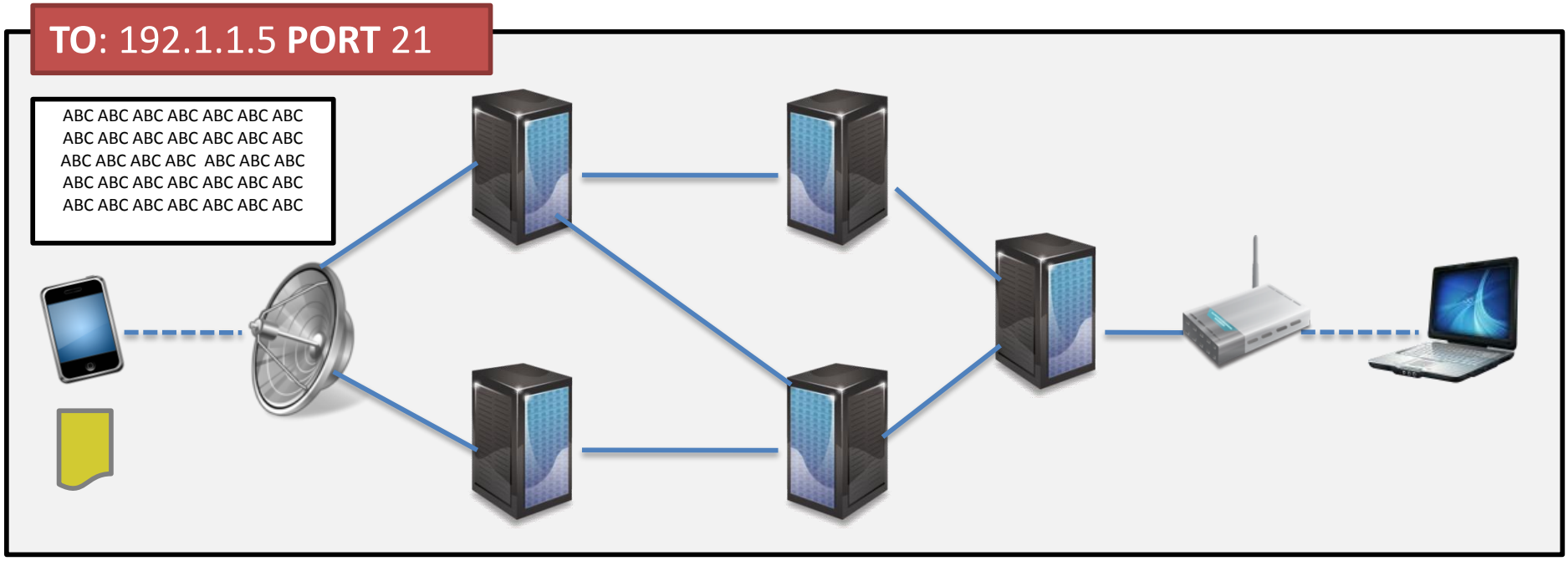
TCP [Dataflow]



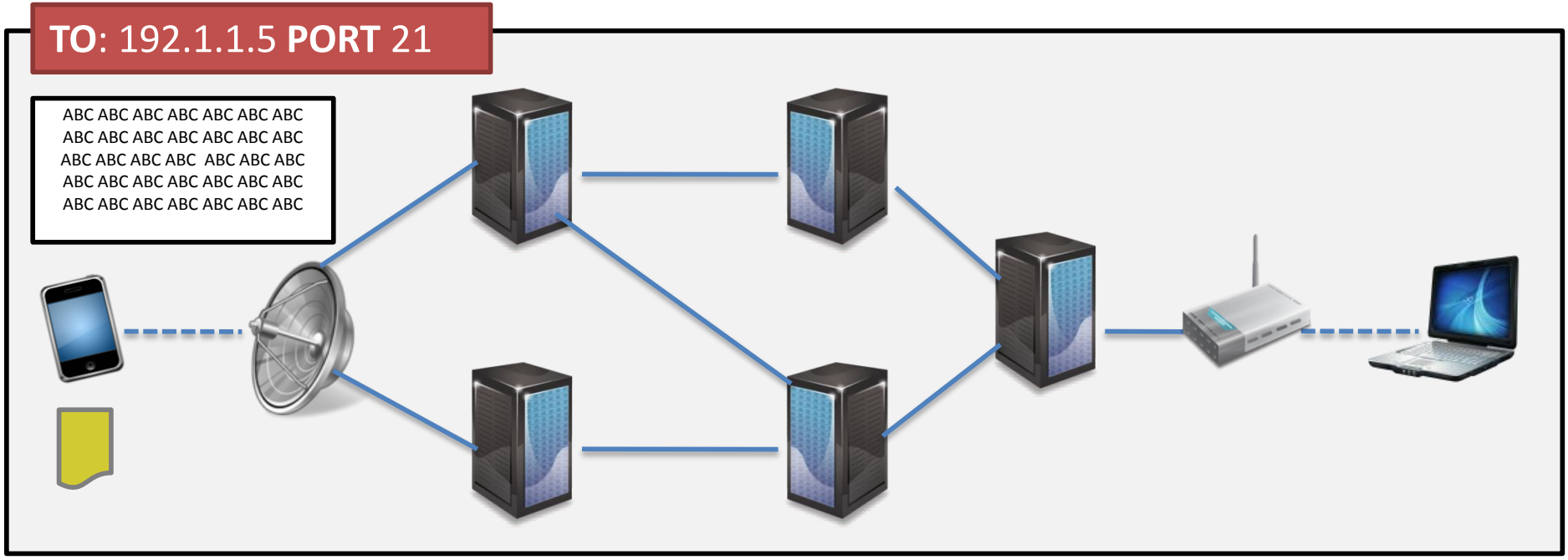
TCP [Dataflow]

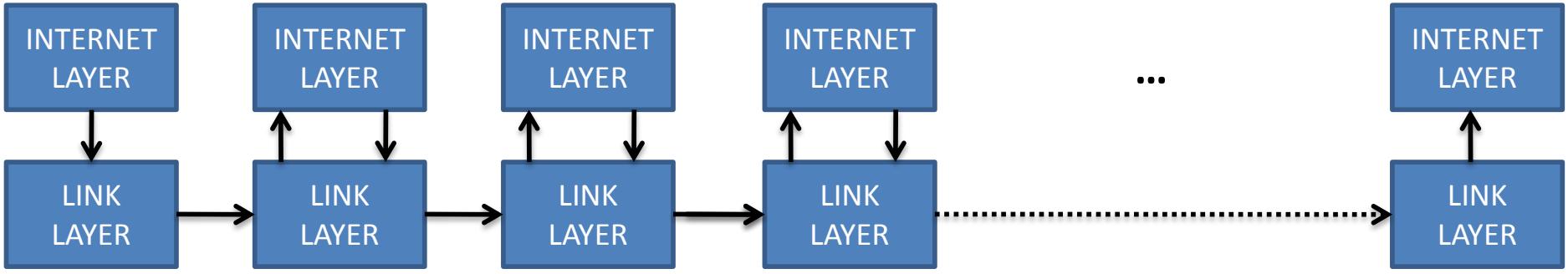


TCP [Dataflow]

