Department of Computer Science
University of California, Los Angeles

Computer Science 144: Web Applications

Spring 2025
Prof. Ryan Rosario

Lecture 3: April 7, 2025

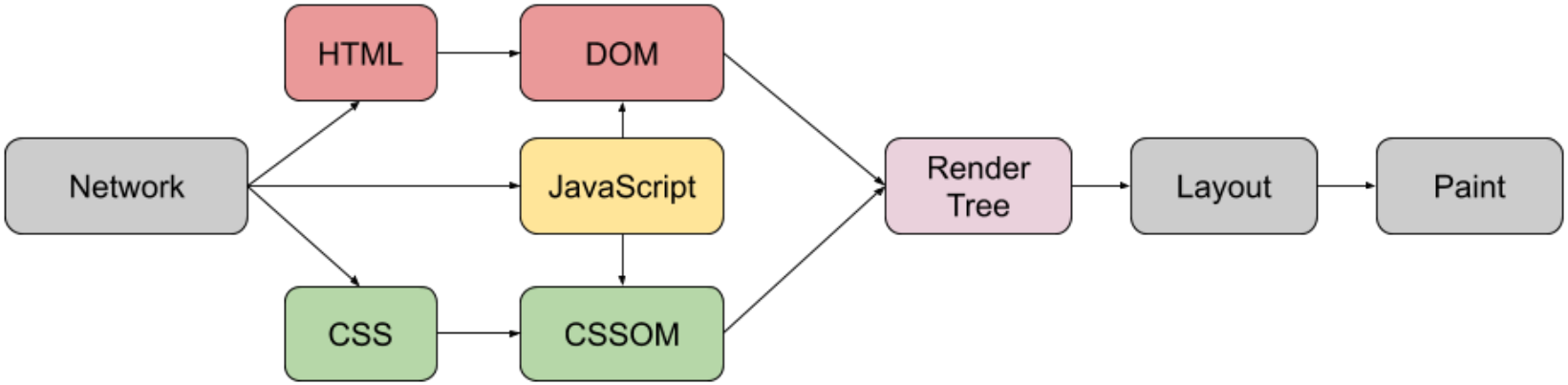
Outline

How the Browser Works

So far, this course has followed a natural progression based on how a browser actually works. We will go into some more detail on this, and then revisit when appropriate. The browser basically executes the following steps:

1. **Resolve**
2. **Connect**
   - **TCP Handshake**
   - **TLS Handshake**
   - **HTTP Request/Response**
3. **Parse**
4. **Render**

How the Browser Works (contd.)

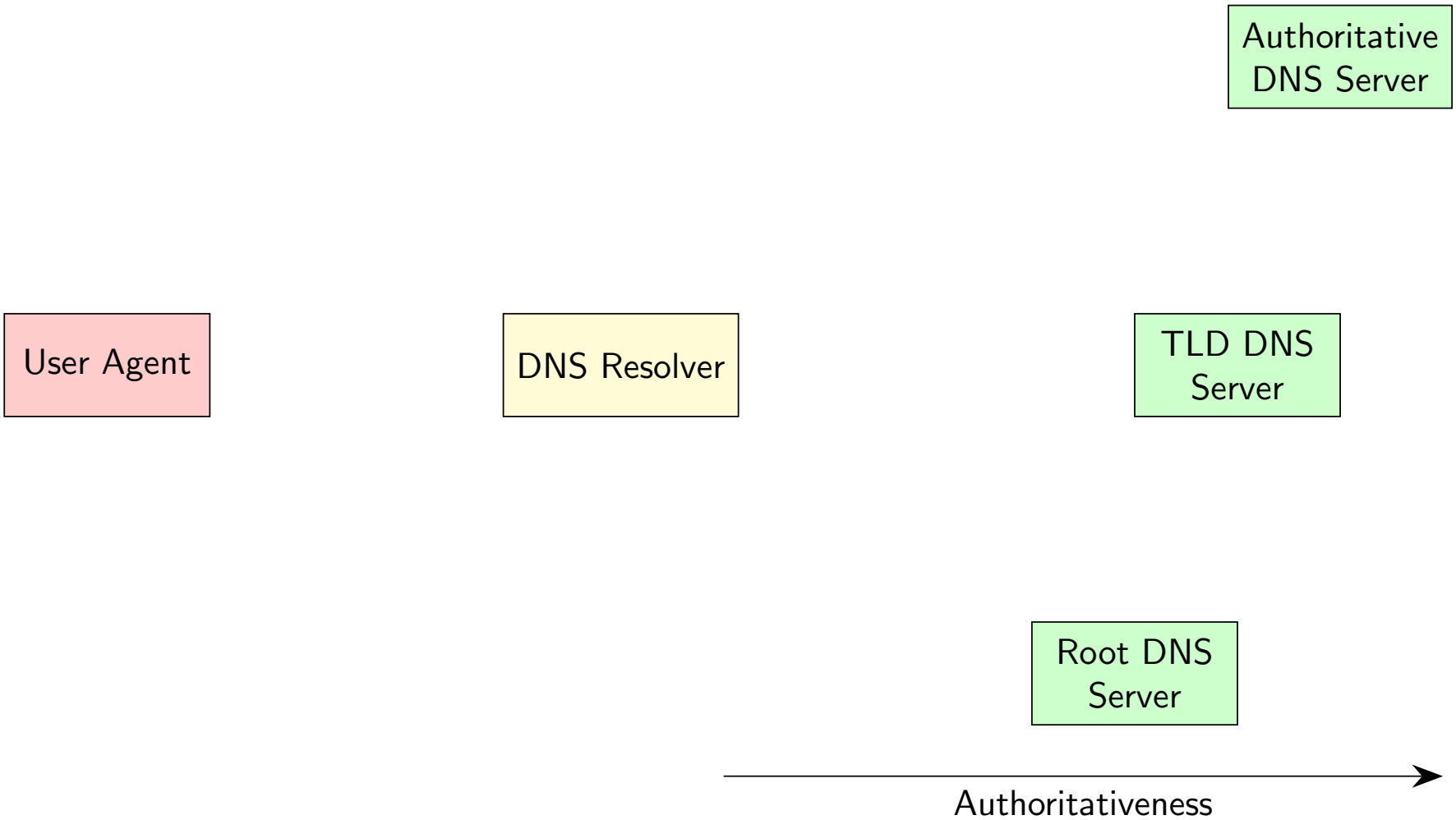


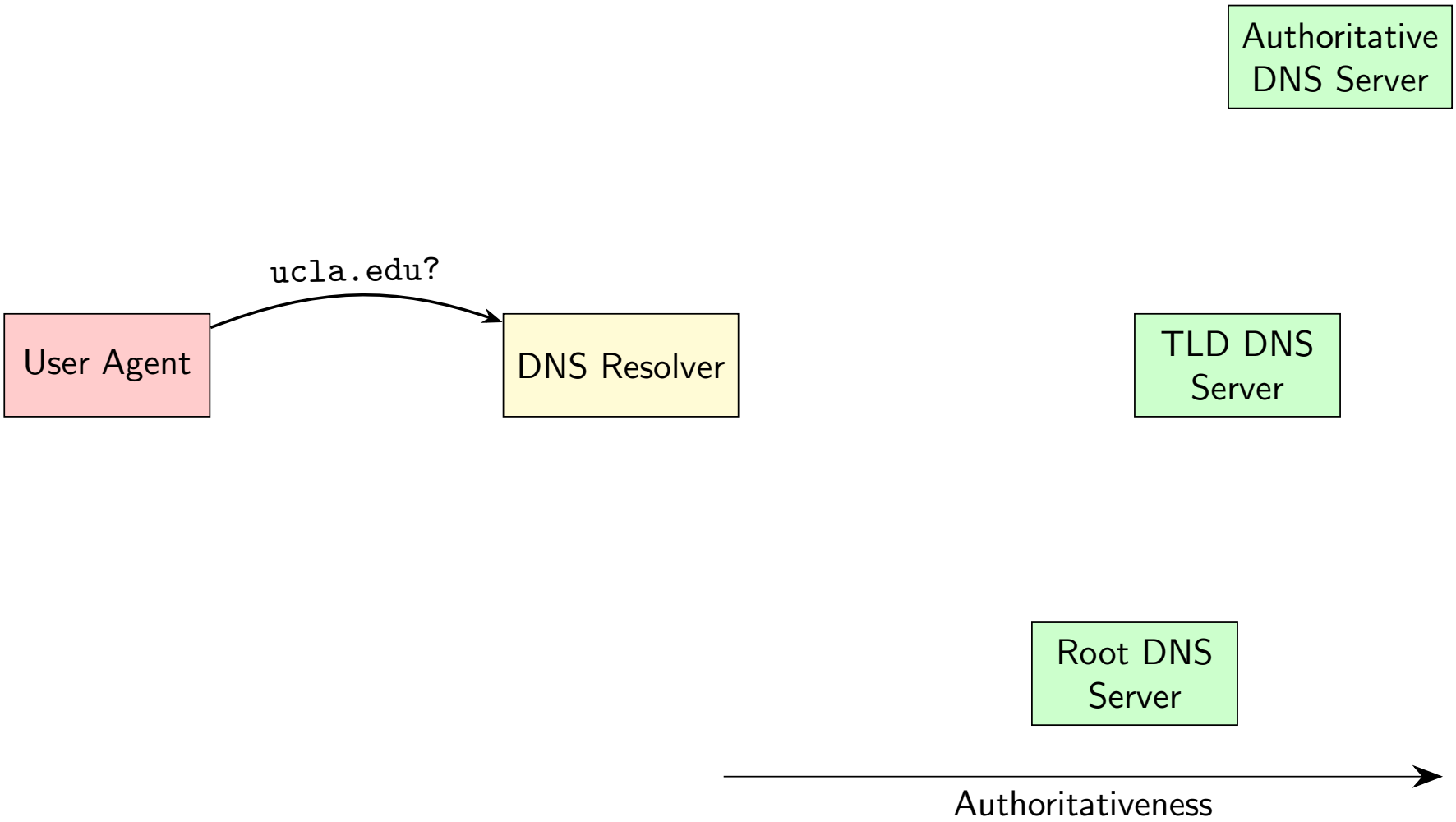
This can vary among WebKit (Safari) and Blink (Chrome, Edge, Opera, Brave). This example is for Gecko (Firefox).

