Hello, I'm fantasai.
I work on the CSS specifications at W3C.

How many people here work with CSS?
How many know what it is, but don't use it?
Anyone not know what CSS is?
Raise your hand if you have heard of W3C.
Keep your hand up if you know what we do. :)
Raise your hand if you have no idea what W3C does.
Cool.

So my day job is working on the CSS specifications.
What this means is that I work with guys from Apple, Microsoft,
Mozilla, Opera, Adobe, HP, Google, and other companies
to design and describe new features for CSS,
and, less glamorously, to fix errors and ambiguities
in the specifications for existing CSS features.

The specs we write at W3C are what define a correct CSS implementation:
they represent an agreement among the borwser vendors as to what,
exactly,
CSS is and how it behaves.

First, to understand CSS layout and the mess it is today, we're going to
go back in time...
—
all the way to the late 1990s, the Age of Geocities
—
Ok, well, not /all/ web pages looked that horrible. :)
But this was the dark ages of Web Design, when H1 meant "bigger font",
when adding background color required table tags, and when the most
sophisticated layout tool designers had was HTML tables. With transparent
spacer GIFs inside.
e
—

CSS was created to replace all the formatting markup
– all the FONT tags and BR tags –
with a simple style sheet language.
Using CSS, you could control the style of an entire website
with a few lines of code in a single file,
and that, was totally radical.

—

CSS Level 1 didn't have any real layout tools.
It had width and height properties, which replaced their namesake HTML
attributes;
and it had floats, which replaced the 'align' attribute on IMG and TABLE.
This was not intended–nor, at that time, used–for coarse layout but for, well,
floats.
Y'know, with text wrapping around them.
Tables were still the accepted way of handling 2D layout; there was
nothing else.
And of course CSS margins and padding provided a much-needed relief
from spacer
gifs and BR tags. These things seem very simple to us now, but back then
they
were revolutionary. :)

With CSS Level 2, which was released in 1998, W3C brought
the Web's state-of-the-art layout tool to CSS:
you could turn any collection of HTML elements into a table display.
Together with other values of the 'display' property, CSS2 promised
the ability to dissociate the layout you desired from the tags you used,
freeing up authors to use more appropriate semantic markup.
Navigation lists could be displayed as tab navigation,
and equal-height columns became a one-liner in CSS:

```
.sidebar, .main { display: table-cell; }
```

Unfortunately, even though this worked in Gecko, Opera, and
KHTML/Webkit,
the lack of support in IE until version 8 made CSS tables unusable on the
Web.

—

And so Web authors, taking heed of advice to use table markup only
semantically,
began to hack out layouts with complicated mixtures of absolute
positioning
and floats.

CSS Layout today is mostly built out of this.
It's an ingeneous use of the materials available,
but a bit like building a house out of duct tape and broomsticks.

—

Meanwhile, the CSS Working Group had started on CSS3,
and several core modules, like Selectors and Color, were mostly completed.
However, we soon realized that what was needed more
was interoperability among the browsers.
CSS2.0 was riddled with errors and ambiguities,
and included several features nobody implemented.
The browsers were similarly riddled with standards-compliance bugs and inconsistent behavior,
making Web design a nightmare of working around various browsers' quirks.
So the CSSWG drafted CSS 2 Revision 1, beginning a long process of plugging the holes, fixing errors, and building test suites for the core CSS standard
Meanwhile, evangelism efforts like the Acid 2 test
helped focus browser dev teams on improving their core CSS support.

It wasn't a very exciting time in CSS development, but it laid the groundwork
for a reliable CSS3 standard. And as the CSS2.1 effort started to wind down,
CSS3 spec and implementation work was revving back up. CSS2.1 was finally
completed in May of last year; and the CSSWG has finally been able to focus
all its efforts on CSS3.

—

Beyond level 2 CSS is modularized: what this means is it's split up into parts, that can each progress individually, levelling up at its own pace.

CSS3 can be broken down into four major efforts:

* Processing Power (Things like Selectors and Media Queries.)
* Decoration (CSS3 Backgrounds and Borders, CSS gradients)
* Typography and Internationalization (Vertical text, custom fonts, line-breaking and justificaiton)
* And, probably the most difficult: Layout

(This whole list kindof goes in order from easiest to define and implement down
to the most difficult.)

It's hard to understand the work that goes into designing and building a layout system if you don't actually work on one – the expected results are so obvious, it seems simple; but the calculations that go into it involve a lot of details. Chapter 10 of CSS2.1, which deals only with calculating widths and heights of elements (and doesn't even touch margin
collapsing or floats and clearance or even tables) runs 18 pages long and is the output of many hours of excruciatingly detailed technical discussion over more than a decade. And people still find errors in it, nevermind in the browser engines that supposedly implement it.