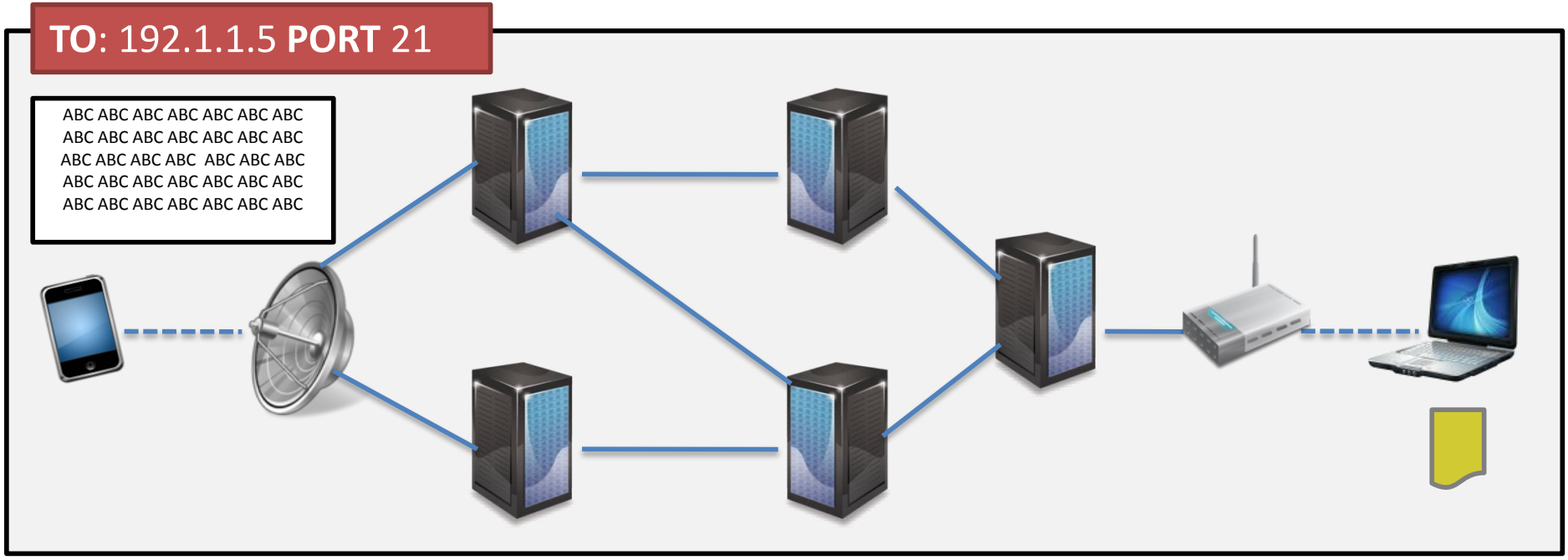


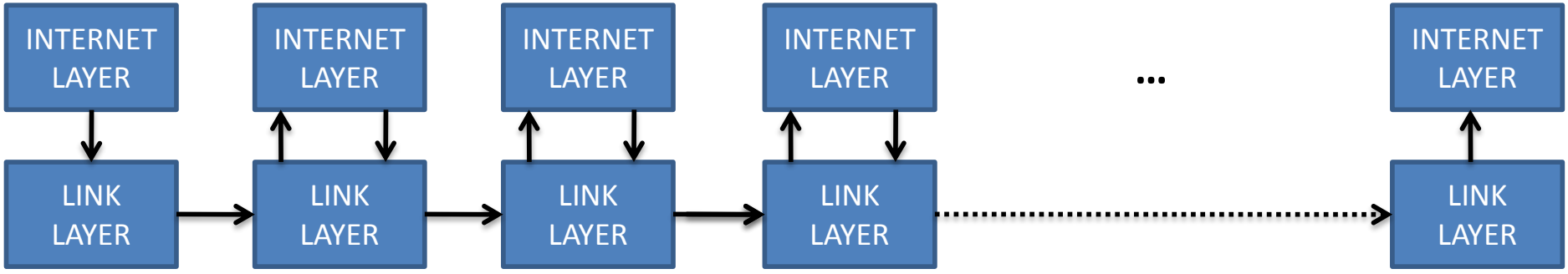
TCP [Dataflow]



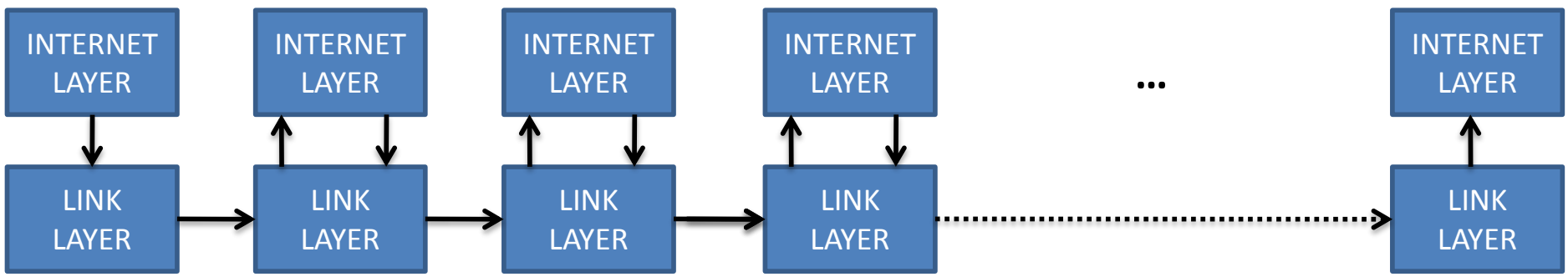


TCP [Dataflow]

