WEB BROWSERS AS COMPILERS

SCOTT BUCKLEY

MACQUARIE UNIVERSITY

buck.ly/sapling15

	Compiler	Browser
Input	Source code (+libraries?)	HTML (+stylesheets?)
Parsing	Source code → AST	HTML → DOM
Annotation **(Translation)** **(Transformation)**	Semantic Analysis (into core language?) (optimisation?)	CSS Layout (build render tree?) (DOM manipulation?)
Output	Code generation	Rendering

WHAT IS CSS?

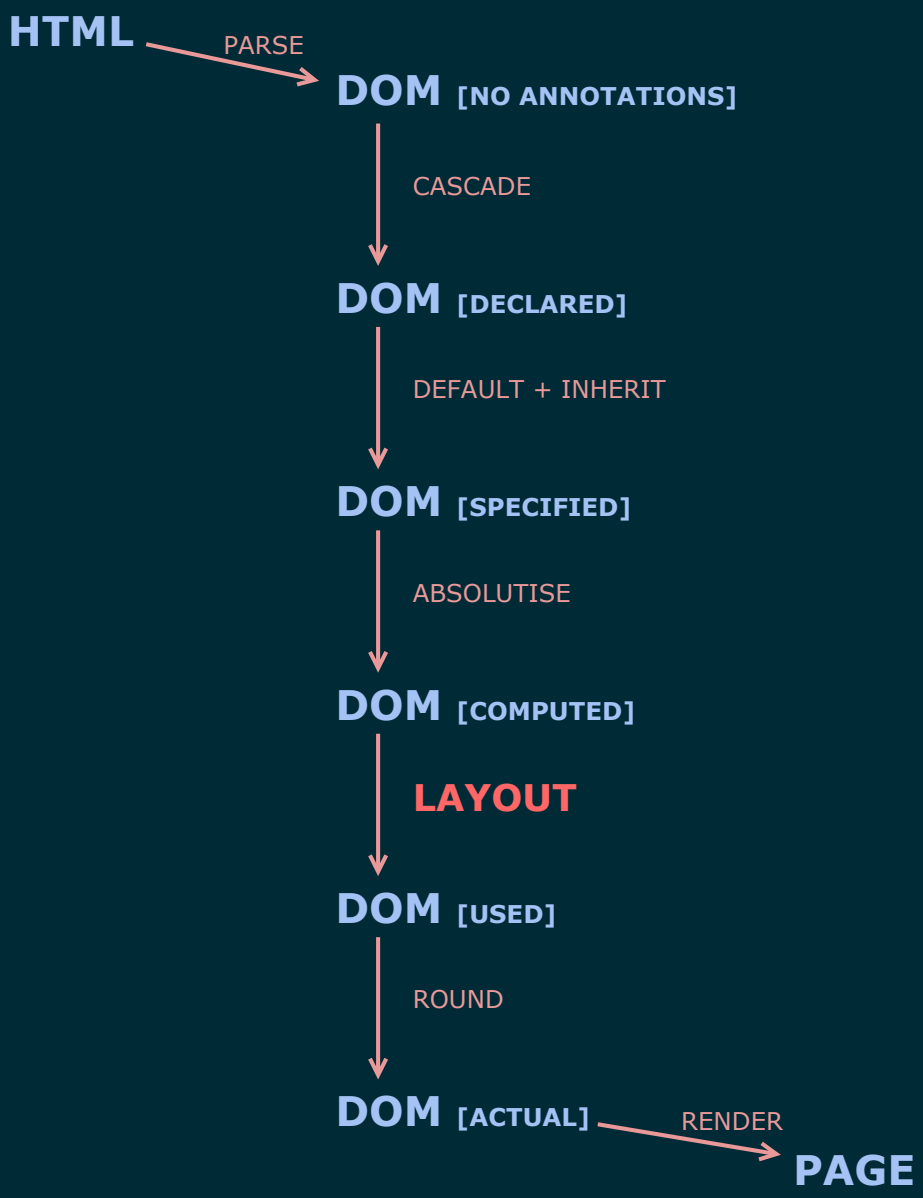
> *"User agents are not required to present HTML documents in any particular way."*
>
> *The HTML spec.*

> *"In general, user agents [...] support CSS."*
>
> *The HTML spec.*

CSS formats HTML documents.

HOW DOES CSS WORK?

- HTML → DOM (tree)
- Many properties per node (e.g. `padding` or `width`)
- Some specified by author (via stylesheets)
- Rest are calculated
- Properties evaluated in stages
- Final values dictate layout


HTML
PARSE
DOM [NO ANNOTATIONS]
No values anywhere
CASCADE
DOM [DECLARED]
Some properties have values
DEFAULT + INHERIT
DOM [SPECIFIED]
All properties have values
ABSOLUTISE
DOM [COMPUTED]
All properties in px, except auto etc
LAYOUT
DOM [USED]
All properties in px
ROUND
DOM [ACTUAL]
Values compatible with screen
RENDER
PAGE

LAYOUT

“[T]he computed value resolves the specified value as far as possible without laying out the document or performing other [difficult] operations, such as [...] retrieving values other than from the element and its parent.”