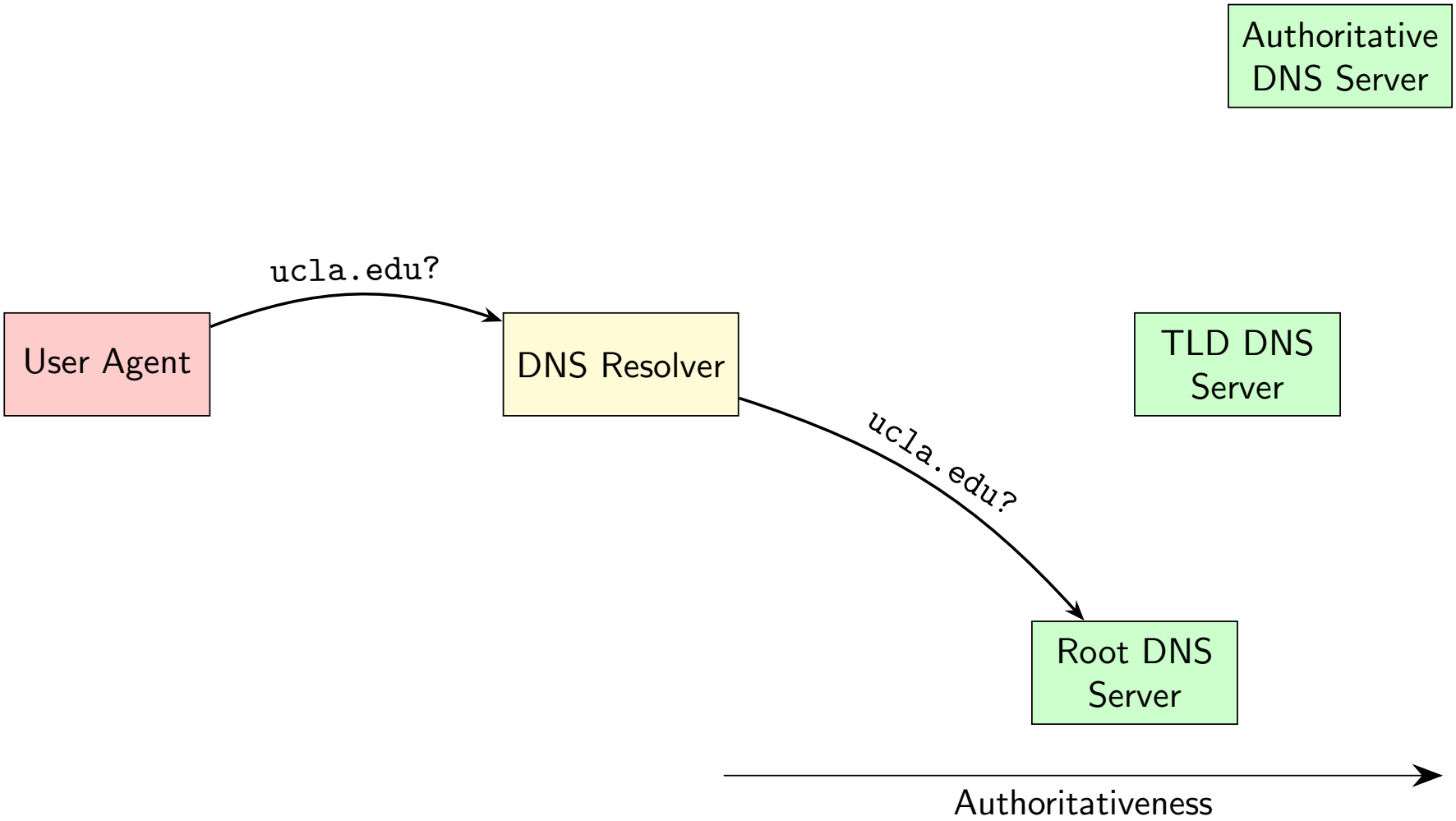
How the Browser Works: Resolve

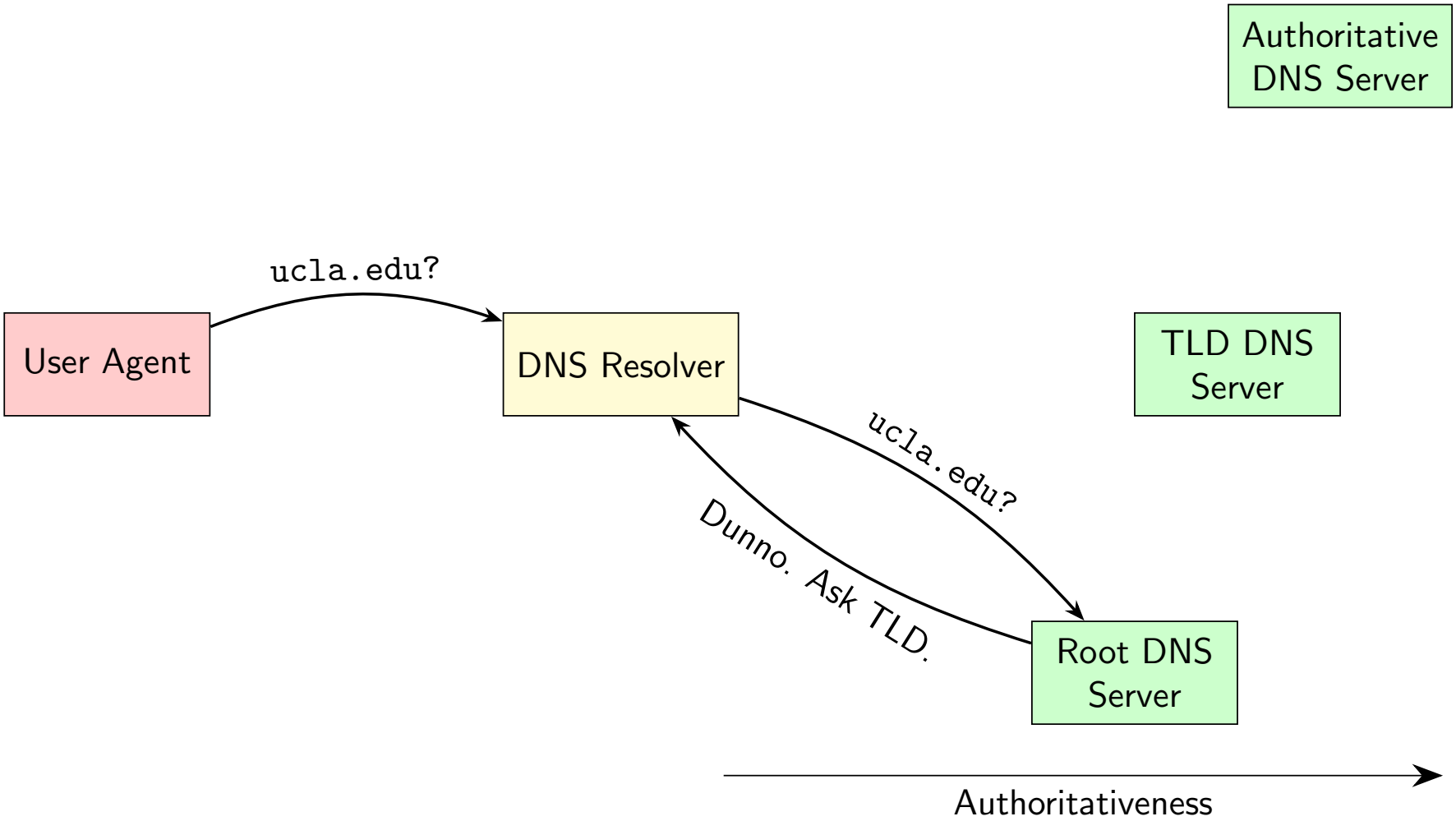
We have discussed DNS, but it was one particular variant that is not the default, so let's revisit it. Forwarded requests is used in enterprise settings but not typically on the Internet. The default configuration has two parts:

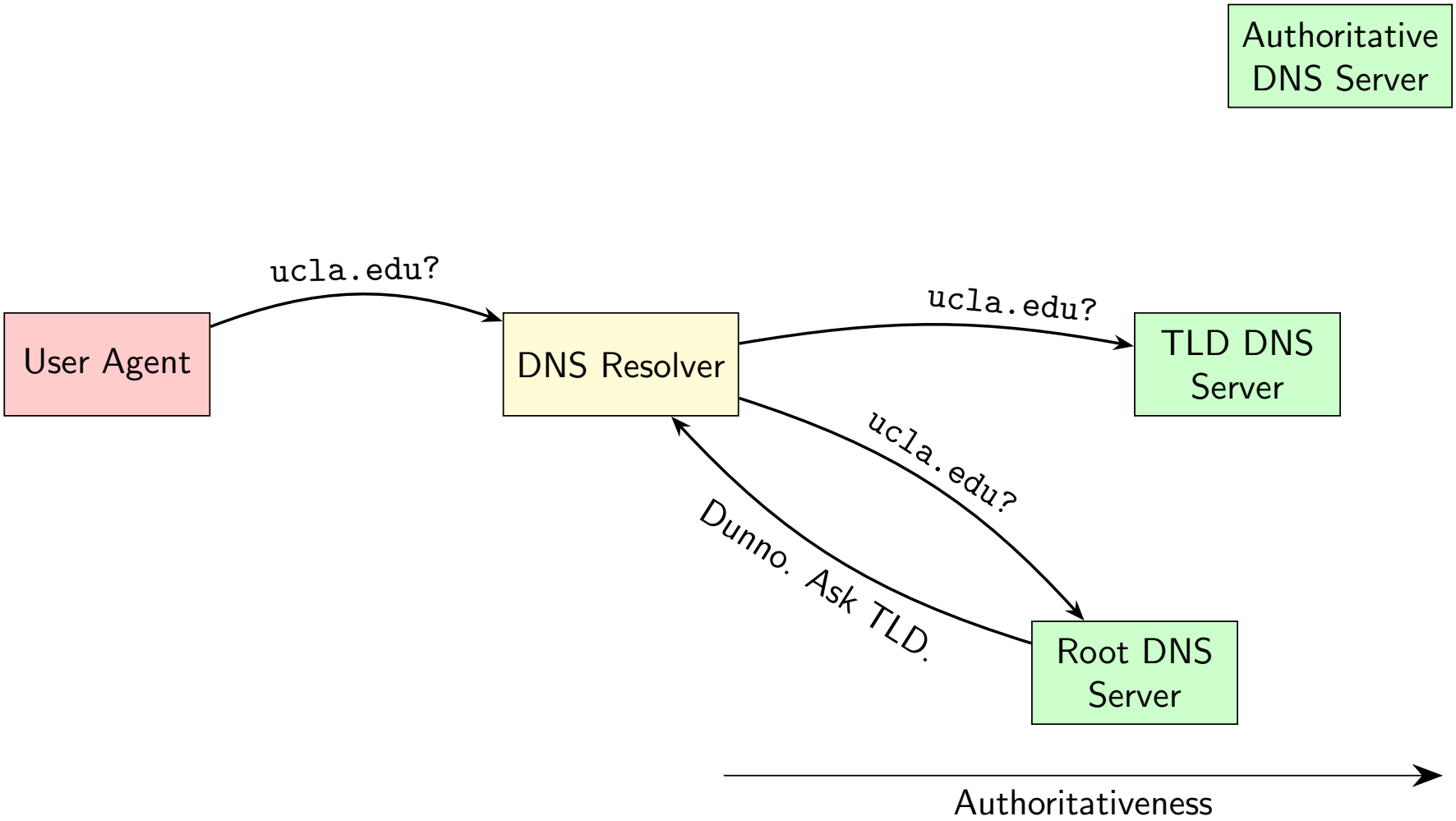
The user agent sends a request to a *recursive* DNS *resolver*. This resolver *iteratively* queries more authoritative DNS servers until it reaches a root or beyond.