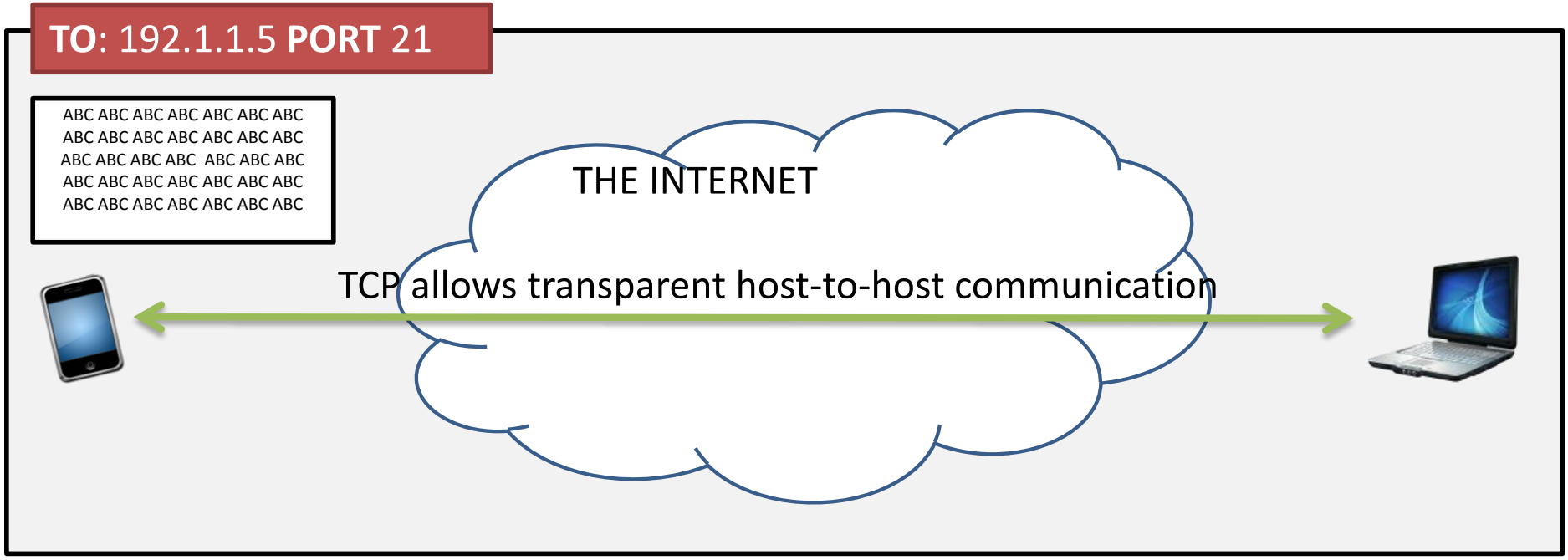


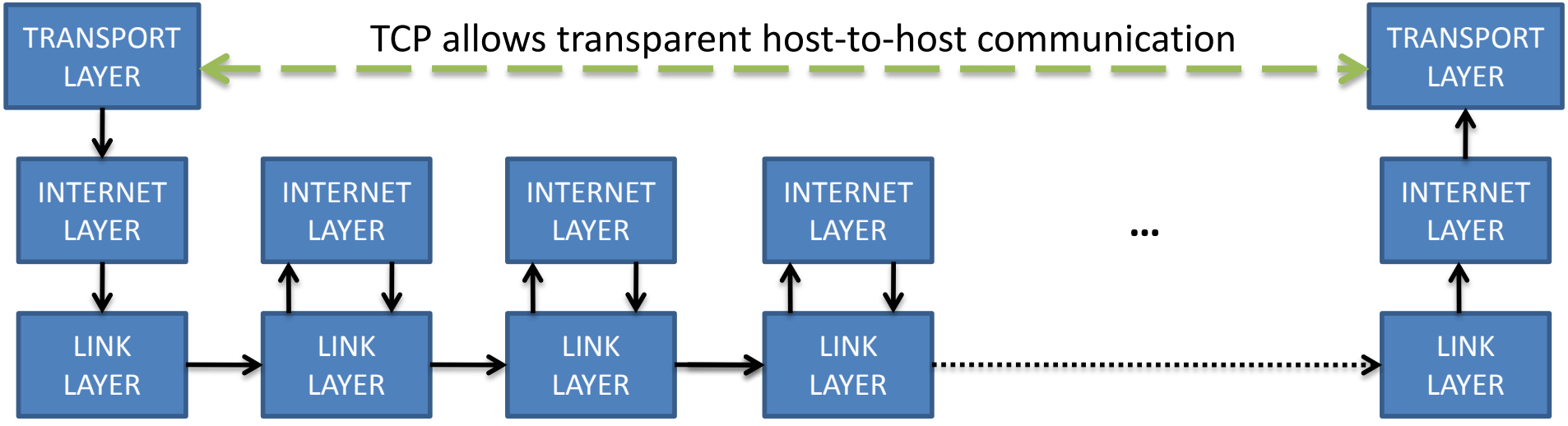


TCP [Dataflow]



TCP [Dataflow]

