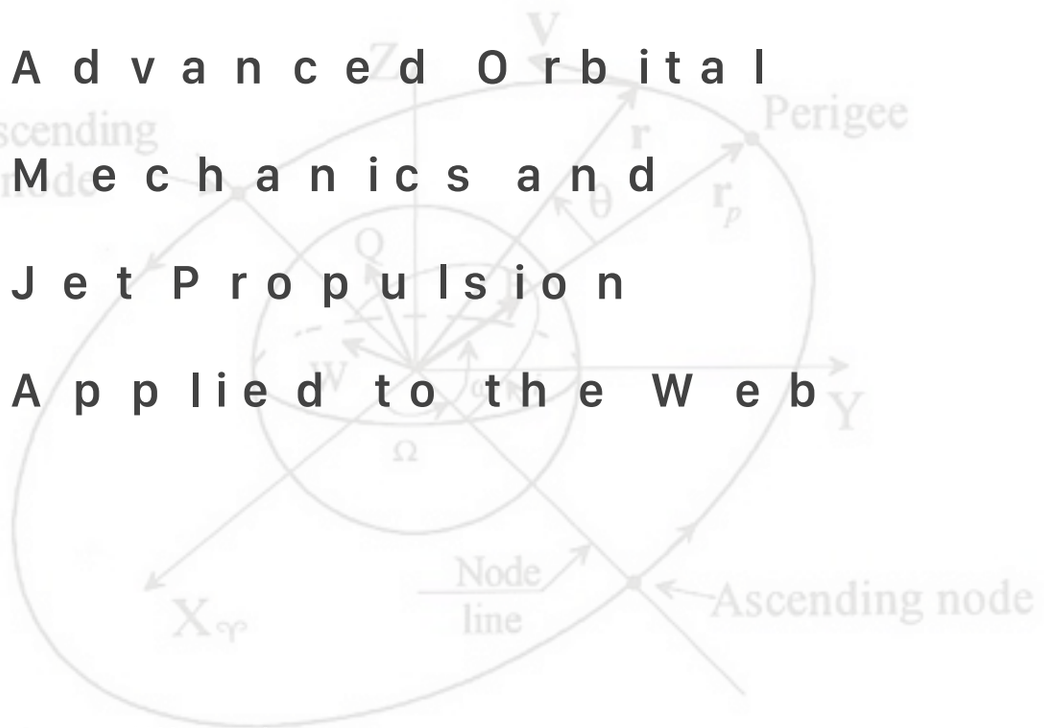
Advanced Orbital Mechanics and Jet Propulsion Applied to the Web



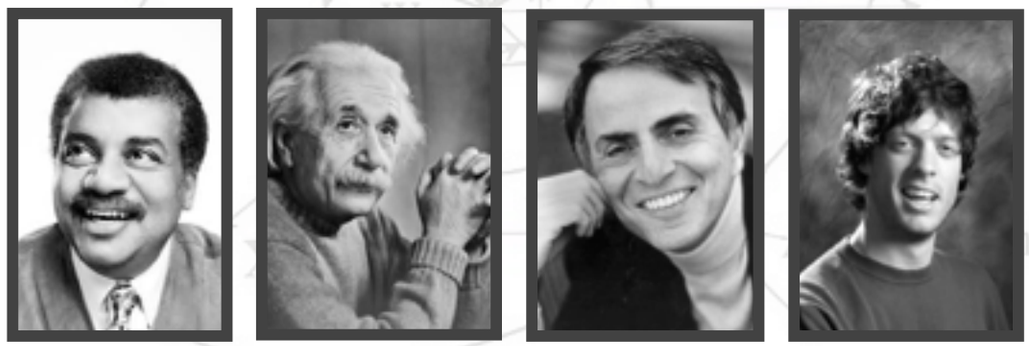
@pezillionaire

- Welcome to Whistler
- Introduce myself
- So I'm sure by now you're all like "Hey Pez, what's with the AWESOME title?"

- A few weeks back I was traveling through cyber space and this happened upon me.
- I lawl'd
- Seriously though, how awesome is it that we can land a ANYTHING on a comet traveling 135,000 km/h in space.
- But wait…
- Humanity how has advance enough technologically that we have the ability properly align a stuff on a web site too!
- An thought "I should explain how to do BOTH of these things!"

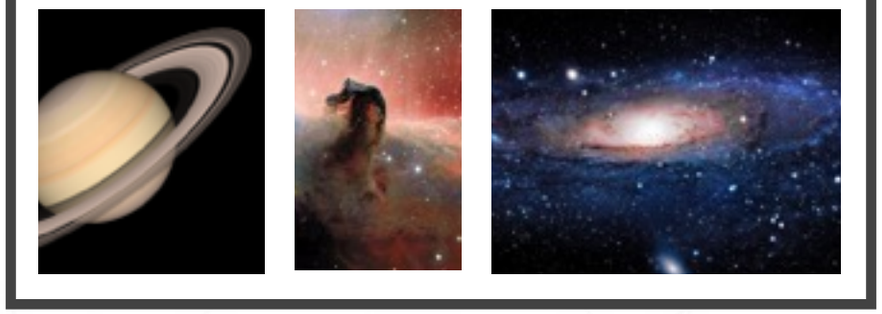
- but I only have 15 min.
- I TOTALLY cant explain how to land a robot on a comet in 15 min.
- Best news: This is only slightly more difficult than the theory special relativity.
- so without further delay…

Grand Unified Theory of Web Layout



- What the experts refer to as the grand unified theory of web layout!
- or you may know it's common name:

Flexbox



* PhD in astrophysics not required.

- Flexbox
- W3C Module for CSS (currently a "Last Call Working Draft" rec.)
- Flexbox way to layout, align and distribute space among items in a container, even when their size is unknown and/or dynamic.
- But to put it simply:

Flexbox is CSS Layout

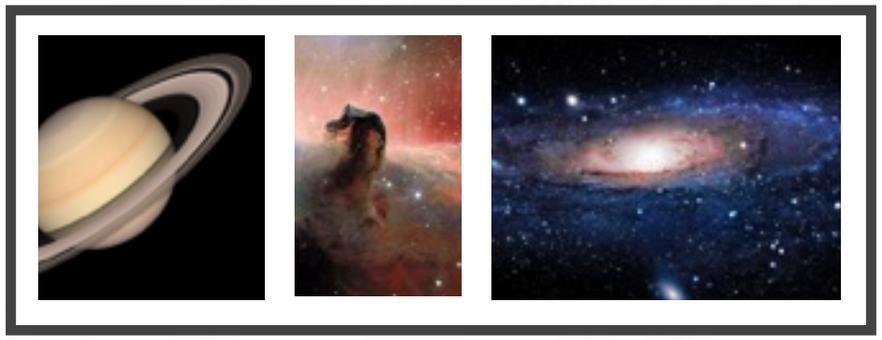
- FLEXBOX IS CSS LAYOUT
- Cool right!?! So…

How does it work?