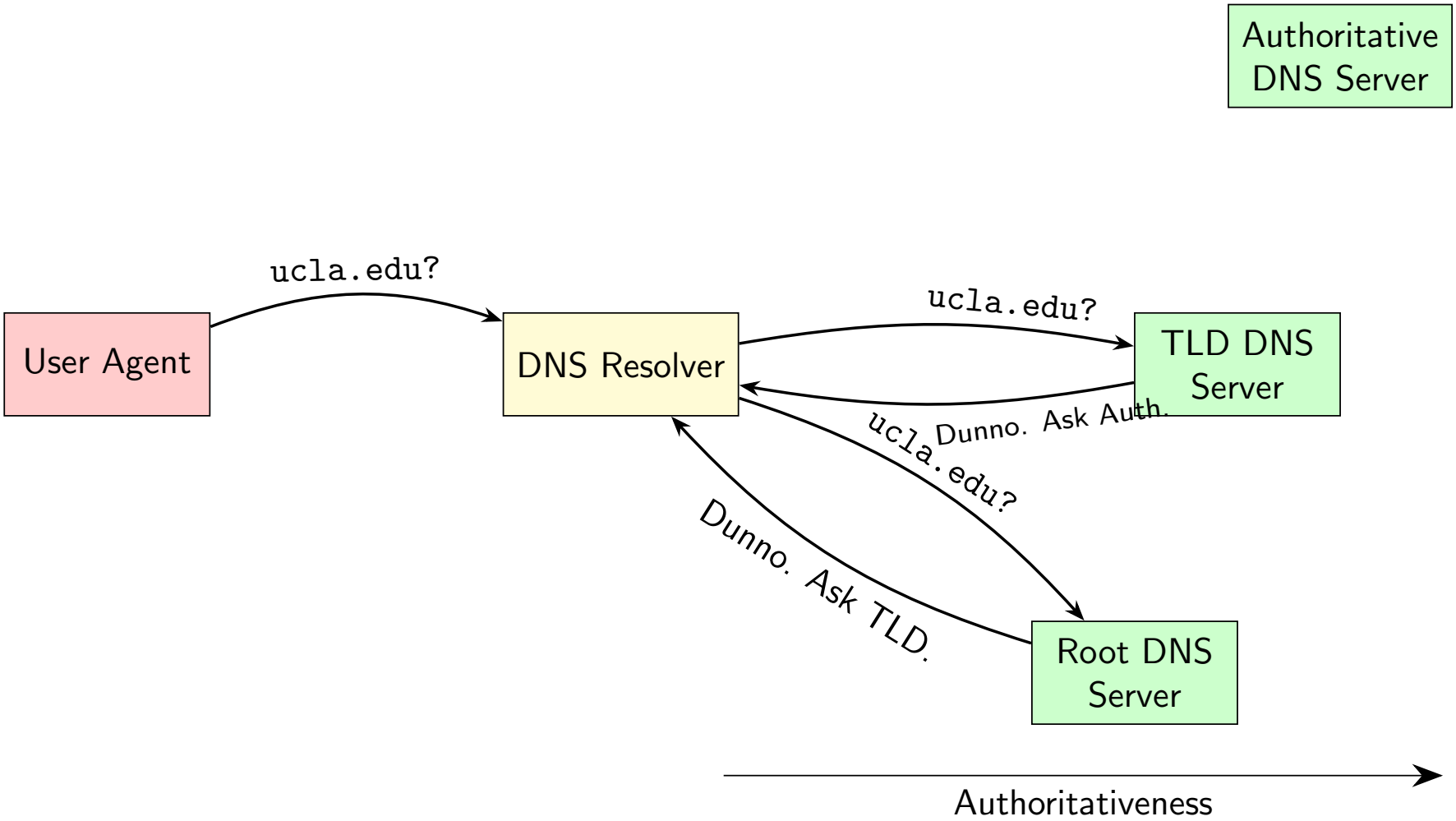

Authoritative DNS Server
User Agent
DNS Resolver
TLD DNS Server
Root DNS Server
Authoritativeness

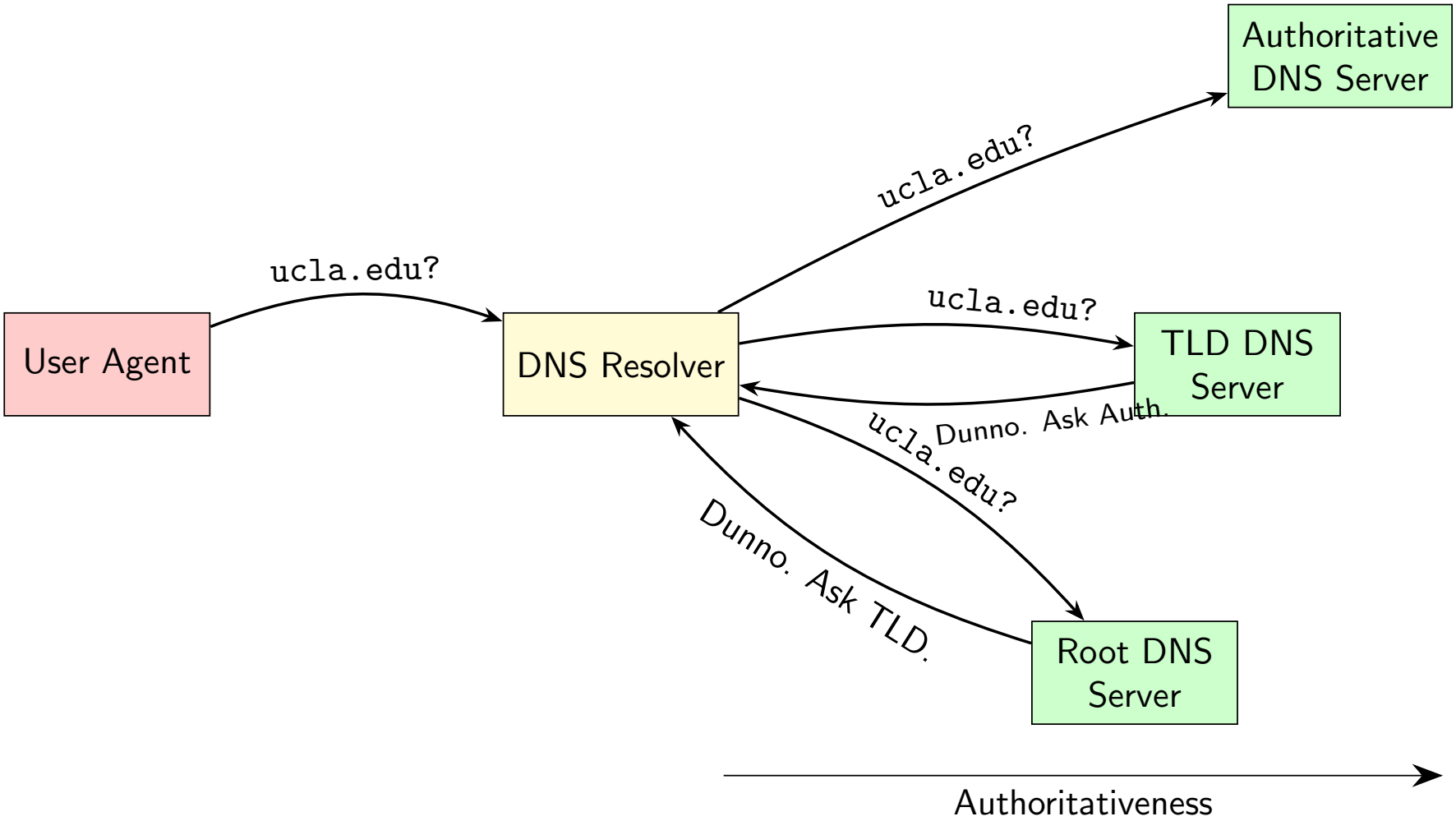

Authoritative DNS Server
ucla.edu?
User Agent
DNS Resolver
TLD DNS Server
Root DNS Server
Authoritativeness

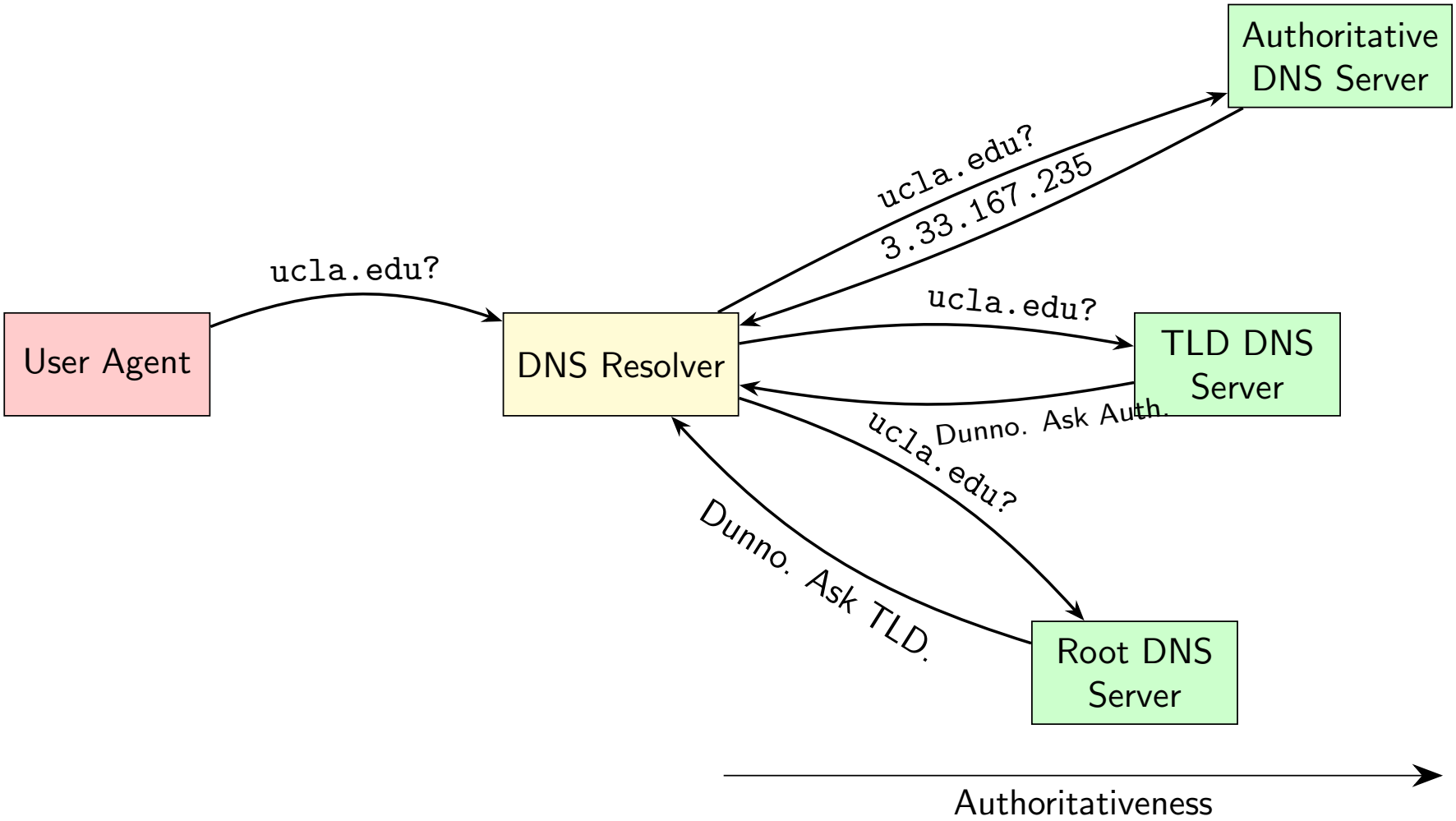

Authoritative DNS Server
ucla.edu?
User Agent
DNS Resolver
TLD DNS Server
ucla.edu?
Root DNS Server
Authoritativeness