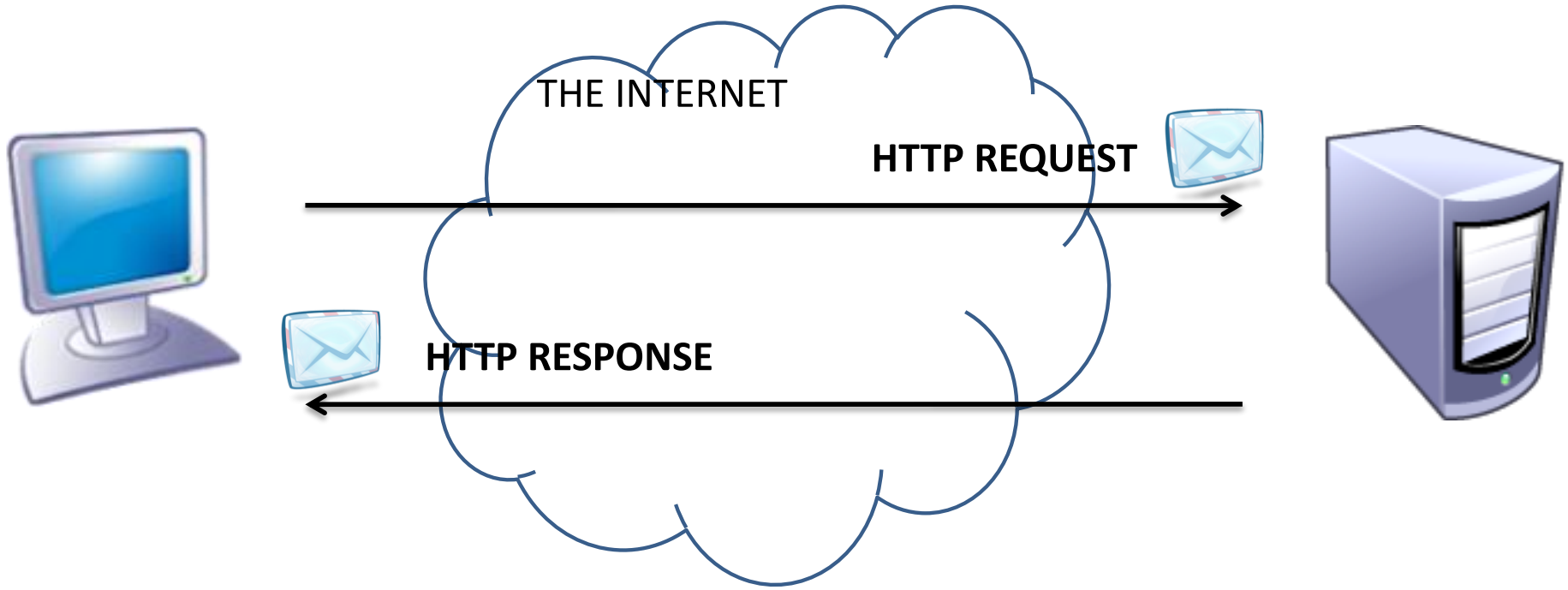




HTTP: Hypertext Transfer Protocol

- Layer on top of TCP
- Essentially an *envelope format*
- Request and response communication protocol for exchanging representations of web resources (e.g., HTML, XML, TEXT, …)
- Communication is always initiated by the client
- Server typically runs on port 80
- Stateless, light-weight

HTTP = Request/Response protocol



Effect of typing http://www.ulb.ac.be/index.html in web browser :

1. Use a **domain name service (DNS)** to get the IP address for www.ulb.ac.be
2. Create a TCP connection to address 164.15.59.215 on port 80
3. Send a HTTP request message over the TCP connection
4. Receive the HTTP response (and visualize in a browser)

Example HTTP Request message


```
GET /index.html HTTP/1.1
Host: www.ulb.ac.be
User-Agent:
Accept: text/html,application/xhtml+xml,application/xml
Accept-language: us,en;q=0.5
Accept-encoding: gzip,deflate
Accept-charset: ISO-8895-15,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
```

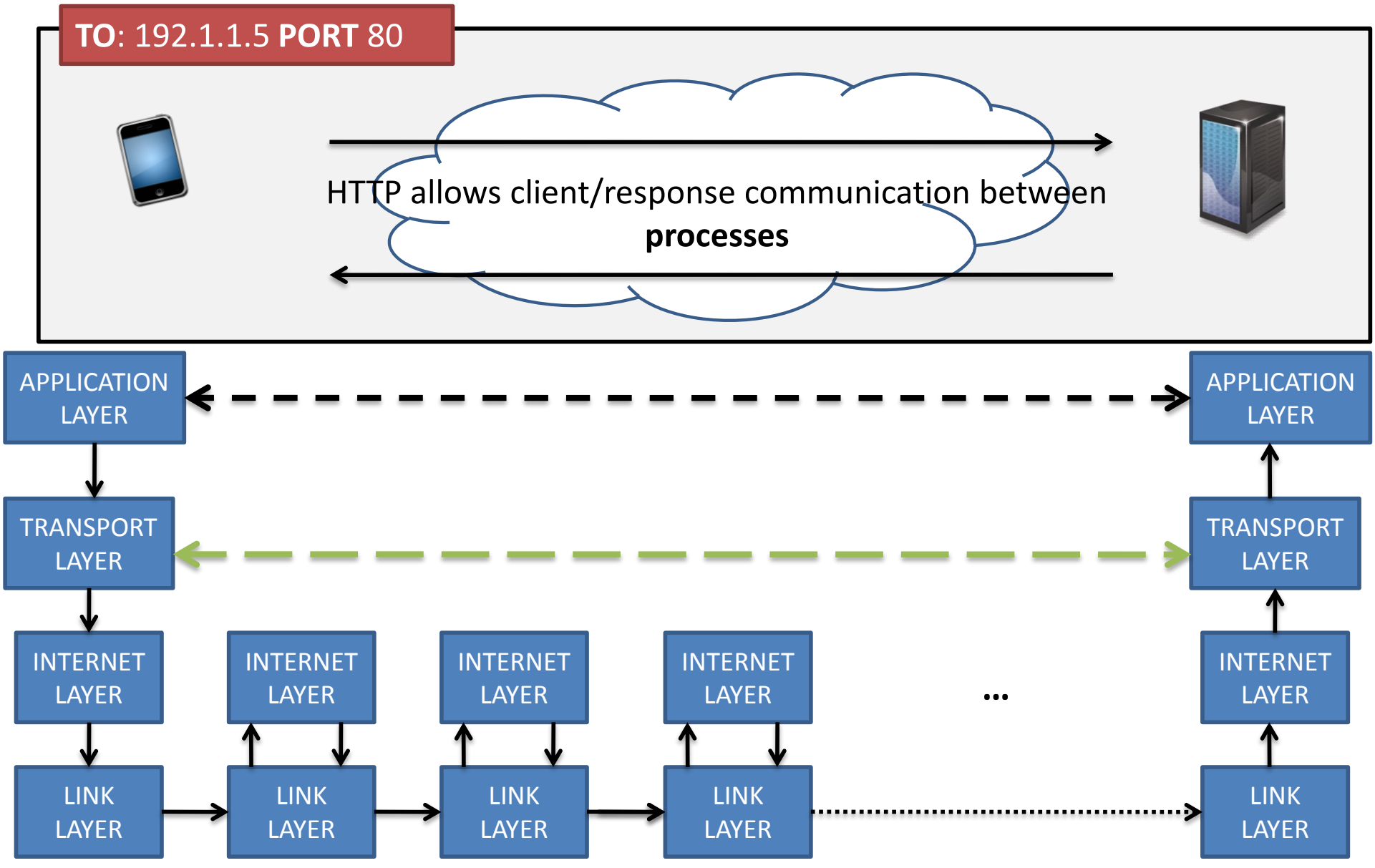
Example HTTP Response message


```
HTTP/1.1 200 OK
Date: Mon, 19 Dec 2011 16:39:16 GMT
Server: Apache/2.2.11 (Unix) mod_ssl/2.2.11 OpenSSL/0.9.7d
Last-Modified: Mon, 19 Dec 2011 15:48:25 GMT
ETag: "2725d3-11441-4b473e18e0130"
Accept-Ranges: bytes
Content-Length: 70721
Content-Type: text/html

<html xmlns=http://www.w3.org/1999/xhtml> … </html>
```