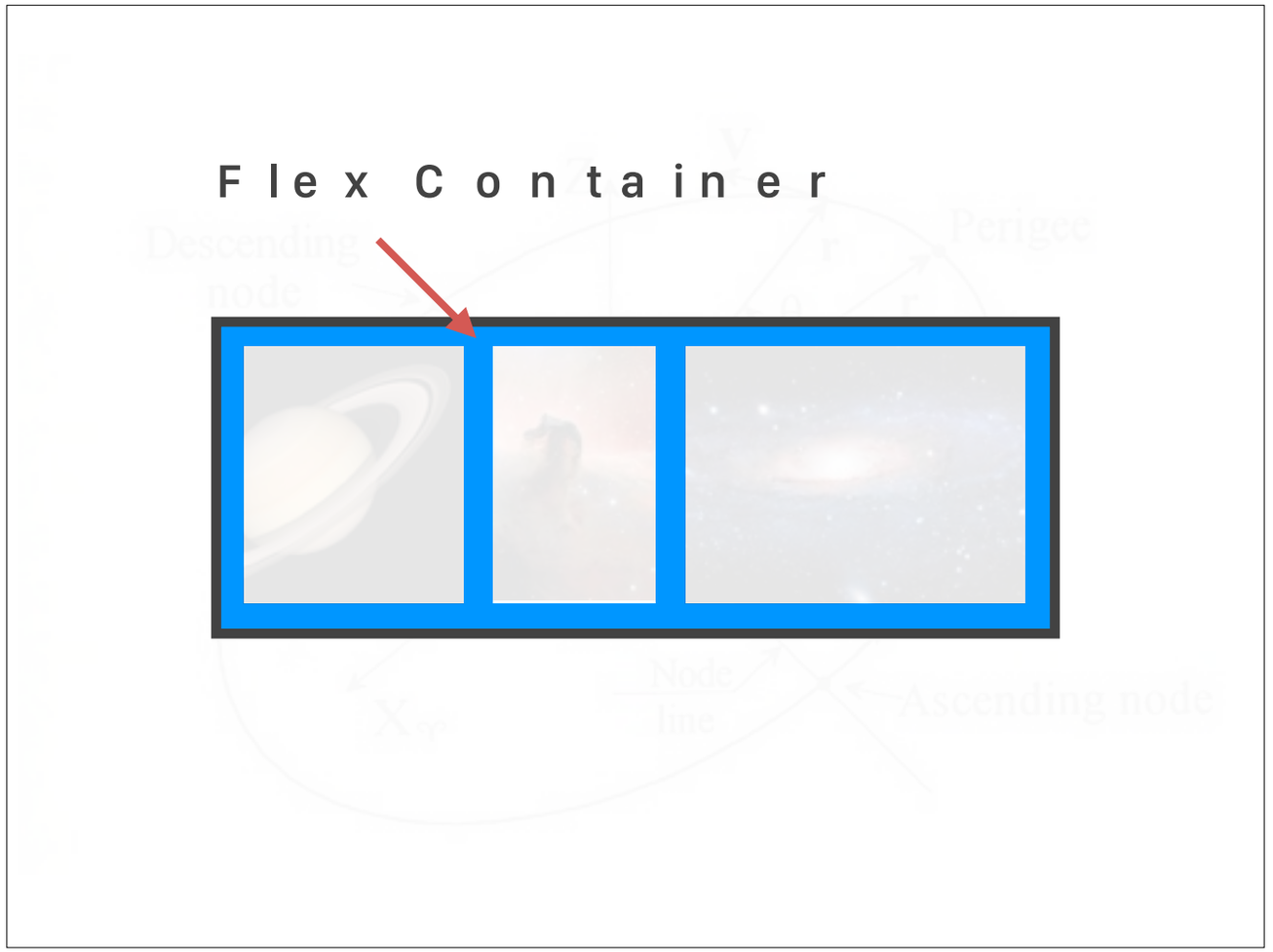
*Realjuggalos always vendor prefix

- Good news is unlike magnets, we know how it works.
- What I will cover is the current spec. Be a good Juggalo. (Violent J / Shaggy 2 Dope)