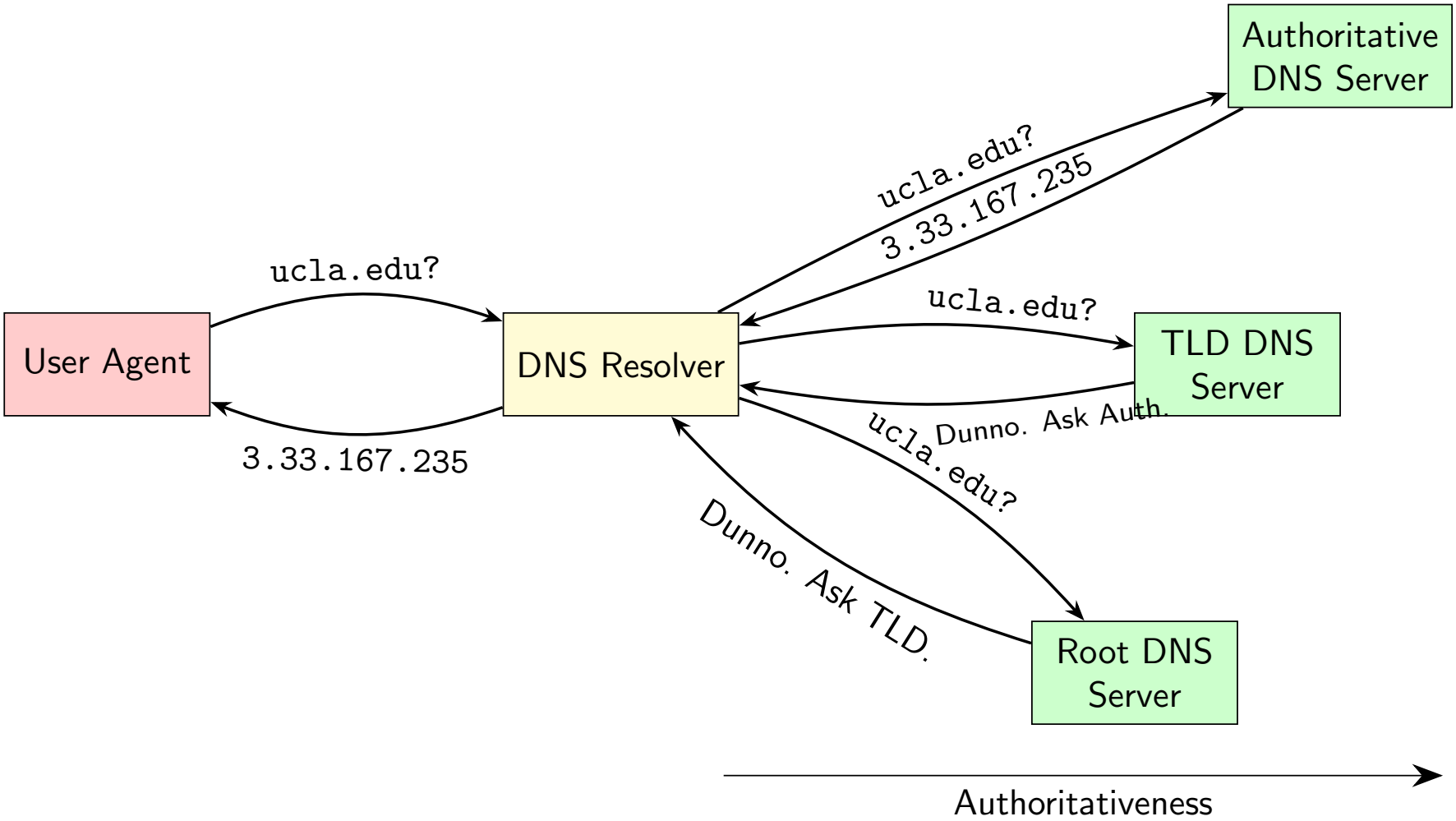

Authoritative DNS Server
ucla.edu?
User Agent
DNS Resolver
TLD DNS Server
ucla.edu?
Dunno. Ask TLD.
Root DNS Server
Authoritativeness


Authoritative DNS Server
ucla.edu?
User Agent
DNS Resolver
ucla.edu?
TLD DNS Server
ucla.edu?
Dunno. Ask TLD.
Root DNS Server
Authoritativeness


Authoritative DNS Server
ucla.edu?
User Agent
DNS Resolver
ucla.edu?
TLD DNS Server
Dunno. Ask Auth.
ucla.edu?
Dunno. Ask TLD.
Root DNS Server
Authoritativeness


Authoritative DNS Server
ucla.edu?
ucla.edu?
User Agent
DNS Resolver
ucla.edu?
TLD DNS Server
Dunno. Ask Auth.
ucla.edu?
Dunno. Ask TLD.
Root DNS Server
Authoritativeness


Authoritative DNS Server
ucla.edu?
3.33.167.235
ucla.edu?
User Agent
DNS Resolver
TLD DNS Server
Dunno. Ask Auth.
ucla.edu?
Dunno. Ask TLD.
Root DNS Server
Authoritativeness


Authoritative DNS Server
ucla.edu?
3.33.167.235
ucla.edu?
User Agent
DNS Resolver
TLD DNS Server
3.33.167.235
Dunno. Ask Auth.
ucla.edu?
Dunno. Ask TLD.
Root DNS Server
Authoritativeness

How the Browser Works: Resolve

One DNS request may be sent to retrieve the HTML page (if not already cached).

Then one lookup for each unique hostname referenced in the HTML page (e.g. for images, CSS).

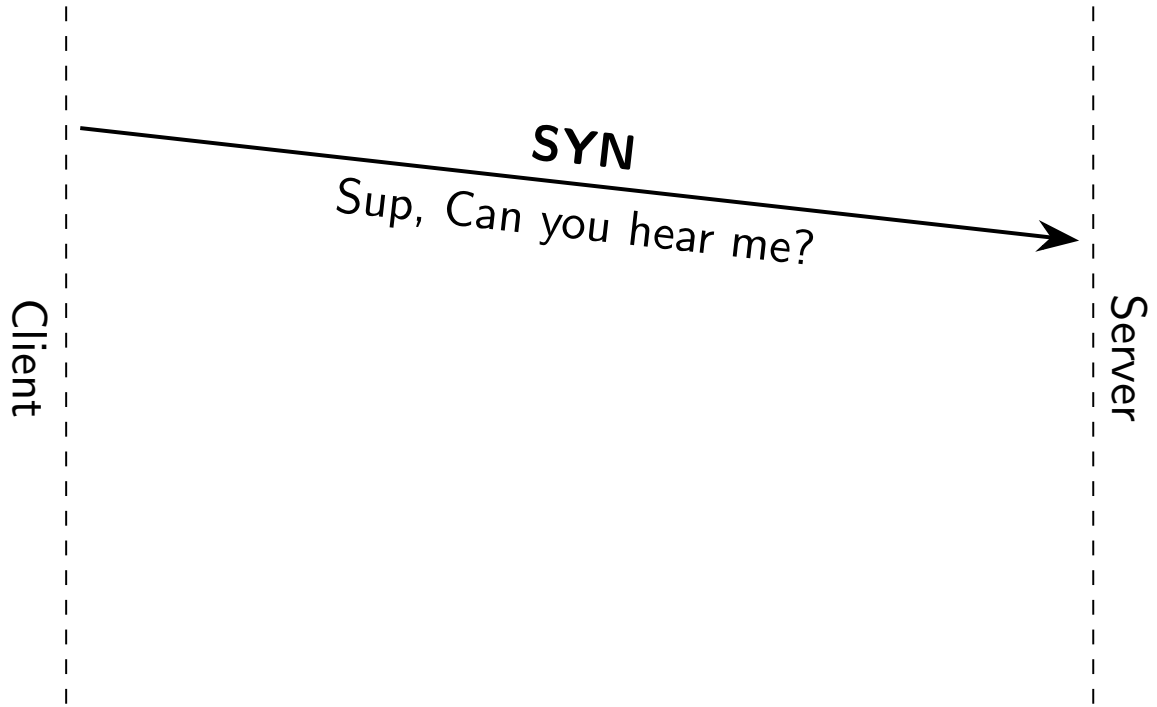
The browser caches the response. **This can be very non-performant over mobile.**

How the Browser Works: Connect (TCP)

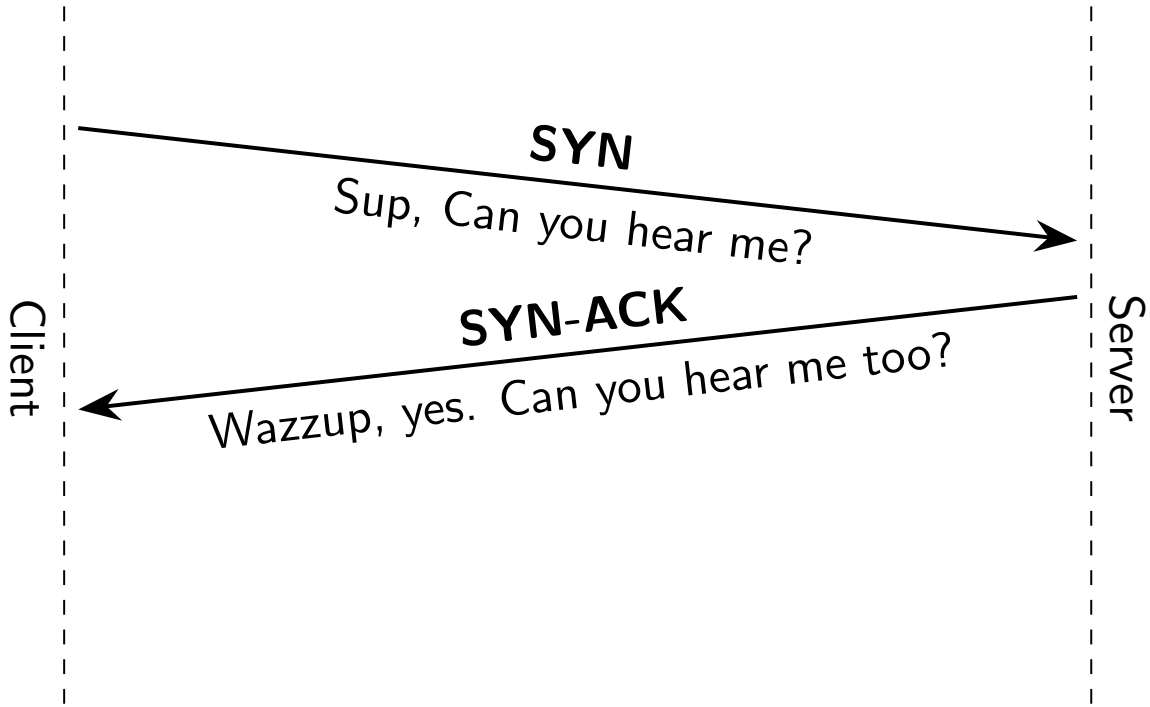
Now that we have the IP address, the browser opens a TCP connection to the server using the 3-way handshake.

How the Browser Works: Connect

How the Browser Works: Connect



How the Browser Works: Connect



How the Browser Works: Connect

Client

SYN

Sup, Can you hear me?

SYN-ACK

Wazzup, yes. Can you hear me too?

ACK

Yes, sweet bro. Let's vibe!