Demo

HTTP [Dataflow]



General structure of a HTTP request

- HTTP is a document-based protocol: the client puts a document in an envelope and sends it to the server
- The server replies with a response document in an envelope
- HTTP defines what the envelope should look like, but doesn't care what goes inside

HTTP Method **Path** **Version**

```
GET /index.html HTTP/1.1
Host: www.ulb.ac.be
User-Agent: Mozilla/5.0 …
Accept: text/html, …
Accept-language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15,…
Connection: keep-alive

```

Start Line

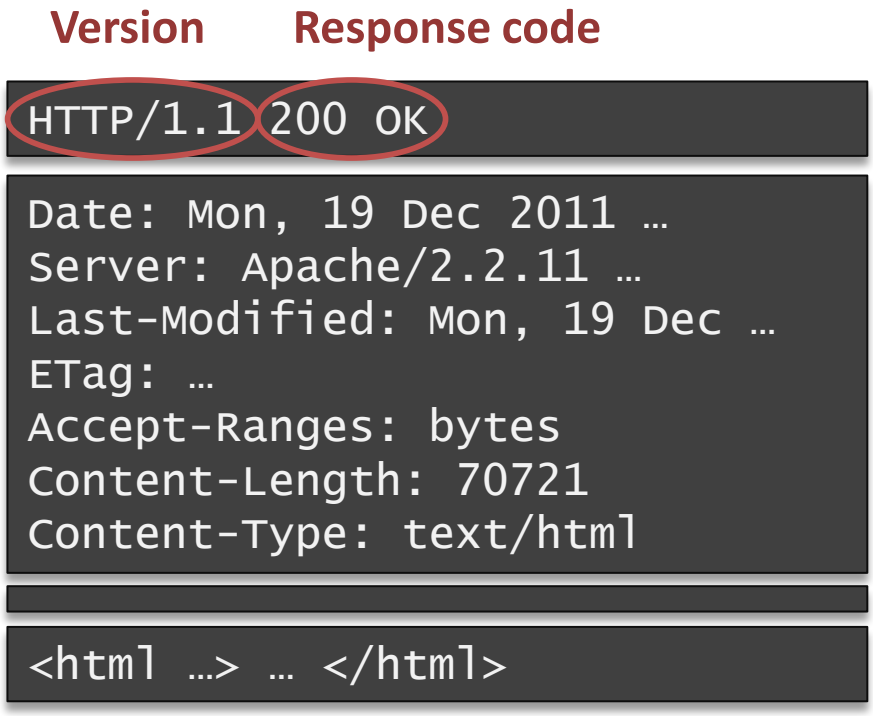
Request Headers
(= a list of key/value pairs)
"the stickers on the envelope"
Many standard headers
User-defined headers possible

Empty line (CRLF)

The optional entity-body (document, resource representation)

General structure of a HTTP response

- HTTP is a document-based protocol: the client puts a document in an envelope and sends it to the server
- Server replies with a response document in an envelope
- HTTP defines what the envelope should look like, but doesn't care what goes inside



Status Line

Response Headers
(= a list of key/value pairs)
"the stickers on the envelope"
Many standard headers
User-defined headers possible

Empty line (CRLF)

The optional entity-body (document, resource representation)

HTTP Methods [request only]

- The HTTP methods that can be used in a request are:

GET	request a resource representation
POST	send data to server and receives result
PUT	create or update a resource
DELETE	delete a resource
OPTIONS	Discover what HTTP methods are supported at target URI
HEAD	requests headers only (similar to GET but omits entity body)

- Most websites use only GET

The message body [request + response]

- The body is a sequence of bytes
- If a HTTP message includes a body, there is usually a header line that describes the MIME format of the body

```
Content-Length: 70721
Content-Type: text/html
```

Length in bytes

MIME type

(others include image/gif, application/xml, application/json, …)

- The MIME type identifies how the entity body should be interpreted.
- Full list of MIME types: http://www.iana.org/assignments/media-types/index.html

HTTP response codes [response only]

- The response code is a three-digit integer, where the first digit identifies the general category of response:
  - 1xx indicates an informational message only
  - 2xx indicates success of some kind
  - 3xx redirects the client to another URL
  - 4xx indicates an error on the client's part
  - 5xx indicates an error on the server's part
- The most common status codes are:

```
200 OK
```

```
404 Not Found
```

- The code is followed by a human-readable phrase, which may vary from server to server

Content negotiation

- Used to select “best” response for a request
  - Server-driven negotiation
  - Agent-driven negotiation
  - Transparent negotiation (not discussed here)

- Server-driven negotiation is normally used

Server-driven content negotiation

- User agent provides a list of preferences

```
Accept: text/html, …
Accept-language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15,…
```

List of MIME media types

List of languages

List of content encodings

List of character sets

- Server decides “best” resource representation from those available
- Vary header may be returned to tell caches what request headers are/can be used to make the decision, e.g.,

```
Vary: Accept-language,Accept-Charset
```


Demo

Agent-driven content negotiation

- User agent provides a list of preferences, as before
- Server returns status code 300 and a list of choices (with their URIs) in either header fields or entity-body
- User-agent decides which one is “best” and issues new request for it (using the provided URIs)
- Requires multiple trips to the server
- Less widely used

Conditional requests

- Certain request headers can be included to make a request conditional
- Based on date and time of last modification:

```
If-Modified-Since: Sat, 28 Jan 2012 19:43:31 GMT
```

```
If-Unmodified-Since: Sat, 28 Jan 2012 19:43:31 GMT
```

- Based on entity tags (returned by server in ETAG: header)

```
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
```

```
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
```

- When specified condition is true, server returns requested resource, otherwise a status code is returned with no message-body (304 Not Modified or 412 Precondition Failed)

Design advantages of HTTP

- Lightweight
- No client state on server, hence scalable (load balancing through multiple servers)
- HTTP brings a *uniform* interface to sharing data on the web (more on this later)