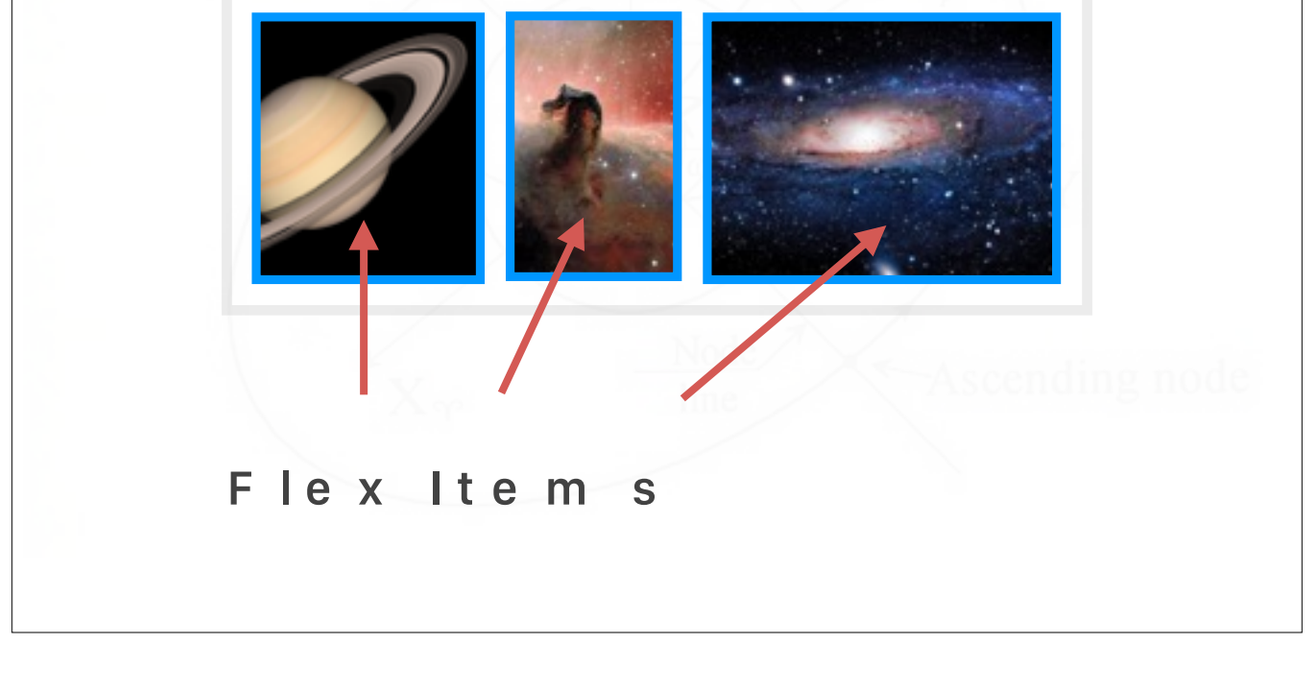
Flexbox Model



- Remember the CSS box model? Let's learn something new!
- The flexbox model has two main components…