Server

How the Browser Works: Connect (TLS)

For secure connections. The TLS handshake:

1. determines which cipher will be used to encrypt the communication
2. verifies the server
3. establishes that a secure connection is in place

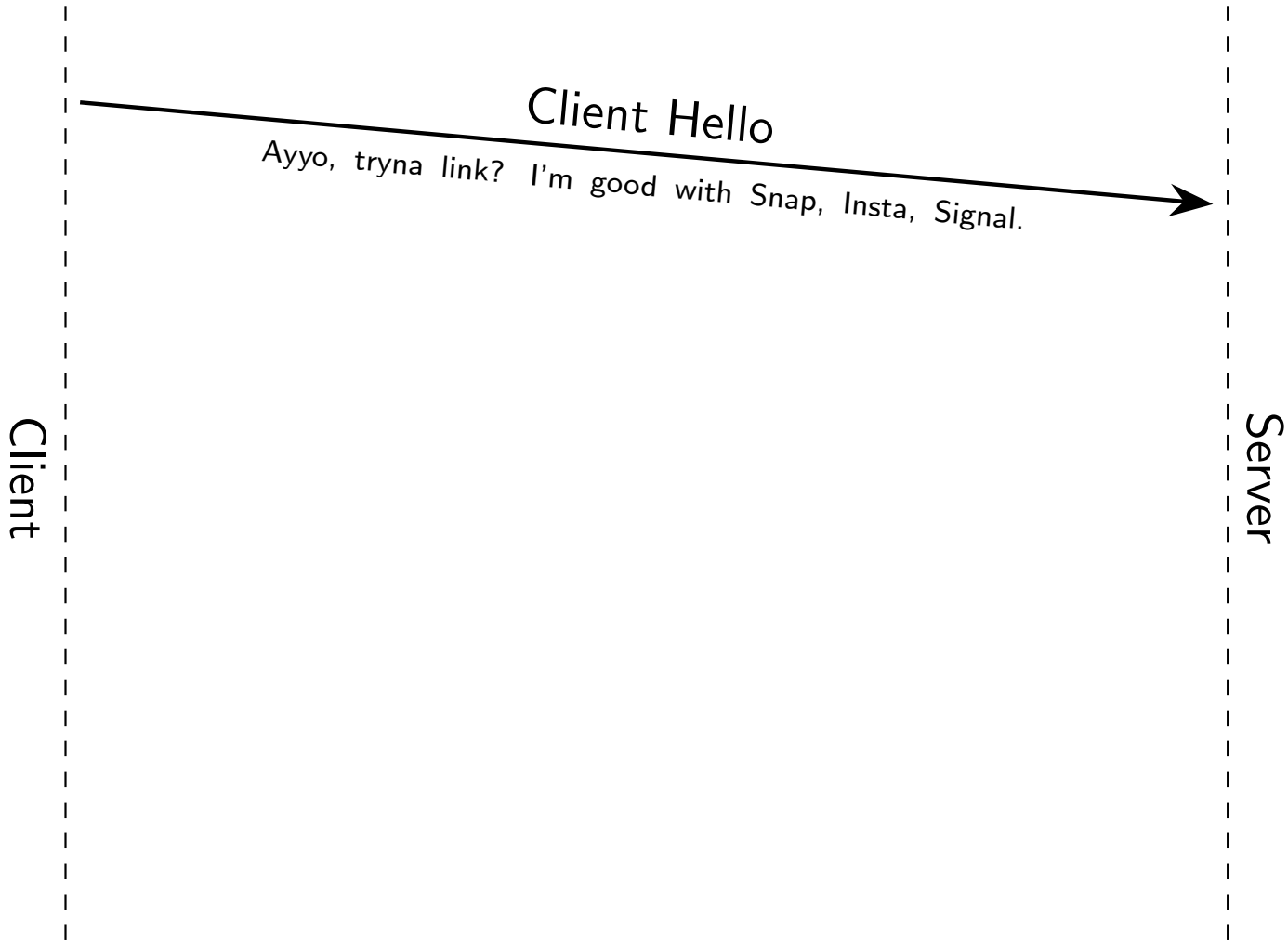
This extra time is worth the safety we get.

How the Browser Works: Connect (TLS) (contd.)

Introducing...

Server

Client


Client Hello
Ayyo, tryna link? I'm good with Snap, Insta, Signal.
Client
Server

Client Hello

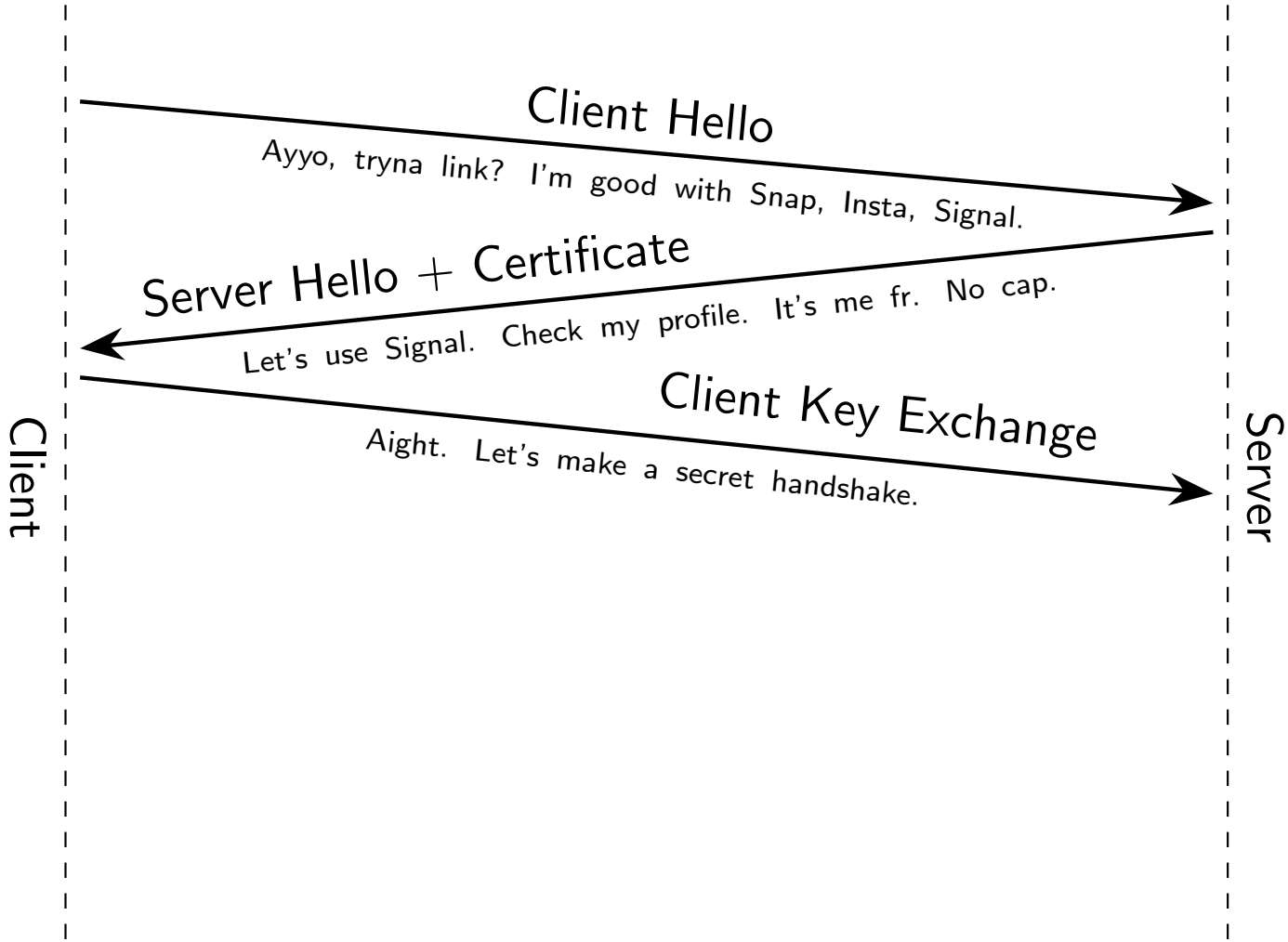
Ayyo, tryna link? I'm good with Snap, Insta, Signal.

Server Hello + Certificate

Let's use Signal. Check my profile. It's me fr. No cap.

Client

Server


Client Hello
Ayyo, tryna link? I'm good with Snap, Insta, Signal.
Server Hello + Certificate
Let's use Signal. Check my profile. It's me fr. No cap.
Client Key Exchange
Aight. Let's make a secret handshake.
Client
Server

Client

Server

Client Hello

Ayyo, tryna link? I'm good with Snap, Insta, Signal.

Server Hello + Certificate

Let's use Signal. Check my profile. It's me fr. No cap.

Client Key Exchange

Aight. Let's make a secret handshake.

Server Key Exchange

Handshake's firm. I'm down.



Client

Client Hello

Ayyo, tryna link? I'm good with Snap, Insta, Signal.

Server Hello + Certificate

Let's use Signal. Check my profile. It's me fr. No cap.

Client Key Exchange

Aight. Let's make a secret handshake.

Server Key Exchange

Handshake's firm. I'm down.

Client Finished

Vibes secured. Here's that gym split in our handshake.

Server





Client

Client Hello

Ayyo, tryna link? I'm good with Snap, Insta, Signal.

Server Hello + Certificate

Let's use Signal. Check my profile. It's me fr. No cap.

Client Key Exchange

Aight. Let's make a secret handshake.

Server Key Exchange

Handshake's firm. I'm down.

Client Finished

Vibes secured. Here's that gym split in our handshake.

Server Finished

Got it! Here's the workout plan back in our code. We good!

Server



HTTP, TLS and HTTPS

We will discuss security later in the quarter.

HTTPS is basically HTTP-over-TLS.

How the Browser Works: Wrapping up Connect

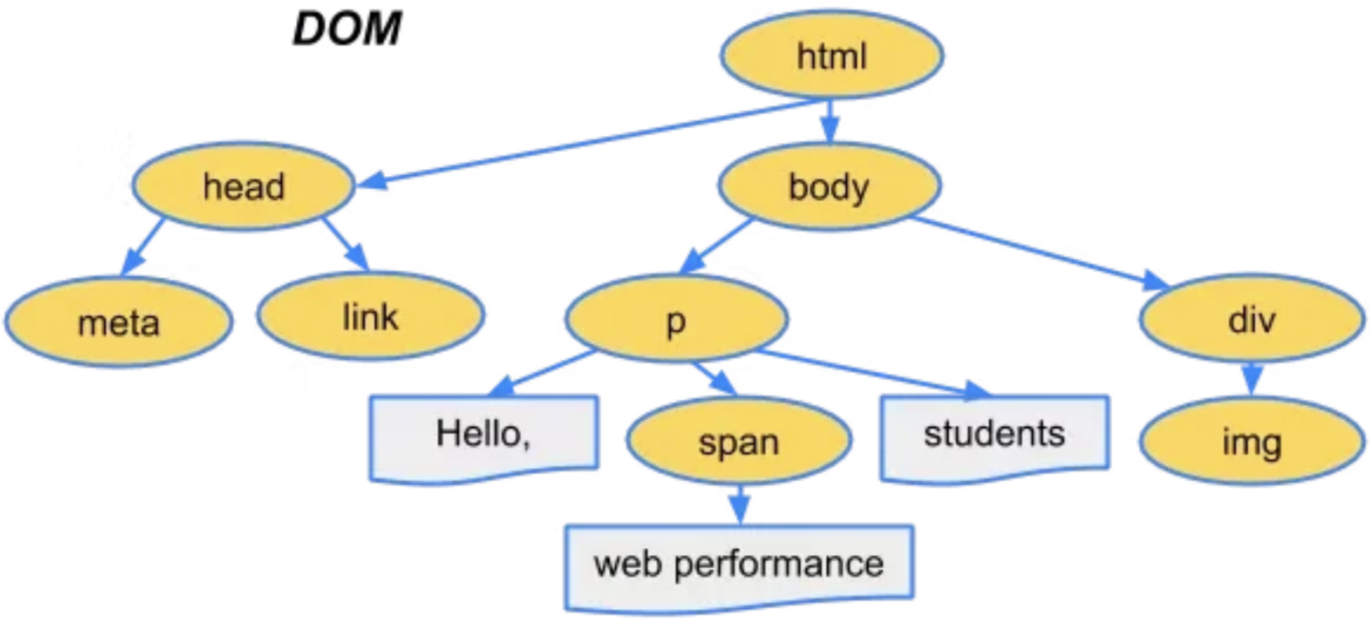
Once the TCP, and optionally TLS, handshakes are complete, the browser can send an HTTP request to the server and receive a response.

The HTTP request and response are split and organized into packets. The packets must be received in order and each packet must be ACKed.

What does the browser do with the HTTP response?

How the Browser Works: Parse

The browser processes any important headers, and converts the HTML into a Document Object Model (DOM).



How the Browser Works: Parse (contd.)