“The used value is the result of [...] completing any remaining calculations to [the final] value used in the layout of the document.”

The CSS spec.

AKA COMPUTED → USED

LAYOUT IS CALCULATION FROM CONTEXT.

LAYOUT IS THE PROCESS OF RESOLVING DEPENDENCIES.

WHAT IS SEMANTIC ANALYSIS?

- Infers information about AST nodes.
- **Links** related nodes (variable use + declaration).
- Finds semantic errors, such as type mismatches.
- Is about recognising **relationships** between elements.
- Uses an element's **larger context** to infer about it.

SEMANTIC ANALYSIS

Finds errors (etc), by

- Understanding context
- Resolving dependencies

LAYOUT

Calculates positions, by

- Understanding context
- Resolving dependencies

For **Semantic Analysis**, we use *Attribute Grammars*, which allow us to annotate AST nodes in a context-sensitive way.

Layout requires resolving dependencies on a tree.
This is **the same problem** as Semantic Analysis.

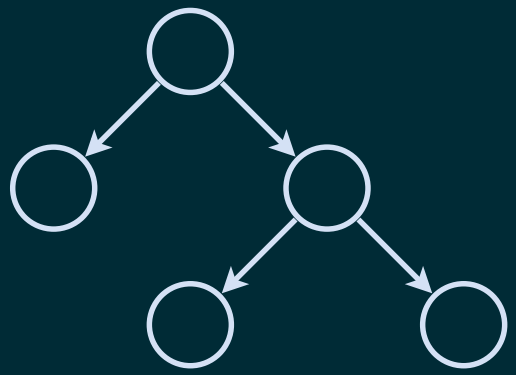
So why don't we use *Attribute Grammars* for **Layout**?

WHAT ARE ATTRIBUTE GRAMMARS?

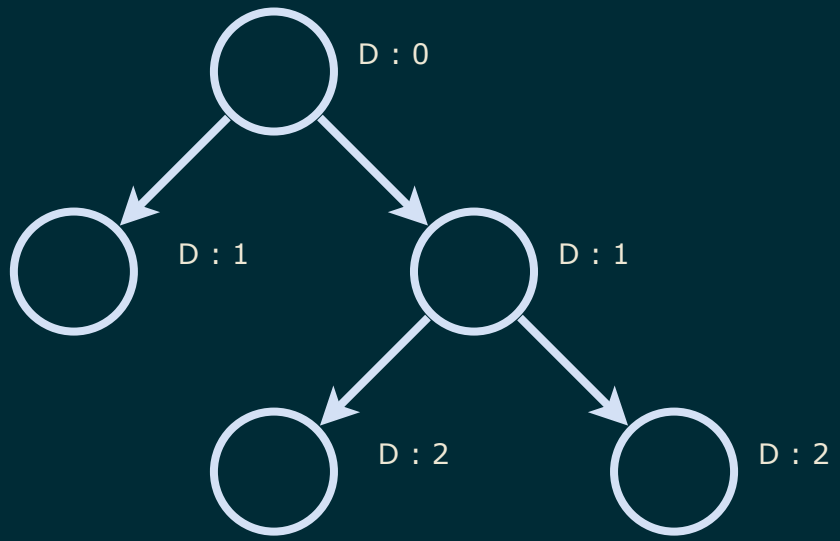
Traditionally, AGs are relational formulae attached to CFG production rules.

Generally, AGs are formulae for relational calculations on trees.

Attribute Grammars annotate tree nodes, using context.

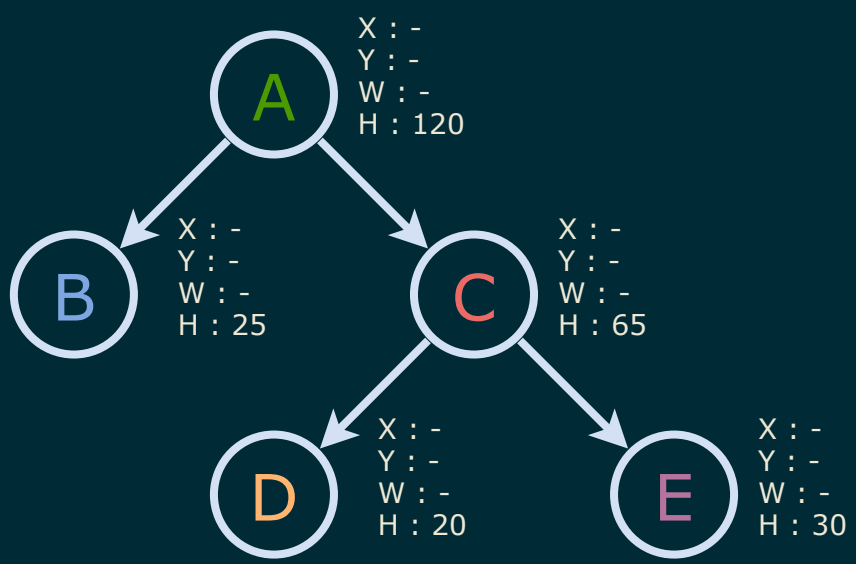


```
node.depth = root: 0
             else: parent.depth + 1
```