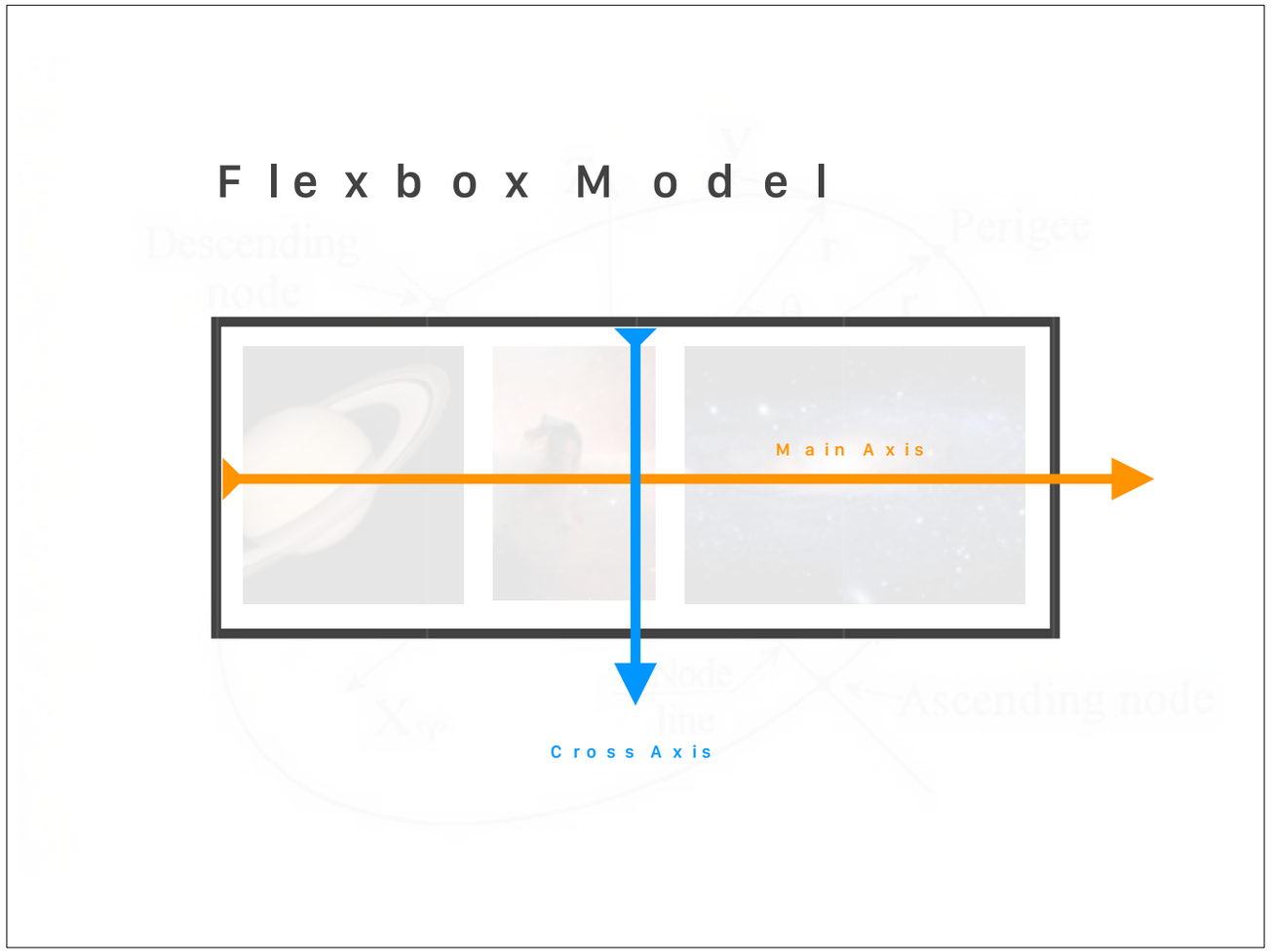
- Flex contianer



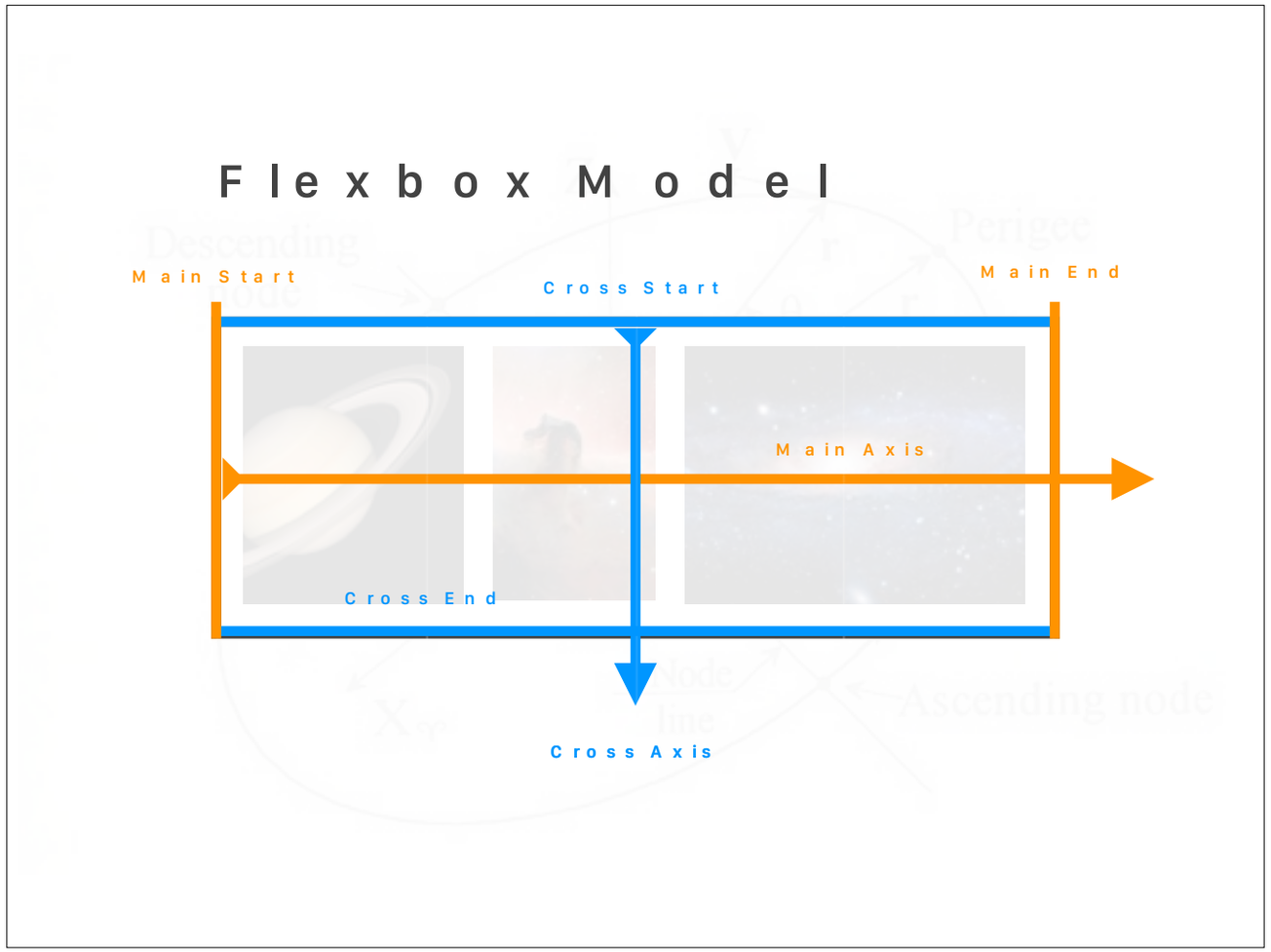
- Flex items

Flex Container

- Has the most CSS properties associated with it and needs to be set up first anyway



- **main axis** - The main axis of a flex container is the primary axis along which flex items are laid out. Beware, it is not necessarily horizontal; it depends on the flex-direction property (see below).
- **cross axis** - The axis perpendicular to the main axis is called the cross axis. Its direction depends on the main axis direction.



- **main-start | main-end** - The flex items are placed within the container starting from main-start and going to main-end.
- **cross-start | cross-end** - Flex lines are filled with items and placed into the container starting on the cross-start side of the flex container and going toward the cross-end side.

Flex Display

Sets up Flexbox

```
.container {
  display: -moz-box;     /* Firefox    */
  display: -ms-box;      /* IE 10      */
  display: -webkit-box; /* Old Safari */
  display: -webkit-flex;/* Chrome     */
  display: flex;         /* Spec       */
}
```

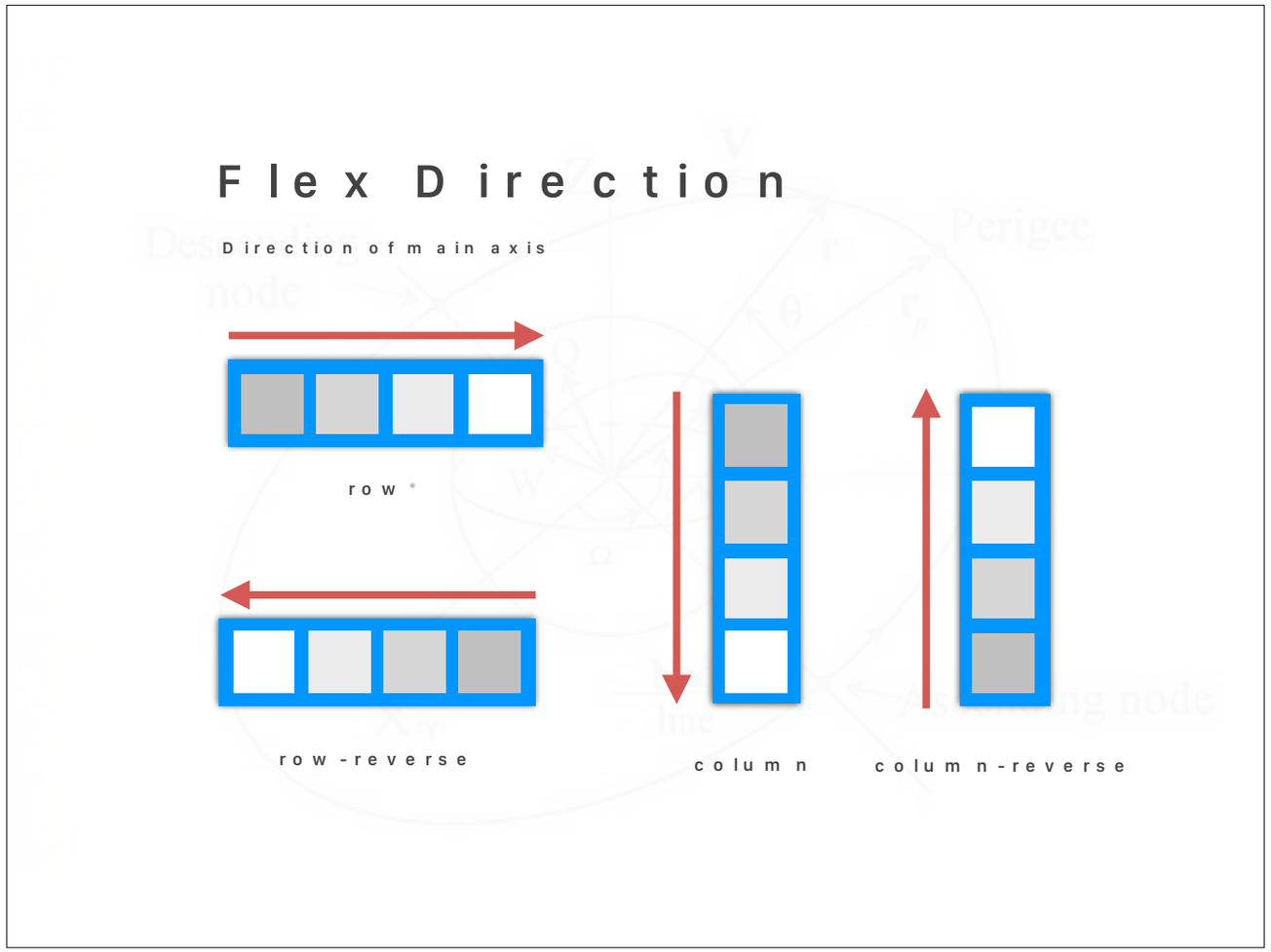
- First and only vendor prefixes you will see, for brevity (and maybe a live demo after) I'm only showing Spec from here on out!
- ANYTHING can act as a container. if you can apply display:block to it you can apply display:flex