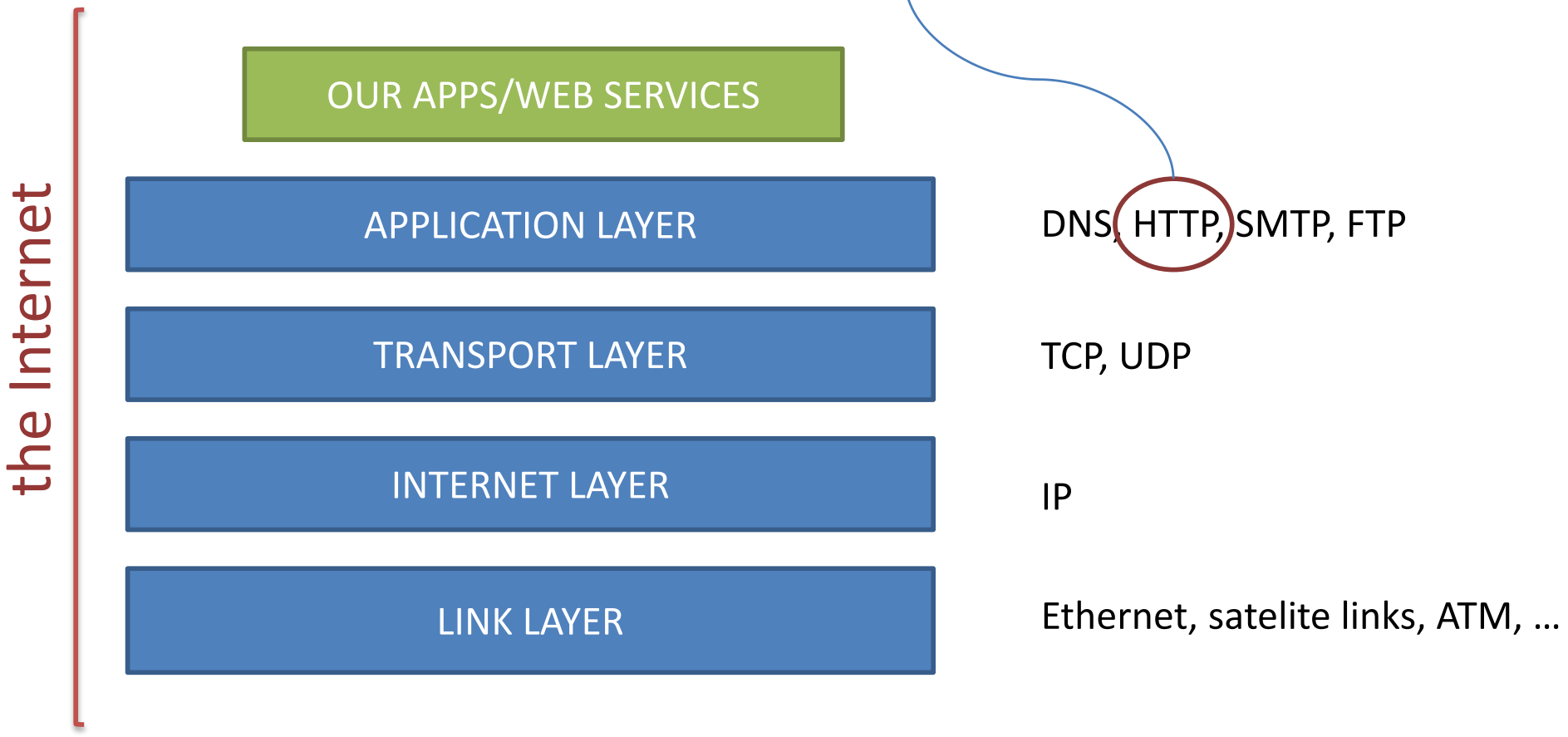
Web Servers

- A Web Server is a server that talks HTTP
- Responsibilities:
  1. Setup connection
  2. Receive & process HTTP requests
  3. Create & send HTTP response
  4. [Logging]

- Well-known web servers:
  - Apache HTTP Server [www.apache.org]
    Freely available
  - Microsoft Internet Information Services

Conclusion: The Web ≠ the Internet

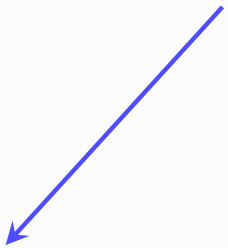
the Web = HTTP + URIs + Resource Representations

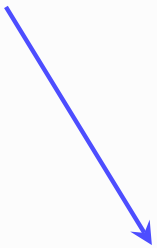


CONSTITUTENT 3: HTML AND CSS AS REPRESENTATION FORMAT

HTML = A MARKUP LANGUAGE FOR HYPERTEXT

HTML = HyperText Markup Language





Definition

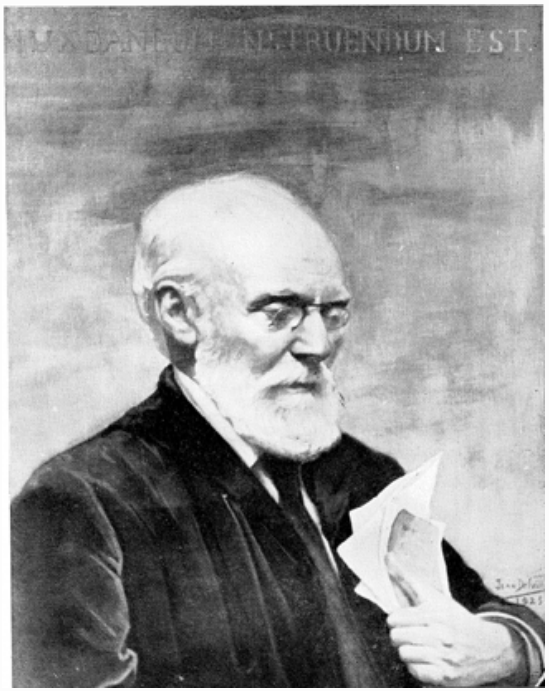
Hypertext is text displayed on a computer or other electronic device with references (hyperlinks) to other text that the reader can immediately access, usually by a mouse click or keypress sequence.

Definition

A Markup Language is a system for annotating a text with structure in a way which is syntactically distinguishable from that text.