Layout is hard. And layout for the Web is harder. Let's talk about why.

—

First, let's talk about layout in Print.

—

In Print Media,
- you know the size of the page
- you know the amount of content you have
  and in some cases, you can even tweak the content to fit the layout
- You know the fonts you have
  and you control their size.
- When you ask the computer to lay something out for you,
  and it get it not quite right,
  you can tweak it afterwards.
- And when you're done, exactly what you see
  is exactly what everyone in your audience gets.

In print, the designer is in control.
Not so on the Web.
- Different viewers have different fonts available
- Different devices have differently-sized screens,
  different resolutions,
  different aspect ratios.
- People resize their windows.
- They zoom the text size.
- And they generate content that you don't control,
  but you have to style.

If content doesn't fit, you can't just clip it. You can't copy-edit it.
You can't change the size of the paper, or the size of the font.

You could send a PDF, but nobody wants that because Web content by and
large adapts to its environment, and people *want* that. They want it to fit *their* screen, use the font-size *they're* comfortable reading, display the content *they* wrote.

So we come to some key differences between Web and Print:
Web layout has to be
 - flexible
 - adaptable
 - automatic
and
 - robust
It can't fail because it's loaded in an environment that wasn't quite what the designer had in mind...
because that happens _all the time_. Especially on my screen, because I have a small computer and I like to size my windows to half the screen so I can put them side-by-side. :)

—

Ok, let's talk about CSS3 Layout. CSS3 is really the first time the CSSWG has attempted to design a system expressly for 2D layout.
And we want to do it right for the Web,
which means we want a system that is

 * Flexible - That adapts gracefully to different screen sizes, font sizes, amounts of contents, etc.
 * Powerful - That can express most layouts designers want to have and the constraints they want it to flex under. One that makes the simple things easy, and the complex ones possible.
 * Robust - That doesn't break down and become unreadable in unexpected conditions like an unexpected font size, or an unusually long paragraph, or an uncommonly-narrow screen.
 * Understandable - Doing layout in CSS right now is complicated, hard to understand, and hard to learn. This is largely because no feature in CSS was intended to do layout. As we intentionally design real layout systems to CSS, we want to make sure the syntax is clear & self-evident, and that the concepts and algorithms make sense and reflect the behaviors authors are most likely to want.
 * Performant - Lastly, the Web is a platform for interactive applications, so a layout system for the Web needs to be performant. This last part is a bit tricky because there are some layouts people want that can only be solved with an iterative constraint solver. But we want to minimize the cases where that's necessary and make the common cases easier to optimize.

—

So far, we only really have this working for 1D layout.
There are ways to do flexible layout susing floats or abspos, but they're tricky and often fragile. These tools weren't designed for what we want in CSs layout, so the CSSWG is building new tools to do layout in CSS.

—

So now I'm going to talk about some upcoming CSS3 modules that are in varying stages of development.

First up is CSS Multi-column Layout.

In some ways, this is the simples of the CSS layout modules,
which is why its spec was completed first,
even though it's one of the hardest to implement and probably does the least
to solve 2D layout on the Web.

But it does show off some of the qualities we just talked about: flexibility and power, robustness and understandability.

Multi-column layout is very simple to use. If you want divid an element into multiple columns of flow, you just give it a column width:
```
  article { columns: 30em;
            /* fits ~30 Chinese characters, or about twice as much Latin */ }
```
and then CSS calculates how many columns would fit and splits the flow into that many columns. If there isn't room for more than one column, then there will be only one colu,mn. And if there's room for 5, there will be five columns.

You can also specify the max number of columns:
```
  article { columns: 30em 3; /* columns at least 25em wide; at most 3 columns */ }
```

The width you specify isn't an absolute, it's a goal.
If 2.5 columns will fit, you'll get two columns 45em wide each.
If only half a column will fit, you'll get one column 15em wide.

So the width might not always be what was specified, but for the reader, the design always fits the screen perfectly.
And since the column heights are auto-balanced, the layout always fits the content.

The main problem with CSS Multi-column layout is that on scrolled media, if you have a lot of content, the columns get really long and the reader has to scroll back and forth a lot, which is awkward. It's something we'll need to fix in the next level. But for short content or in paged media, it works great. And it's implemented across browsers right now, so it's something you can use today.

—

The next layout module I want to talk about is CSS Flexbox Layout.
This module is still being worked out, but it's stabilizing, and we have several experimental implementations. I'd expect it to be released later this year.