Flex Direction

Direction of main axis

```
.container {
  flex-direction: row;*
                  row-reverse;
                  column;
                  column-reverse;
}
```

- Sets your Main Axis.



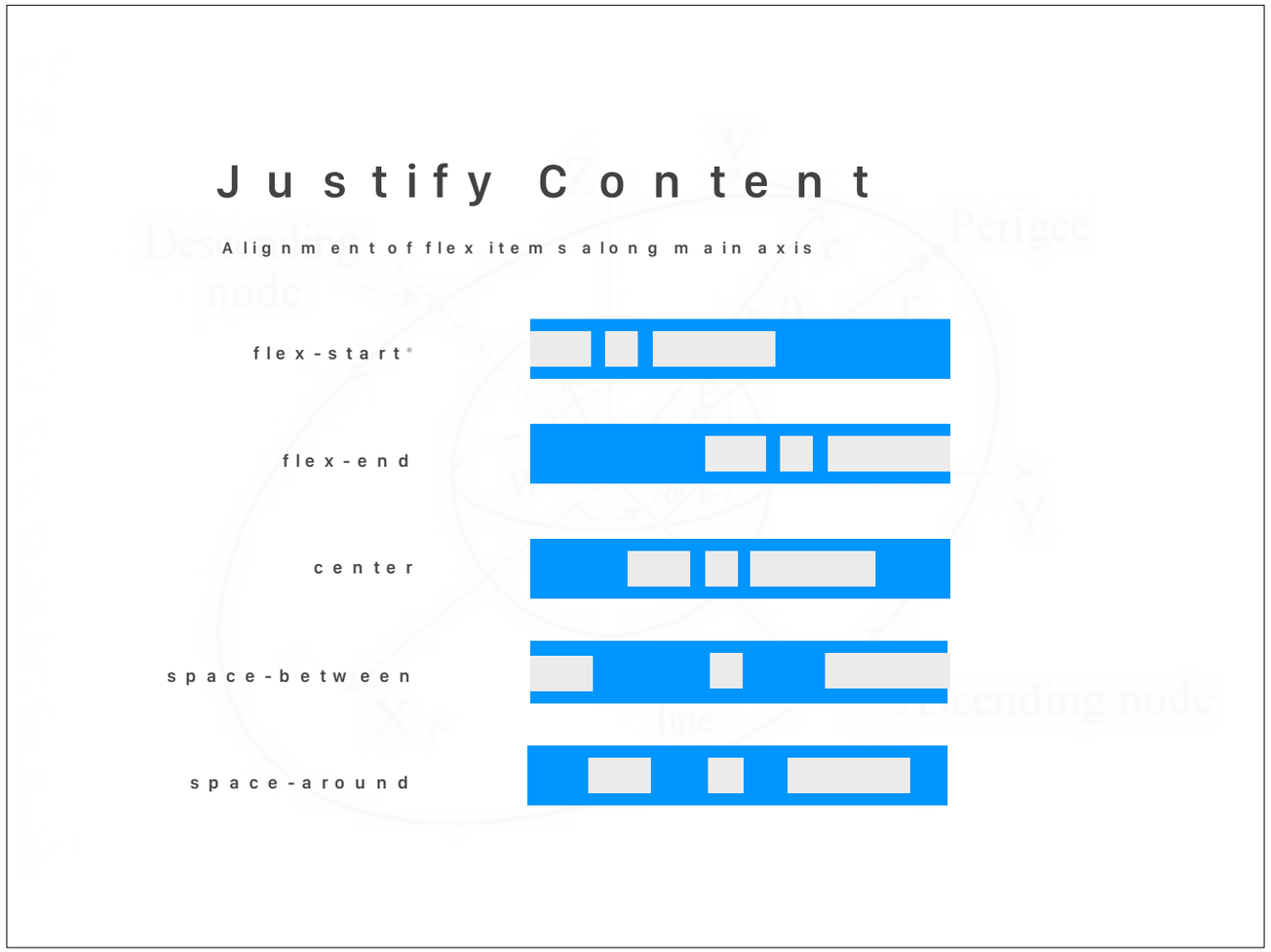
- row (default): left to right in ltr; right to left in rtl
- row-reverse: right to left in ltr; left to right in rtl
- column: same as row but top to bottom
- column-reverse: same as row-reverse but bottom to top
- **NOTE:** For ease of explanation, all visual examples will be using flex-direction:row and LTR

Justify Content

Alignment of flex items along main axis

```
.container {
  justify-content:  flex-start;*
                    flex-end;
                    center;
                    space-between;
                    space-around;
}
```

This defines the alignment along the main axis. It helps distribute extra free space left over when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.



flex-start (default): items are packed toward the start line

flex-end: items are packed toward to end line

center: items are centred along the line

space-between: items are evenly distributed in the line; first item is on the start line, last item on the end line.

space-around: items are evenly distributed in the line with equal space around them