THE HISTORY OF HYPERTEXT

- **1934**: Paul Otlet hints at hypertext in his philosophical treatise *Traité de documentation*



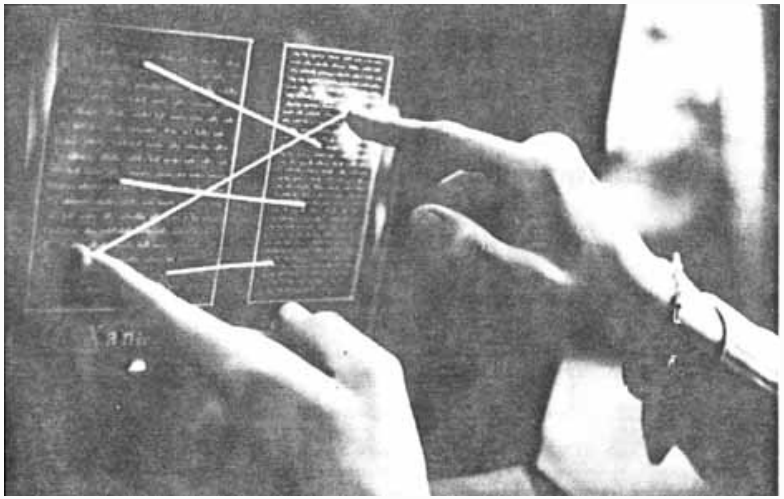
THE HISTORY OF HYPERTEXT

- **1945**: Vannevar Bush describes a hypothetical system called **Memex** in *As we May Think*

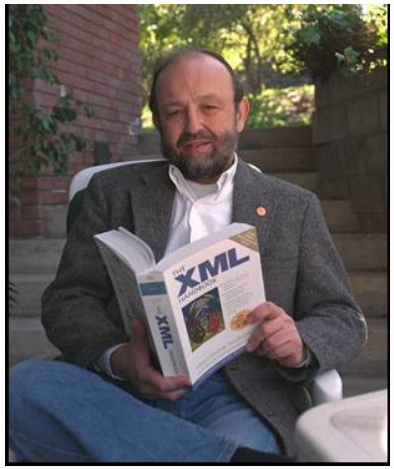
" ... The memex affords an immediate step, however, to associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select immediately and automatically another. This is the essential feature of the memex. The process of tying two items together is the important thing..."

THE HISTORY OF HYPERTEXT

- **1968**: Ted Nelson coins the term **Hypertext**



THE HISTORY OF MARKUP LANGUAGES



Charles Goldfarb:

- **1969**: Generalized Markup Language (**GML**) $\rightarrow$ InTIME
- **1974**: Standard Generalized Markup Language (**SGML**), ISO standard in 1986

" ... Markup should describe a document's structure and other attributes, rather than specify the processing to be performed on it, as descriptive markup need be done only once, and will suffice for future processing. Markup should be rigorous so that the techniques available for processing rigorously-defined objects like programs and data bases, can be used for processing documents as well. ... "

The approach taken by SGML (and HTML, XML)

Markup is delimited by angle brackets

- Consisting of elements formed by open tags and close tags ...
- which can have attributes

Example

```
<greeting> Hi</greeting>said<person ssn="123">Joe</person>
```

THE APPROACH TAKEN BY SGML (AND HTML, XML)

Markup is delimited by angle brackets

- Consisting of elements formed by open tags and close tags ...
- which can have attributes

Example

```
<greeting>Hi</greeting>said<person ssn="123">Joe</person>
```

open tag

THE APPROACH TAKEN BY SGML (AND HTML, XML)

Markup is delimited by angle brackets

- Consisting of elements formed by open tags and close tags ...
- which can have attributes

Example

```
<greeting>Hi</greeting>said<person ssn="123">Joe</person>
```

open tag close tag

THE APPROACH TAKEN BY SGML (AND HTML, XML)

Markup is delimited by angle brackets

- Consisting of elements formed by open tags and close tags ...
- which can have attributes

Example

```
<greeting> Hi</greeting>said<person ssn=”123”>Joe</person>
```

element

THE APPROACH TAKEN BY SGML (AND HTML, XML)

Markup is delimited by angle brackets

- Consisting of elements formed by open tags and close tags ...
- which can have attributes

Example

```
<greeting> Hi</greeting>said <person ssn="123">Joe</person>
```

open tag

THE APPROACH TAKEN BY SGML (AND HTML, XML)

Markup is delimited by angle brackets

- Consisting of elements formed by open tags and close tags ...
- which can have attributes

Example

```
<greeting> Hi</greeting>said<person ssn="123">Joe</person>
```

open tag close tag

THE APPROACH TAKEN BY SGML (AND HTML, XML)

Markup is delimited by angle brackets

- Consisting of elements formed by open tags and close tags ...
- which can have attributes

Example

```
<greeting> Hi</greeting>said<person ssn="123">Joe</person>
```

element

THE APPROACH TAKEN BY SGML (AND HTML, XML)

Markup is delimited by angle brackets

- Consisting of elements formed by open tags and close tags ...
- which can have attributes

Example

```
<greeting> Hi</greeting>said<person ssn="123">Joe</person>
```

attribute + value

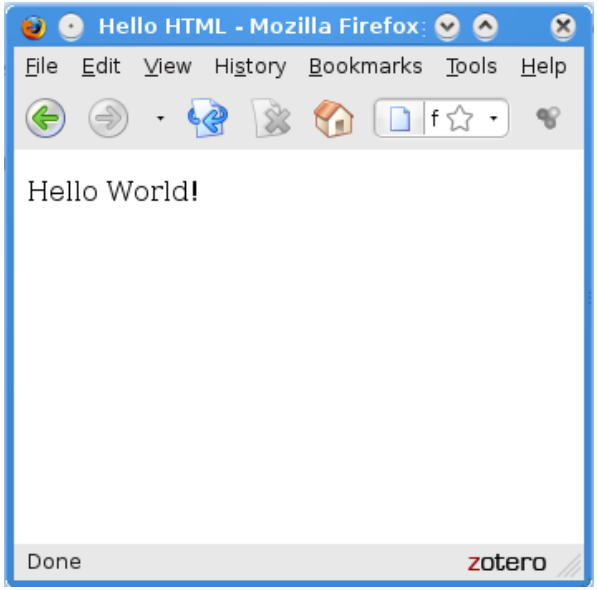
HTML

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

- At its core, HTML, is a simple markup language to describe the logical structure of a document

HTML

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```



- At its core, HTML, is a simple markup language to describe the logical structure of a document
- Browsers are free to interpret this structure
- Lots of different HTML versions

A CRASH COURSE ON HTML

By means of

Online

demonstration

LOGICAL VERSUS PHYSICAL

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

- At its core, HTML, is a simple markup language to describe the logical structure of a document
- Often one also want to specify the physical layout (bold, italic, font, size, ...)
- Doing this consistently within HTML itself gets messy ...

LOGICAL VERSUS PHYSICAL

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

- At its core, HTML, is a simple markup language to describe the logical structure of a document
- Often one also want to specify the physical layout (bold, italic, font, size, ...)
- Doing this consistently within HTML itself gets messy ...