During parsing, if the browser encounters **non-blocking resources** like images or stylesheets, it will fire off a request to get them. The browser doesn't wait.

Scripts actually block rendering so we know what the DOM should look like as a result.

If it does not, the browser may have to go back and modify the DOM after execution as a result. **Race condition.**

How the Browser Works: Parse (contd.)

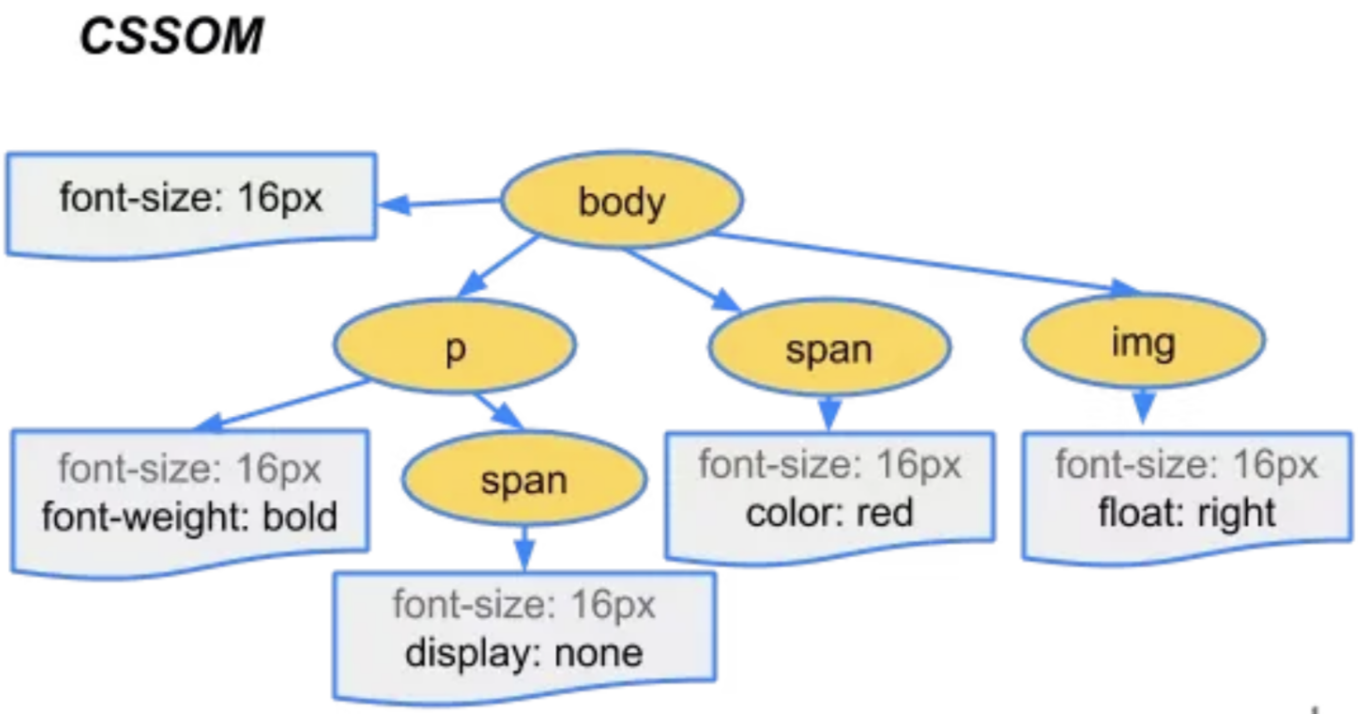
The Preload Scanner

While the browser grows the DOM tree, the preload scanner separately (in parallel) parses the content and looks for high priority resources and fetches them.

This way they are hopefully ready when the parser reaches them.
Waiting for CSS blocks Javascript.

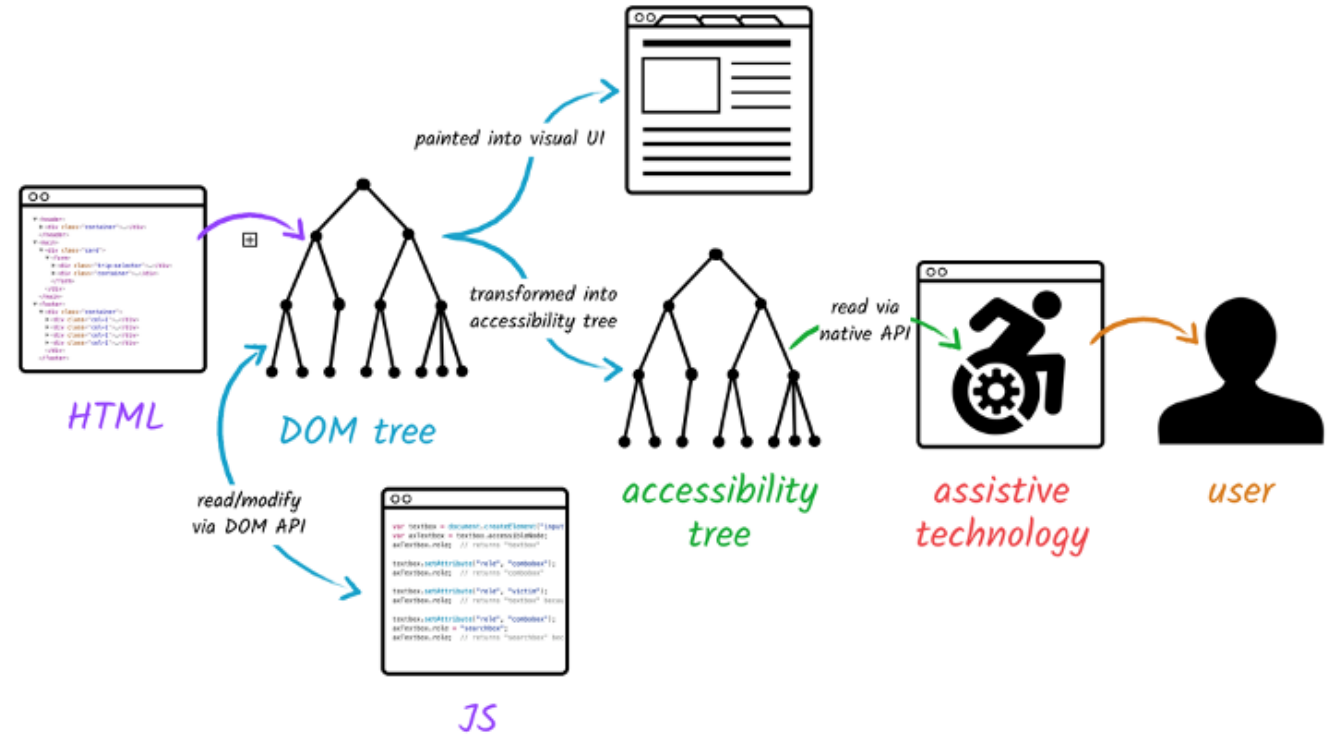
How the Browser Works: Parse (contd.)

Next, CSS is parsed into a CSSOM, very similar to a DOM.



How the Browser Works: Parse (contd.)

A tree similar to the DOM is grown to provide information to accessibility tools.



How the Browser Works: Parse (contd.)

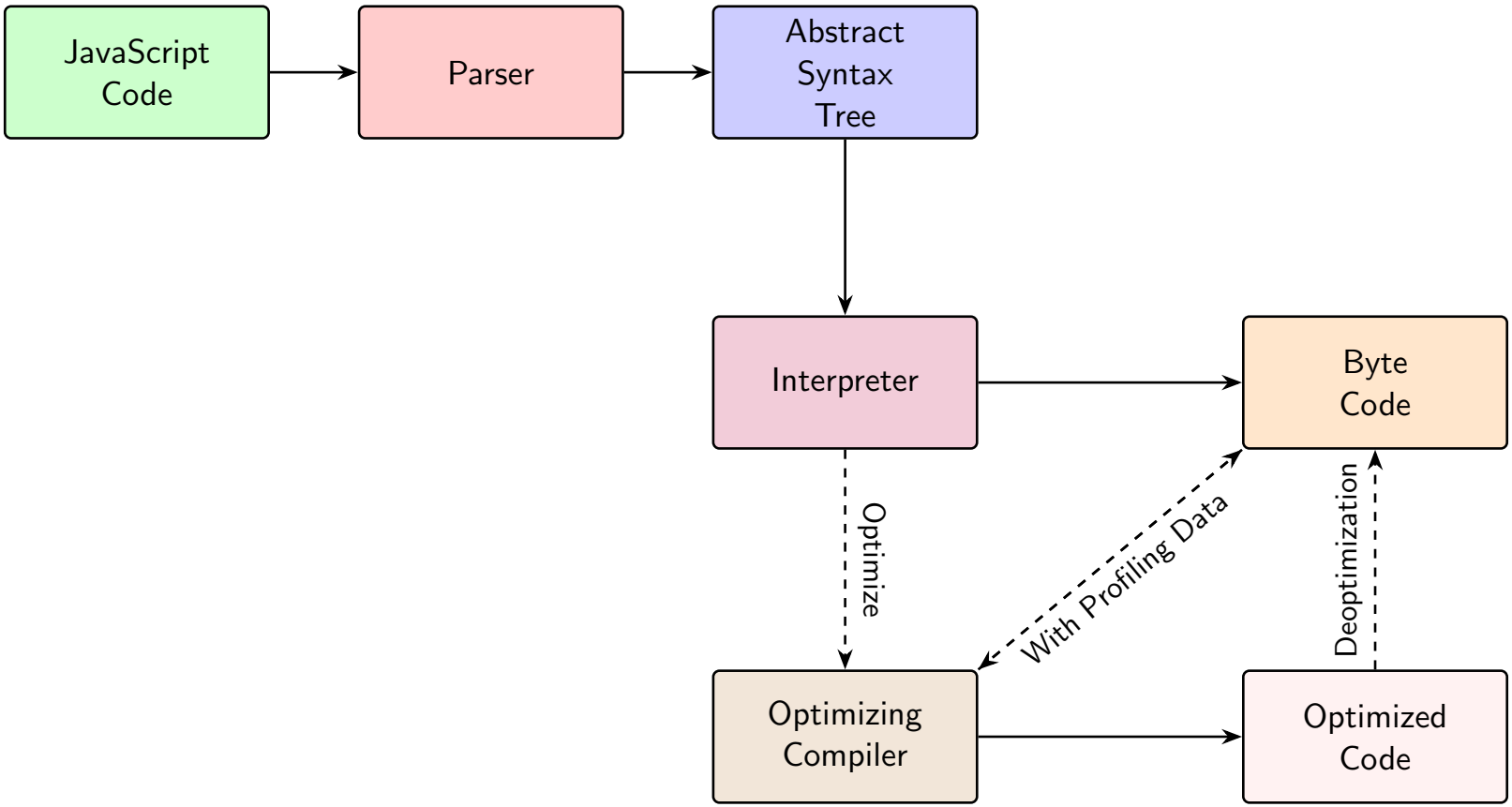
While the CSS is being parsed, Javascript is downloaded by the preload scanner.

It is parsed into an Abstract Syntax Tree (AST), compiled (JIT) and interpreted.

Some browser engines take these ASTs and pass them to a compiler that outputs bytecode (Javascript compilation).