Align Items

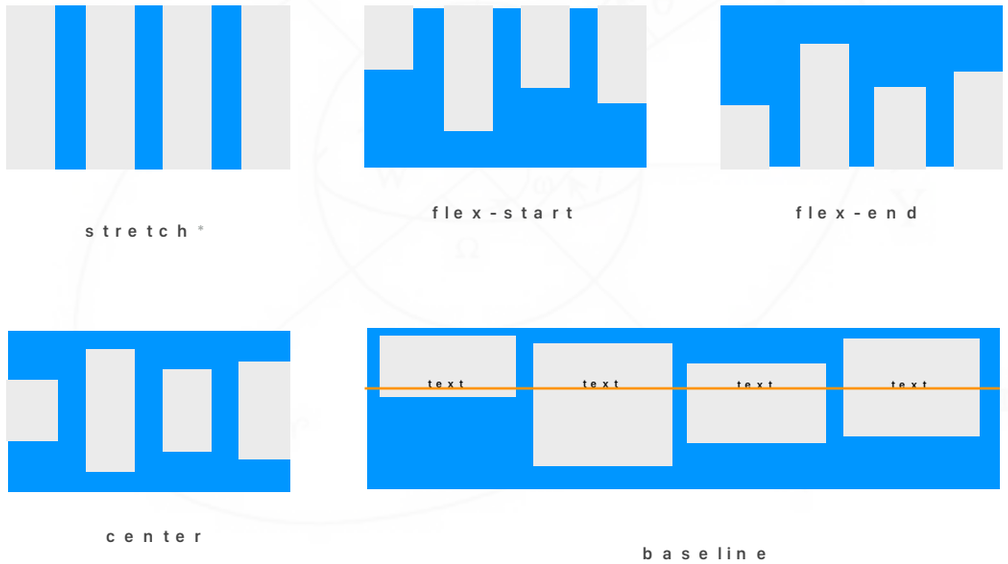
Layout of flex items along cross axis

```
.container {
  align-items: stretch;*
               flex-start;
               flex-end;
               center;
               baseline;
}
```

This defines the default behaviour for how flex items are laid out along the cross axis on the current line. Think of it as the justify-content version for the cross-axis (perpendicular to the main-axis).

Align Items

Layout of flex items along cross axis

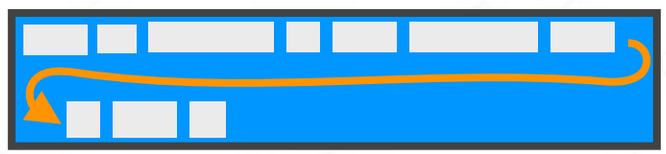


flex-start: cross-start margin edge of the items is placed on the cross-start line
flex-end: cross-end margin edge of the items is placed on the cross-end line
center: items are centered in the cross-axis **(OMFG VERTICAL CENTREING)**
stretch: (default): stretch to fill the container (still respect min-width/max-width)
baseline: items are aligned such as their baselines align

Flex Wrap

Allow flex items to wrap on multiple lines

```
.container {
  flex-wrap: nowrap;*
             wrap;
             wrap-reverse;
}
```



wrap (ltr)

- Support is limited currently
- By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property. Direction also plays a role here, determining the direction new lines are stacked in.
- **nowrap** (default): single-line / left to right in ltr; right to left in rtl
- **wrap**: multi-line / left to right in ltr; right to left in rtl
- **wrap-reverse:** multi-line / right to left in ltr; left to right in rtl

Align Content

Alignment of multiple lines along the cross axis

```
.container {
  align-content:  space-around;*
                  flex-start;
                  flex-end;
                  center;
                  space-between;
}
```

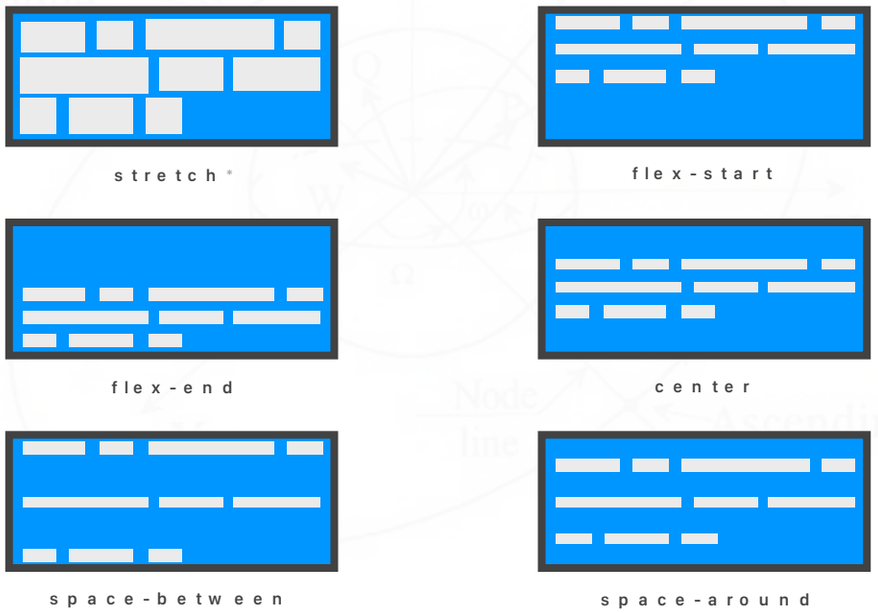
Note: This property only applies when there is more than one line of flex items.

This aligns a flex container's lines within when there is extra space in the cross-axis, similar to how justify-content aligns individual items within the main-axis.

Note: this property has no effect when there is only one line of flex items.

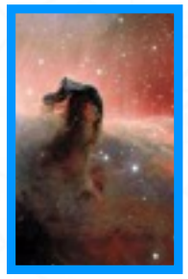
Align Content

Alignment of multiple lines along the cross axis



stretch (default): lines stretch to take up the remaining space
flex-start: lines packed to the start of the container
flex-end: lines packed to the end of the container
center: lines packed to the center of the container
space-between: lines evenly distributed; the first line is at the start of the container while the last one is at the end
space-around: lines evenly distributed with equal space between them

Flex Items


- Now we can play with the items in our flexbox
- Fewer properties to remember, but some awesome stuff.

Order

Set the frigging order of flex items!

```
#item {
  order: [number];
}
```

```
#item {order: 2;}
```


```
#item {order: 3;}
```

Note: TOTALLY vendor prefix this shiz!