Flexbox introduces the concept of flexible sizes, so you can specify sizes as a proportion of the available space. This is pretty useful for things like toolbars:

You can use flexes absolutely,
where the proportions you give are the proportions you get;
or you can use them relatively, where leftover space is distributed after you've assigned sizes based on the content.

Flexbox layout was originally designed to handle Mozilla's UI, which is implemented in XML+CSS+JS, so the types of layout done in browser UI will be easily available for web apps Very Soon Now.

—

The third module I wanted to talk about is CSS Grid Layout.
This is a module created by putting together Bert Bos's Template Layout proposal
with a Grid Layout proposal from Microsoft. We're still in the process of fitting it all together, so the module is not finalized at all;
but it will make 2D layout grids a simple and obvious thing to develop with in CSS.

The way it works is that you declare an element to be a grid,
give it some row and column sizes, and maybe also a slot template:

```
#main { ... }
```

Then you can assign descendant elements into slots in the grid,
either by using the slot names or by positioning it with row/column indices.

I'll note we're still working out the exact syntax for this,
so don't go copy-pasting these examples. :)
But you might start to see experimental implementations of Grid Layout in the not-too-distant future.

—

The next modules I wanted to touch on are CSS Regions and Exclusions.
These are two proposals from Adobe for doing magazine-style layouts.
I want to show these because they're an example of capabilities that came from print and are really cool, but in their current state aren't suitable for the Web.

Regions allows chaining boxes into a common flow, similar to how the different columns in a multi-column element are chained into different flows.
And Exclusions allows for non-rectangular containers and floats.

To take a concrete example, Exclusions and Shapes lets you fill a circle with text.
But if you increase the font size a few notches,
the text no longer fits and starts to disapear.

Currently the proposals handle problems like this
by relegating the responsibility for solving them to JavaScript.
Which arguably works, but isn't what we're committed to.

To take another concrete example, if you want to put a circular float in the middle of an element sized to fit its content,
the position of the float depends on how the float impacts the content, which can be very expensive, performance-wise, to figure out.
You can get a fast answer or you can get an accurate one, not both.

The CSSWG is struggling with how to take these proposals
– which work fine for fixed-size, known-font, stable-content situations –
and make them work in a flexible manner,
to make them really work for the diverse input/output environment that is the Web.

And so those proposals are unlikely to make much progress in the WG
until problems like these are satisfactorily solved.
Opera and Mozilla, at least, don't want to implement CSS layout
that fails to satisfy the design principles of CSS, and that can do so
without using JavaScript or injecting empty elements into the DOM.

—

Lastly, I wanted to talk about Media Queries,
which is a module that is already well-implemented across browsers.
Media Queries allow you to tailor and reconfigure your CSS
to different output environments.
Here's a design Divya Manian, Jason Cranford Teague, and I put together
for the CSS Working Group home page.

Watch how the design folds down from 3-columns to 2-columns to 1-column.
There is no JavaScript; this is all CSS.

While CSS layout systems can and should adapt to changes in their input/output,
there are thresholds where the layout really needs to transform; and
Media Queries allows you to do that, to take multiple designs and
conditionally combine them into a single page.

So that's an overview of what's here, what's coming, when you can expect it,
and why it's not here yet. :)

Let's move on to how things go from idea to impelmentation at W3C, and
then I want to close with how you can get involved.

—

[process slide]
In the CSSWG, we can break down the development of a spec into various
levels of stability:
[steal content from blog post]

[sources of innovation slide]
So, let's see where ideas come from.
There's been a lot of argument over where designs for new CSS features
should come from.
Should they come from implementations.
Should they come from designers.
Should they come from the experts in the working group.
Actually nobody makes that argument, it's not controversial enough. :)

The truth is, CSS feature designs come from everywhere.
And we find that this works.
It's not like any of these groups has a monopoly on good ideas.
And a good design has to make sense to all of them–the people who use them,
the people who implement them, and the people who can notice problems with them
before we spend all this effort deploying them.

In reality, most CSS specs are a mix of input from all three sources anyway.

Most of the work in the standards process is not initially drafting a proposal,
but in adapting the proposals in response to feedback.
Specs go through multiple cycles of review and revision;
this is what helps ensure that their features actually make sense
and are well-designed,
that they can be implemented cross-platform,
and that their descriptions have enough detail to be implemented
interoperably
by unrelated product teams.

The CSS Working Group maintains a list of the specs it's working on
on its current-work page:
[...]
[Get Involved slide]

Anyone can get involved in standards.
You just have to have the desire and technical inclination
to understand what's going on under to hood,
and the tenacity to dig really deep into the details and implications
of all the deeply technical work that goes into building a technical standard.