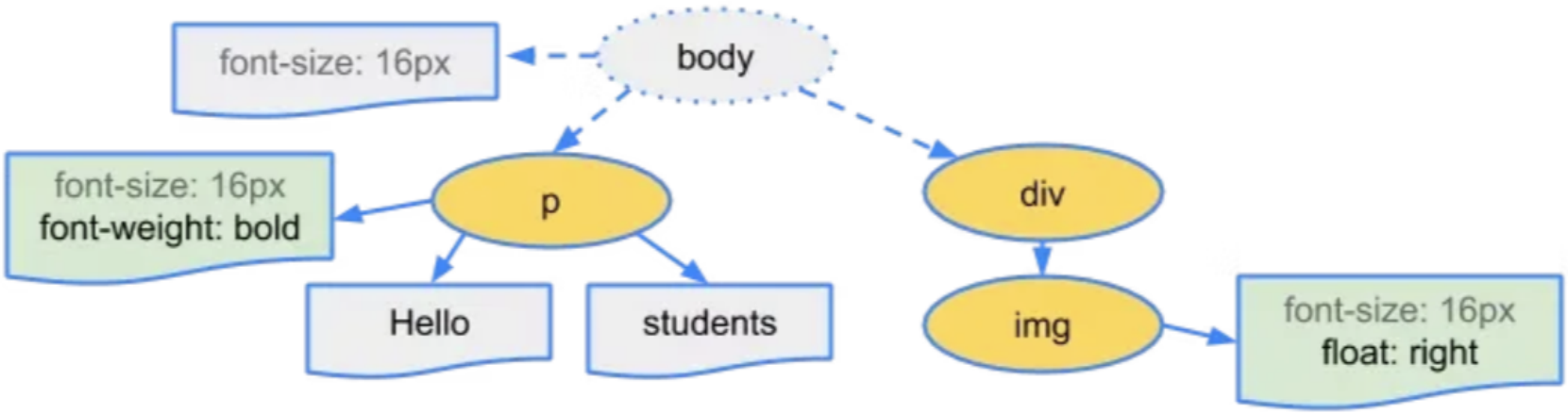
The code is then interpreted on the main thread (unless using web workers).

How the Browser Works: Parse (contd.)



How the Browser Works: Render

Both the DOM and CSSOM are used to create a **Render Tree** which determines the layout of the page.



How the Browser Works: Render (contd.)

There are four steps in rendering. The browser:

1. **Style** - determines the style of each object on the page. This is done by traversing the DOM and CSSOM trees.
2. **Layout** - determines the size and position of each object on the page. This is done by traversing the DOM and CSSOM trees.
3. **Paint** - paints the pixels to the screen. This is done by traversing the render tree and drawing each object.
4. **Composite** - combines the painted layers into a single image. This is done by the GPU.
5. **Reflow / Repaint / Recompose** - repeats when changes are made to the DOM, CSSOM, or a resource becomes available.

How the Browser Works: Render (contd.)

The browser makes an interesting **optimization**.

Style, reflow and paint must take at most 16.67ms (Mozilla) to ensure smooth scrolling.

Sometimes the drawing is broken down into layers in parallel, and then merged together in the **Composite** step.

Introduction

Cascading Styles Sheets (CSS) are used to *style* and *format* content within HTML pages.

CSS is deceptively complex. My main reference for this course is *2,000 pages.* The basics are straightforward, but additional study and practice is needed to truly master it.

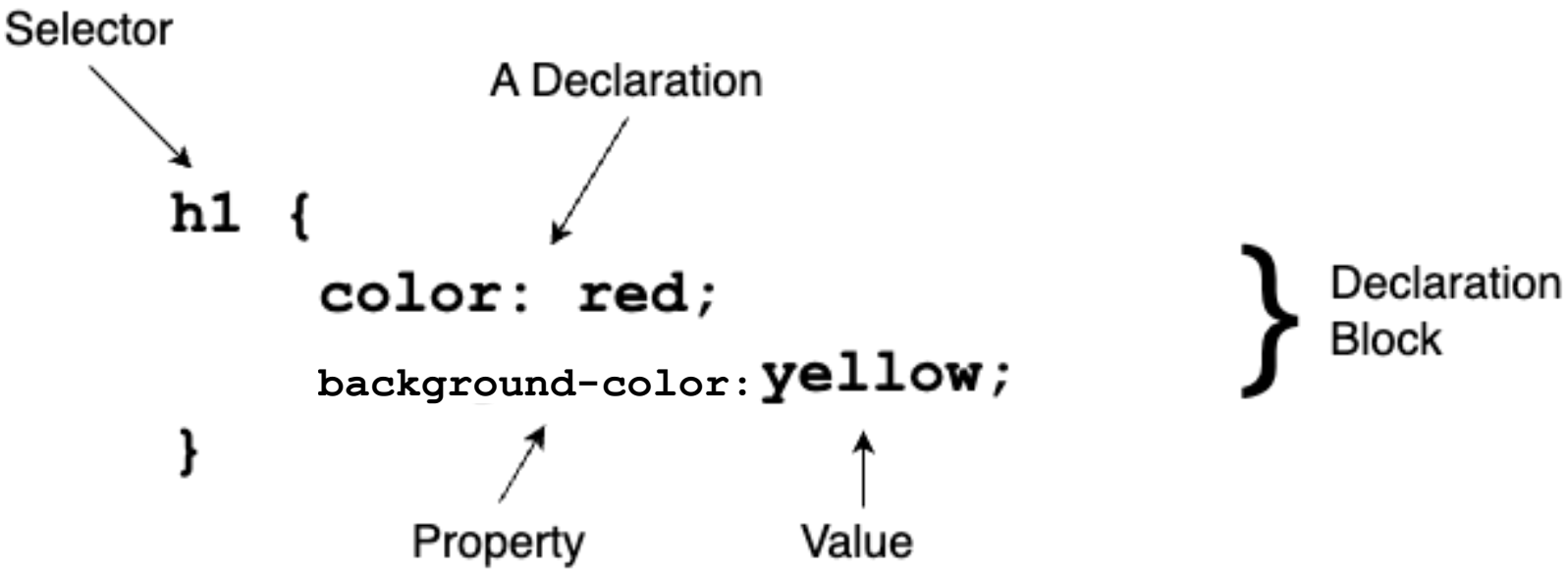
Introduction (contd.)

Today we will introduce some important concepts and notations in CSS. Instead of thoroughly covering all syntax we will learn some of the more important styles by osmosis and provide resources.

Next time we will dig into layouts, media queries and responsive web design.

Anatomy of a Rule

CSS rules contain a **selector** and a series of **declarations** each containing a **property** and **value**.



CSS specifies a series of rules on **tags**, **classes**, **IDs** and **atributes**.

Type Selectors

Type selectors are sometimes called tag or element selectors and they target specific HTML tags.

```
body { color: black; }
p { color: gray; }
h1 { color: silver; }
```

Styles are **inherited** by subelements nested *within* the element as well.

Class Selectors

We can specify one or more classes on an HTML element. For CSS, the class specifies what styles should be applied to the relevant elements.

```
.warning { font-style: italic; }

p.warning { font-weight: bold; }

span.warning { font-style: italic; color: red; }
```

The rules `p.warning`, `span.warning` are more *specific* than the `.warning` rule, but it **inherits** from the class. We can also override properties in `p.warning` or `span.warning` if we want.

Class Selectors (contd.)

HTML elements can have *more than one class*. The class names are separated by whitespace:

```
<p class="urgent warning">
    When handling plutonium, care must be taken to avoid the
    formation of a critical mass.
</p>
<p>With plutonium, <span class="warning">the possibility of
    implosion is very real, and must be avoided at all costs
    </span>. This can be accomplished by keeping the various
    masses separate.</p>
```

The order of the class names **does not matter**.

Class Selectors (contd.)

We can apply styles to both classes independently, and to the combination.

```
.warning { font-weight: bold; }
.urgent { font-style: italic; }
.warning.urgent { background-color: silver; }
```

The order of `warning` and `urgent`. `.warning.urgent` selects elements with **both** classes, regardless of nesting.

Class Selectors (contd.)

Checkpoint:

```
p.warning.help { background-color: red; }
```

What does this mean? Will it apply to this element:

```
<p class="urgent warning help">Help me Rhonda!</p>
```

ID Selectors

Remember that there should be one element with each ID value in an HTML page. This is actually an HTML standard thing, CSS does not care. We use the **octothorpe**, #

```
#first-para {
    font-family: Arial, Helvetica, sans-serif;
    font-variant: small-caps;
    font-weight: 500; /* a little darker than normal */
}

/* Equivalent to */
#first-para { font: small-caps 500 Arial, sans-serif; }
```

By the way, there are more text options in the CSS Fonts module.

Attribute Selectors

We can also target elements that have specific attributes as well.

```css
/* Highlight all img elements that have alt text. For debugging */
img[alt] { outline: 3px solid forestgreen; }

/* All links that have both an href and title (tooltip) attribute */
a[href][title] { font-weight: bold; }

/* All links that lead to https://cs.ucla.edu homepage */
a[href="https://cs.ucla.edu"] {
    color: blue; background-color: yellow;
}

/* All links that do not lead to https://cs.ucla.edu */
a:not([href="https://cs.ucla.edu"]) {
    color: red; background-color: gold;
}
```

Attribute Selectors (contd.)

Caution!

What do you think this targets?

```
p[class="urgent warning"] { color: red; }
```

Probably not what you think.

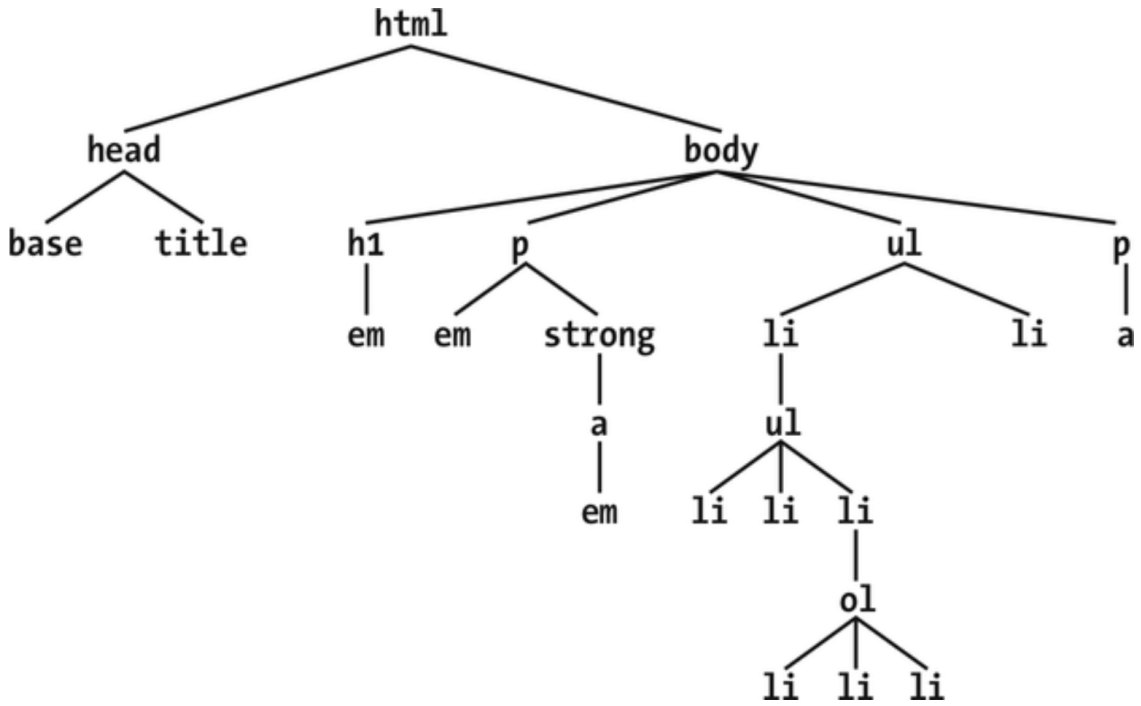
Attribute Selectors (contd.)

We can also target partial matches on attributes. It's here for your reference.