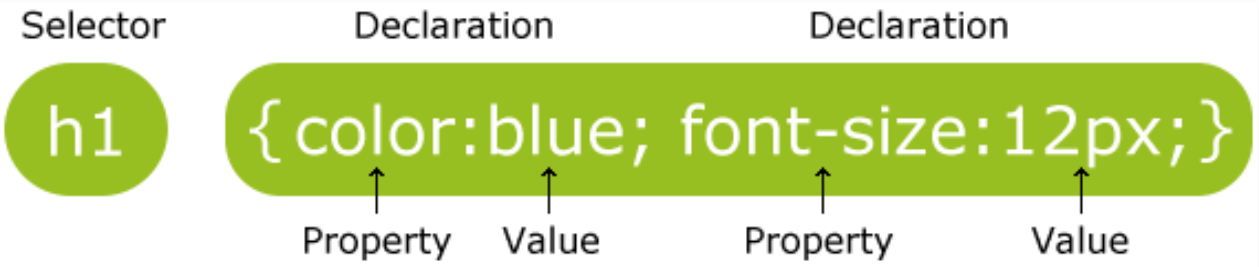
Definition

Cascading StyleSheets (CSS) separate structure from layout

A CRASH COURSE ON CSS

- Cascading Stylesheets separate structure from layout
- A CSS consists of a sequence of rules
- Each rule two parts: a selector and one or more declarations
- A declaration assigns a value to a property

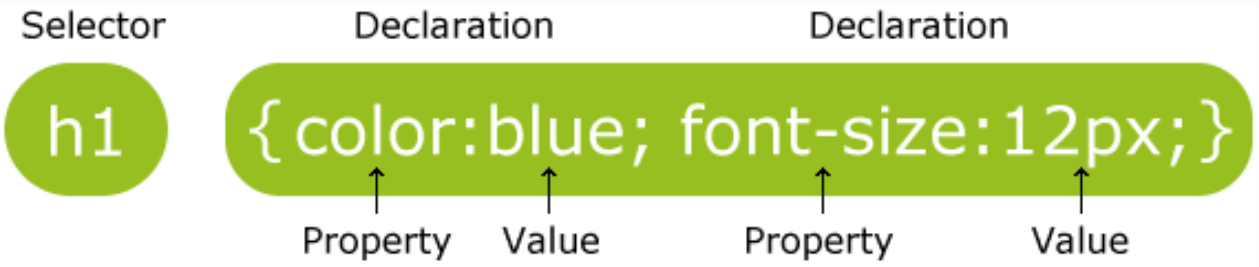
Example rule:



A CRASH COURSE ON CSS

- Cascading Stylesheets separate structure from layout
- A CSS consists of a sequence of rules
- Each rule two parts: a selector and one or more declarations
- A declaration assigns a value to a property

Example rule:



The *selector* is normally the name of the element you want to style:

- h1 → apply style to all `<h1>` elements *and their descendants*
- p → apply style to all `<p>` elements *and their descendants*
- table p → apply style to all `<p>` elements and their descendants *that occur within* a `<table>` element

A CRASH COURSE ON CSS (2)

By means of

Online demonstration

FROM HTML TO XML

HTML is limited:

- It is a presentation format that describes document structure only for purpose of presentation on a browser.
- As such, it has a fixed, limited, set of *tag names* (`<h1>`,`<p>`,`<a>`, ...)
- In many applications, we want to describe more structure

Consider that we want to describe recipes. Then we need:

1. Name of the recipe
2. Date created
3. Ingredients
4. Description of how to prepare

FROM HTML TO XML(2)

```
<h1>Rhubarb Cobbler</h1>
<h2>Wed, 4 Jun 95</h2>
This recipe is suggested by Jane Dow. Rhubarb Cobbler made with bananas
as the main sweetener. It was delicious.

  <table>
  <tr><td> 2 1/2 cups <td> diced rhubarb
  <tr><td> 2 tablespoons <td> sugar
  <tr><td> 2 <td> fairly ripe bananas
  <tr><td> 1/4 teaspoon <td> cinnamon
  <tr><td> dash of <td> nutmeg
  </table>

<i>Combine all and use as cobbler, pie, or crisp.</i>
<p>This recipe has 170 calories, 28% from fat,
58% from carbohydrates, and 14% from protein.
<p>Related recipes: <a href="#GardenQuiche">Garden Quiche</a>
```

Problems:

1. Not every HTML document is a valid recipe
2. Recipe programs need to infer recipe data from HTML presentation. What if we change the order of certain elements later?

FROM HTML TO XML(2)

```
<h1>Rhubarb Cobbler</h1>
<h2>Wed, 4 Jun 95</h2>
This recipe is suggested by Jane Dow. Rhubarb Cobbler made with bananas
as the main sweetener. It was delicious.

  <table>
  <tr><td> 2 1/2 cups <td> diced rhubarb
  <tr><td> 2 tablespoons <td> sugar
  <tr><td> 2 <td> fairly ripe bananas
  <tr><td> 1/4 teaspoon <td> cinnamon
  <tr><td> dash of <td> nutmeg
  </table>

<i>Combine all and use as cobbler, pie, or crisp.</i>
<p>This recipe has 170 calories, 28% from fat,
58% from carbohydrates, and 14% from protein.
<p>Related recipes: <a href="#GardenQuiche">Garden Quiche</a>
```

We need a special recipe markup language

FROM HTML TO XML (3)

HTML:

1. Is a presentation format, that describes structure only for purpose of presentation on a browser.
2. Fixed,limited set of tag names
3. Instance of SGML

XML:

1. Is a data exchange format, capable of describing arbitrary structure
2. Unlimited set of tag names
3. Simplification of SGML, which was difficult to parse

WHAT IS THE WORLD WIDE WEB?

- **1989**: At the CERN physics laboratory, Tim Berners-Lee designs a simple global hypermedia system, know known as the World Wide web

Three constituents (with their current meaning):

- URIs as a means for identifying & locating resources
- HTTP as a protocol for transmitting information over networks
- A myriad of data formats for describing information (resource representations):
  - HTML for display in a browser
  - XML as a data exchange format
  - JSON as a data exchange format, alternative to XML
  - RDF as a machine-interpretable data model

ESSENTIAL ONLINE RESOURCES

- Uniform Resource Identifier (URI): Generic Syntax – RFC 3986 (`http://tools.ietf.org/html/rfc3986`)
- Architecture of the World Wide Web, Volume One (`http://www.w3.org/TR/2004/REC-webarch-20041215/`)
- HTTP made really easy (`http://www.jmarshall.com/easy/http/`)
  Note that this document uses an outdated interpretation of “resources”

Questions?