- By default, flex items are laid out in the source order.
- The order property controls the order in which they appear in the flex container.
- please vendor prefix this for max compatibility.

Flex Grow

Defines how flex items grow

```
.item {
  flex-grow: [number]; /* 0 */
}
```

```
.item {flex-grow: 0;}
```


```
.item {flex-grow: 1;}
```


```
#two {flex-grow: 2;}
```

- This defines the ability for a flex item to grow if necessary.
- It accepts a unitless value that serves as a proportion.
- Dictating what amount of the available space inside the flex container the item should take up.

EXAMPLES

- Set to 0, an item will not grow larger than needed.
- If all items have flex-grow set to 1, every child will set to an equal size inside the container.
- If you were to give one of the children a value of 2, that child would take up twice as much space as the others.

Flex Shrink

Defines how flex items shrink as container collapses

```
#item {
  flex-shrink: [number]; /* 1 */
}
```

```
#item {flex-shrink: 1;}
```


```
#item {flex-shrink: 2;}
```

- Opposite of flex-grow.
- This defines the ability for a flex item to shrink as a container collapses.
- 0 will only shrink item to minimum possible size to fully hold it’s contents (may even overflow out of container)
- all items set to 1 will scale down proportionally
- Hard to describe so you should totally play with this!

Flex Basis

Defines the default size of an element

```
#item {
  flex-basis: auto;*
              [length];
}
```

```
#item {flex-basis: auto;}
```

```
#item {flex-basis: 50px;}
```

This defines the default size of an element before the remaining space is distributed.

AWESOME EXAMPLE

If you set basis to 50px and shrink to 0 an item will not shrink below 50px.

Flex (shorthand)

One property to rule them all

```
#item {
  flex: [grow][shrink][basis];
  /*      0      1      auto    */
}
```

Note: I can't remember if I told you to vendor prefix or not but you should probably vendor prefix.

- This is the shorthand for grow, shrink and basis.
- The second and third parameters (flex-shrink and flex-basis) are optional. Default is 0 1 auto.
- please vendor prefix this for max compatibility.

Align Self

Override default alignment on individual flex item

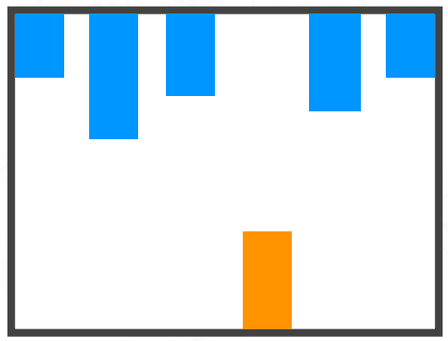
```
.item {
  align-self: stretch;*
              auto;
              flex-start;
              flex-end;
              center;
              baseline;
}
```

- This allows the default alignment (or the one specified by align-items) to be overridden for individual flex items.
- Same values as align-items.

Align Self

Override default alignment on individual flex item

```
.container {align-items: flex-start;}
#item {align-self:flex-end;}
```

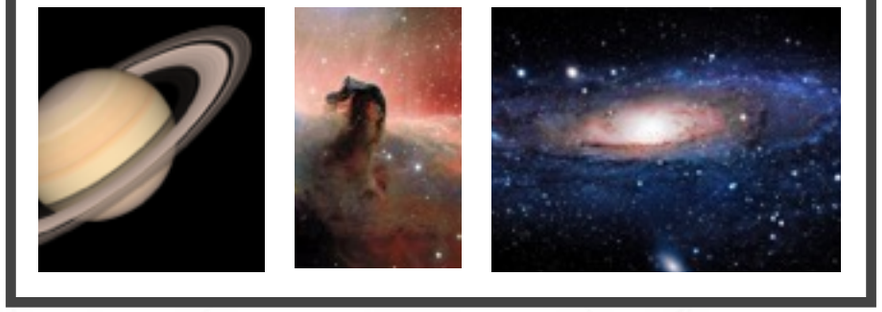


EXAMPLE

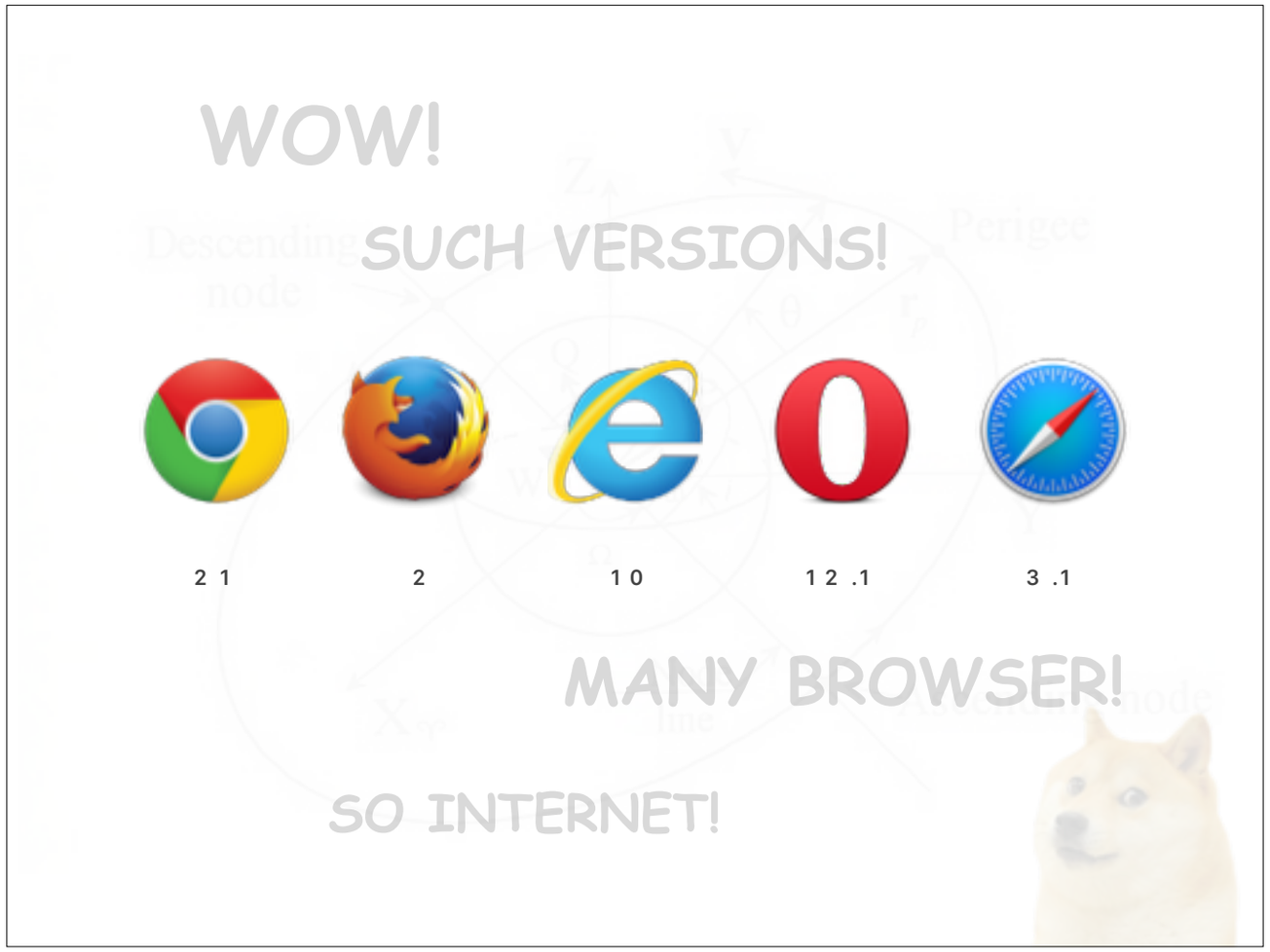
if my container’s align-items is flex-start and I want one item at flex-end