Type	Description
`[foo~="bar"]`	Atributes containing the full word `bar` as a whitespace delimited token.
`[foo*="bar"]`	Attributes that contain `bar` anywhere in the value.
`[foo^="bar"]`	Attributes with values that start with `bar`.
`[foo$="bar"]`	Attributes with values that end with `bar`.
`[foo|="bar"]`	Attributes with values that start with `bar` followed by a hyphen, or are exactly `bar`.

Structure: Descendant Selectors

Below is an example of a document tree structure for a simple web page:



Structure: Descendant Selectors (contd.)

We can combine selectors:

```
/* Target em elements that are descendants of h1 element. Doesn't
mean directly following.*/
h1 em { color: grey; }

/* Similarly */
ul ol ul em { color: grey; }

/* Target strong elements descending from blockquote OR strong
elements descending from p elements */
blockquote strong, p strong { color: green; }

/* Target span elements that descend from elements with class help. */
.help span { color: red; }
```

Structure: Child Selectors

```css
/* Target strong elements that are DIRECT children,
not descendants, of an h1 element. */
h1 > strong { color: red; }

/* Set the top margin to 0, for any p element that is a SIBLING
appearing IMMEDIATELY after an h1 element. */
h1 + p { margin-top: 0; }

/* Target all ol elements that appear some time after an h2 element,
and both share a parent. */
h2 ~ ol { font-style: italic; }
```

Question: What would `li + li` do?

Structure: Child Selectors (contd.)

Fun for the whole family! A few more:

```
/* Every even row in a table should have green background,
like a ledger. */
tr:nth-child(even) { background-color: green; }
tr:nth-child(2n) { background-color: green; }

/* Color the 2nd row blue. */
tr:nth-child(2) { background-color: blue; }
```

Question: What do you think `div :nth-child(2)` targets?

Structure: Child Selectors (contd.)

Text	**Description**
space	Some kind of descendent.
no space	and
,	or
>	child
+	next sibling
~	any later sibling
`:nth-child(n)`	nth child

Other Selectors

```
/* Pseudoclasses select based on the STATE of an
element. */
a:link:hover { font-weight: bold; }
a:visited { color: red; }
a:visited:hover { font-weight: bold; }
/* parent has an image child */
:has(> img) { background-color: red; }

/* Pseudoelement styles -- mainly decorative
   These do not target specific syntax but have
   other meaning */
p::first-letter { font-size: 2em; }
```

For more information on pseudoclasses, see here.

Where CSS Lives

The most standards compliant:

```
<html>
    <head>
        <link rel="stylesheet" href="mystyle.css" title="Default">
    </head>
</html>
```

Where CSS Lives (contd.)

As seen earlier, you can also specify the CSS in the `<style>` within the `<head>` section, **but you should not do this.**

```
<html>
    <head>
        <style>
        h1 { color: red; }
        </style>
    </head>
</html>
```

Note that other CSS stylesheets can be imported using `@import(path)`.

Where CSS Lives (contd.)

Style can also be specified inline. This should only be done if you need to apply a one-time style to a single element.

```
<div class="myclass">
    This is a very <span style="color: red; font-weight: bold;">
        important</span> message.
</div>
```

A Note on Inheritance

Descendant elements inherit style from their parents, unless overridden in the style.

Most box model styles (`background-color`, `border`, `margin`, `padding`) are not automatically inherited.

Can inherit with: `background-color: inherit;`

`margin` is a tricky little beast.

A Note on Inheritance (contd.)

Relative sizings like `2em` are relative to the size of the parent. Suppose the root is 16px.

```
body { font-size: 16px; }
body > div { font-size: 2em; }
body > div > div { font-size: 3em; }
body > div > div > div { font-size: 0.5em; }
/* What is the size? */

body > div { font-size: 0.75rem; }
/* What is the size? */
```

We can escape this compounding by using `rem` which is relative to the root element.

Conflicting Styles

There will be times when styles conflict. This is example of an origin conflict:

```
<html>
    <head>
        <link rel="stylesheet" href="mystyle.css">
            <style>
                .box { border-color: red; }
            </style>
    </head>
</html>
```

mystyle.css

```
.box { border-color: green; }
```

Which one wins?

Conflicting Styles (contd.)

It can also happen in stylesheets. . .

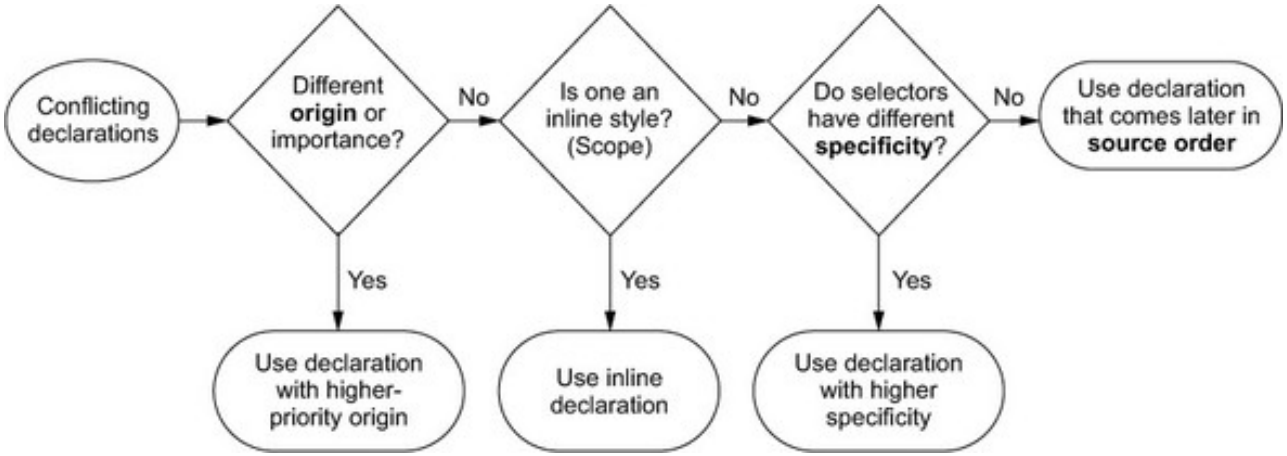
Remember our radioactive example?

The Cascade

The **cascade** in Cascading Style Sheets refers to the cascading nature of styles, but also the method used to resolve style conflicts.

The Cascade (contd.)

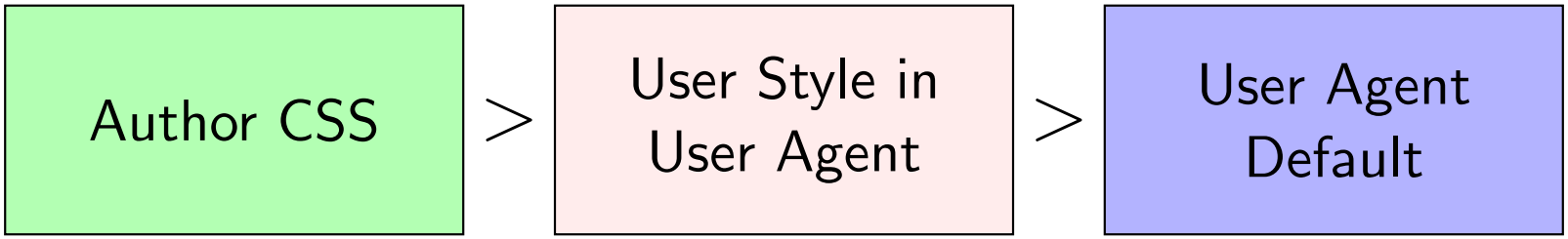
The cascade considers several aspects of the styles.



1. the origin of the stylesheet
2. where the declaration is defined in the user's code
3. specificity
4. whichever declaration is made **last** in the style.

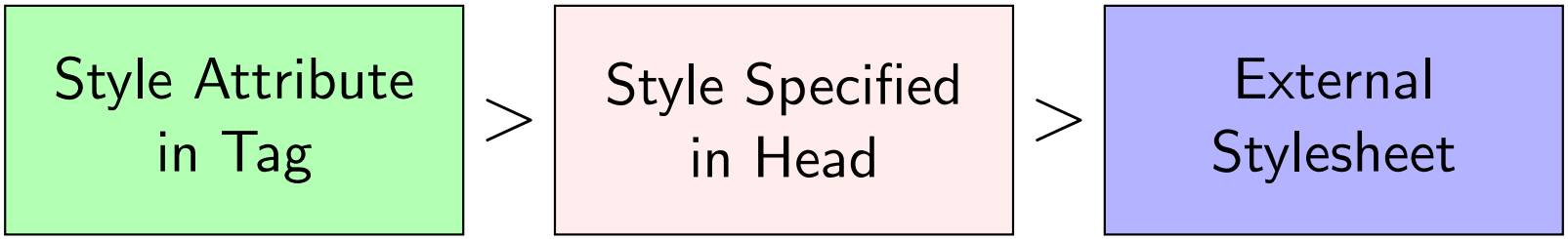
The Cascade: Origin

The first rule of the cascade is where the styles are defined. We typically focus on author stylesheets, but styles can also come from the user or user agent (browser). The priority from highest to lowest:



The Cascade: Scope

Standards typically dictate the use of external (linked) stylesheets. Though they actually have the lowest priority in a conflict. The priority:



The Cascade: Specificity

Basically:

1. A inline styles win
2. B if there is a tie, the selector with more IDs wins
3. C if there is a tie, the selector with more classes, attribute selectors, pseudo-classes wins
4. D if there is still a tie, the selector with the most tags and pseudo-elements wins
5. If there is still a tie, the one declared last in the style wins

The Cascade: Specificity (contd.)

When looking at selectors, we can compute a vector for each rule. The winner is based on lexicographic order:

`(A, B, C, D)`

For example, `body header.page-header h1` yields `(0,0,1,3)` if we assume the style is not inline.

+, ~, <, * are not counted.

The Cascade: Specificity (contd.)

More examples:

Selector	Inline?	IDs	Classes	Tags	Vector
`html body header h1`	0	0	0	4	`(0,0,0,4)`
`body header.page-header h1`	0	0	1	3	`(0,0,1,3)`
`.page-header .title`	0	0	2	0	`(0,0,2,0)`
`#page-title`	0	1	0	0	`(0,1,0,0)`

The highest vector, lexicographically, wins.

The Cascade: Specificity (contd.)

Some exercises for you:

Selector	Inline?	IDs	Classes	Tags	Vector
`h1`					
`p em`					
`.grade`					
`p.bright em.dark`					
`#id216`					
`div#sidecar *[href]`					

The Cascade: Specificity (contd.)

A Snake in the Grass: Suppose we change the following CSS:

```
h1 { font-weight: normal; }
.highlight { font-weight: bold; }
```

Into this:

```
h1.highlight { font-weight: bolder; }
```

The Cascade: Specificity (contd.)

Or...

```
header nav ul li a.active.current-page {
  color: tomato;
}

/* and */
a.current-page {
  color: green;
}
```

Question: What could we do instead?

The Cascade: Specificity (contd.)

This is `!important`

One "code smell" in CSS is to use `!important` to override the cascade. **It's lazy, it's dangerous!**

```
p { color: red !important; }
```

overrides everything except another `!important` with higher specificity. It's a front-of-the-line pass.

It's like using `sudo` for everything.

The Cascade is More Complicated

Our discussion of the cascade is sufficient for now.

It gets more complicated and more information can be found here.

Learning More

We will come back to CSS later in the quarter, particularly for layout and responsive web design.

The best way to master CSS is via practice:

- CSS Zen Garden
- CSS Battle
- Front End Mentor

Conclusion

This lecture covered CSS Style which is just part of the CSS landscape.

Next time we will discuss CSS Layout, responsive web design and CSS graphics.