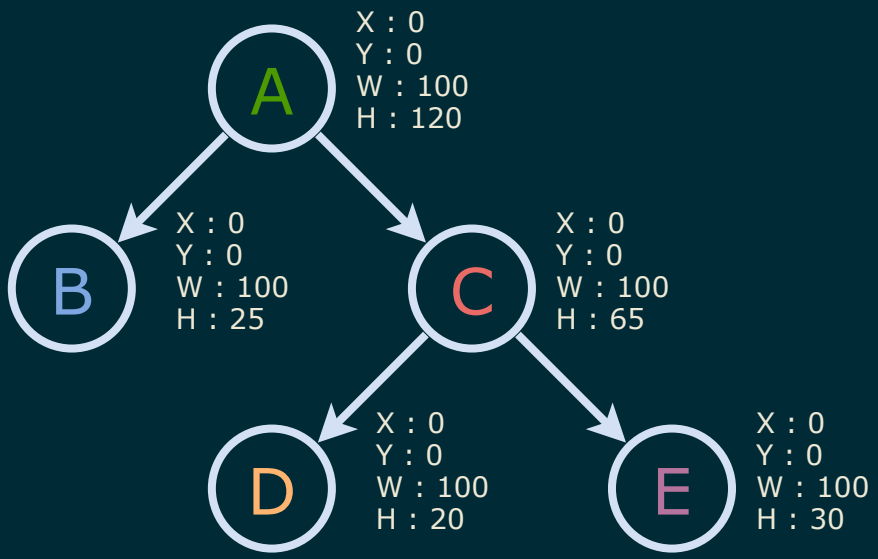


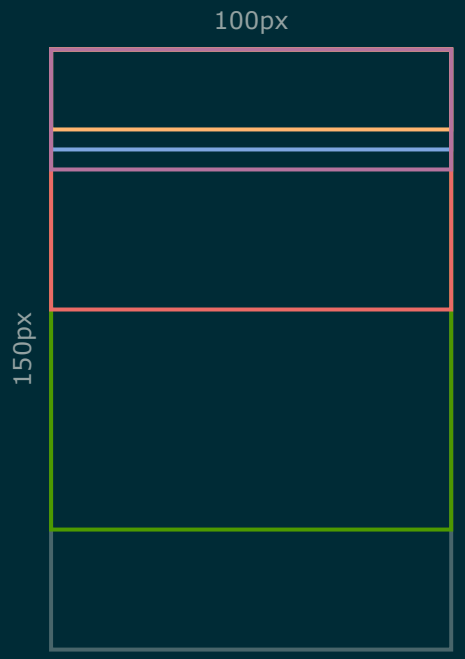
ATTRIBUTE GRAMMARS FOR LAYOUT

A PARTIALLY DECORATED TREE



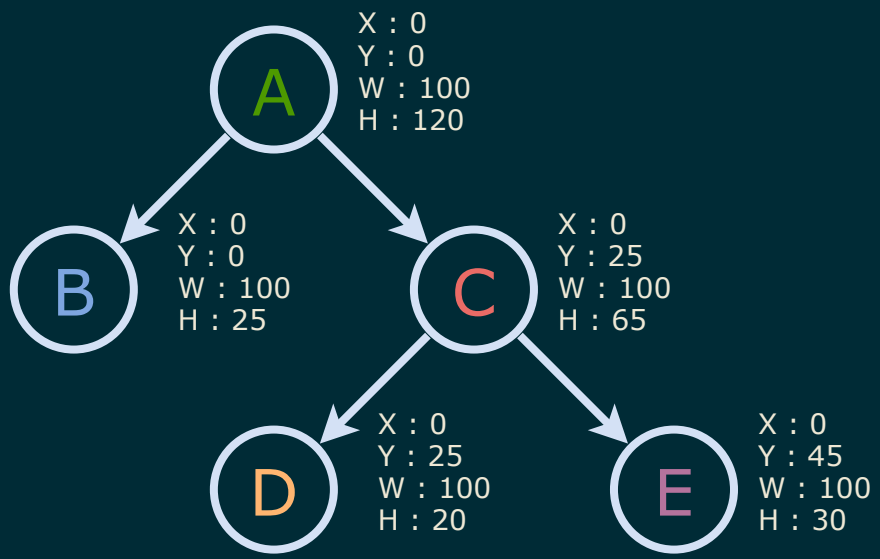
APPLY DEFAULTS AND RENDER

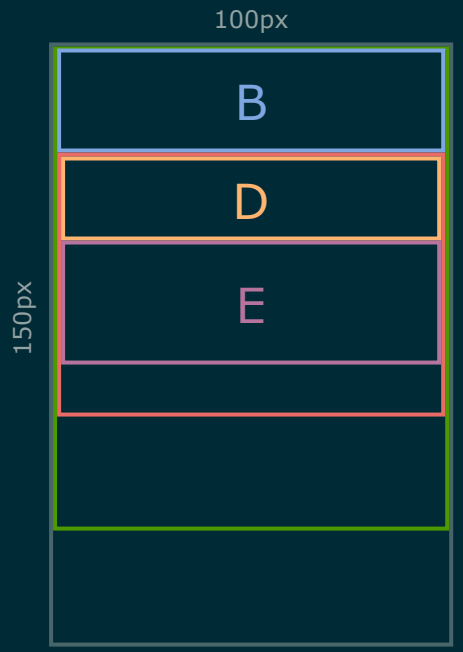




AN ATTRIBUTE FOR Y-POS

```
div.y =  root: 0
        first: parent.y
         else: prev.y + prev.height
```



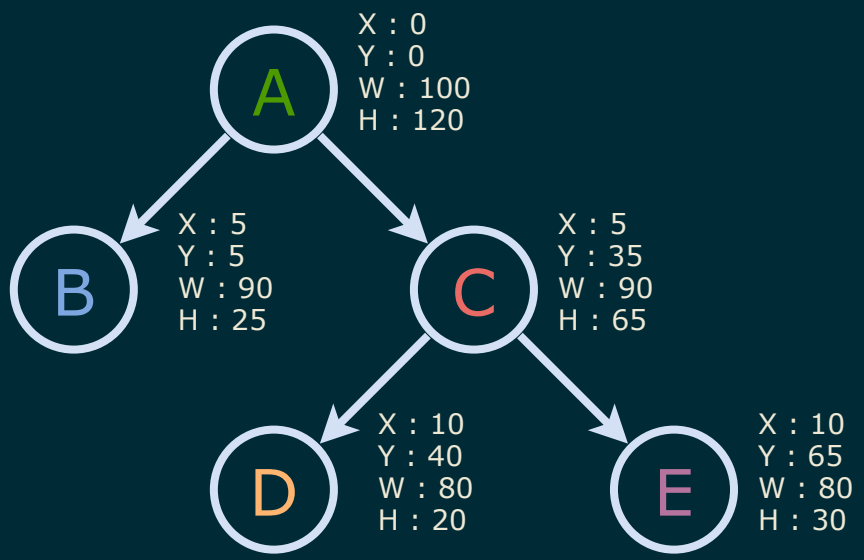


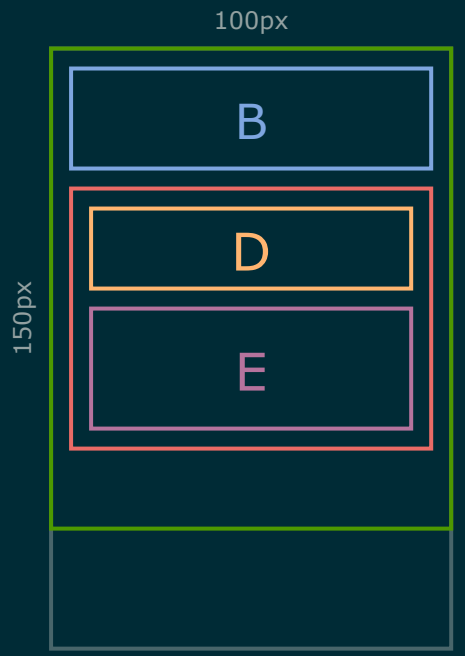
AND PADDING

```
div.y =  root: 0
        first: parent.y + PAD
         else: prev.y + prev.height + PAD

div.x = root: 0
        else: parent.x + PAD

div.w = root: 100
        else: parent.w - PAD×2
```





WHY USE ATTRIBUTE GRAMMARS?

- They suit the problem.
- They are succinct.
- They are well-researched.
- They can be optimised (externally).

CHALLENGES

- Some subproblems don't fit so nicely into the AG schema.
- A direct implementation will not run as fast as existing browsers.
- Interaction with DOM manipulation may pose challenges.*

QUESTIONS?