(End of tech stuff)

Flexbox



- Congratulations! You've now learned the grand unified theory of web layout!
- Now, you too can powerfully and accurately position dynamic content on a web page.
- Please, go home and play around with flexbox if you haven't already!



- Very well supported by all major browsers (unless IE 9)

If you don't use vendor prefixes

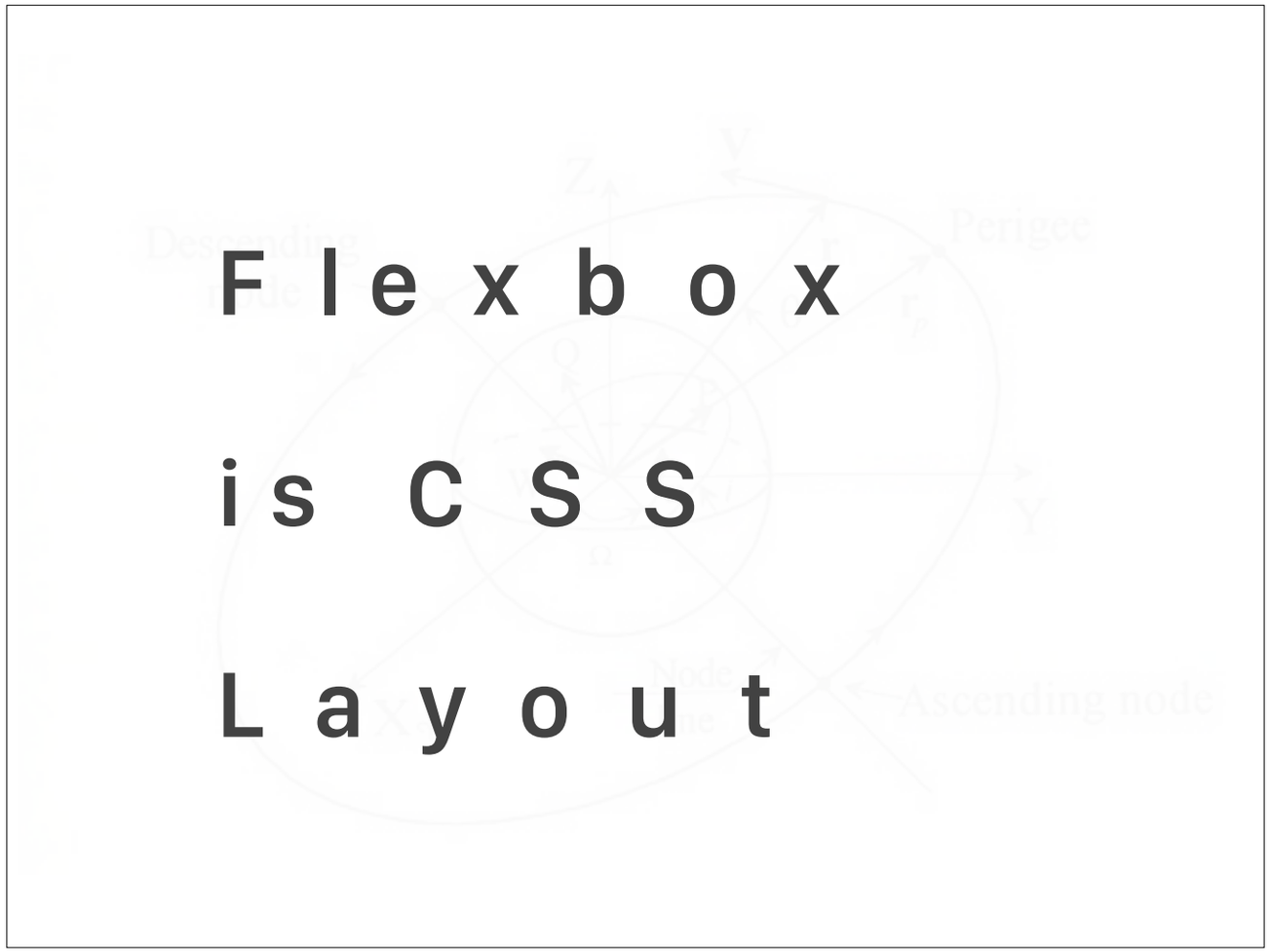
You're gonna have a bad time.

- Seriously, for maximum compatibility, please vendor prefix this.
- Or, if you use Sass, use a library such as Compass or Bourbon to do it for you!
- But let's take the opportunity…

Humanity's victories:

- [x] **land probe perfectly on a comet 310 million miles away, using science**
- [x] **get stuff on a web page to align properly using CSS**

- … and celebrate humanity's victories together!
- (cheers if you have beers)
- And from now on remember…



FLEXBOX IS CSS LAYOUT!

Thank You

& Enjoy Whistler

pezillionaire.com/talks/flexbox

@pezillionaire

Thank you!