Project Ribbon

Layerizing the Style Engine by CSS Property

jiameng@

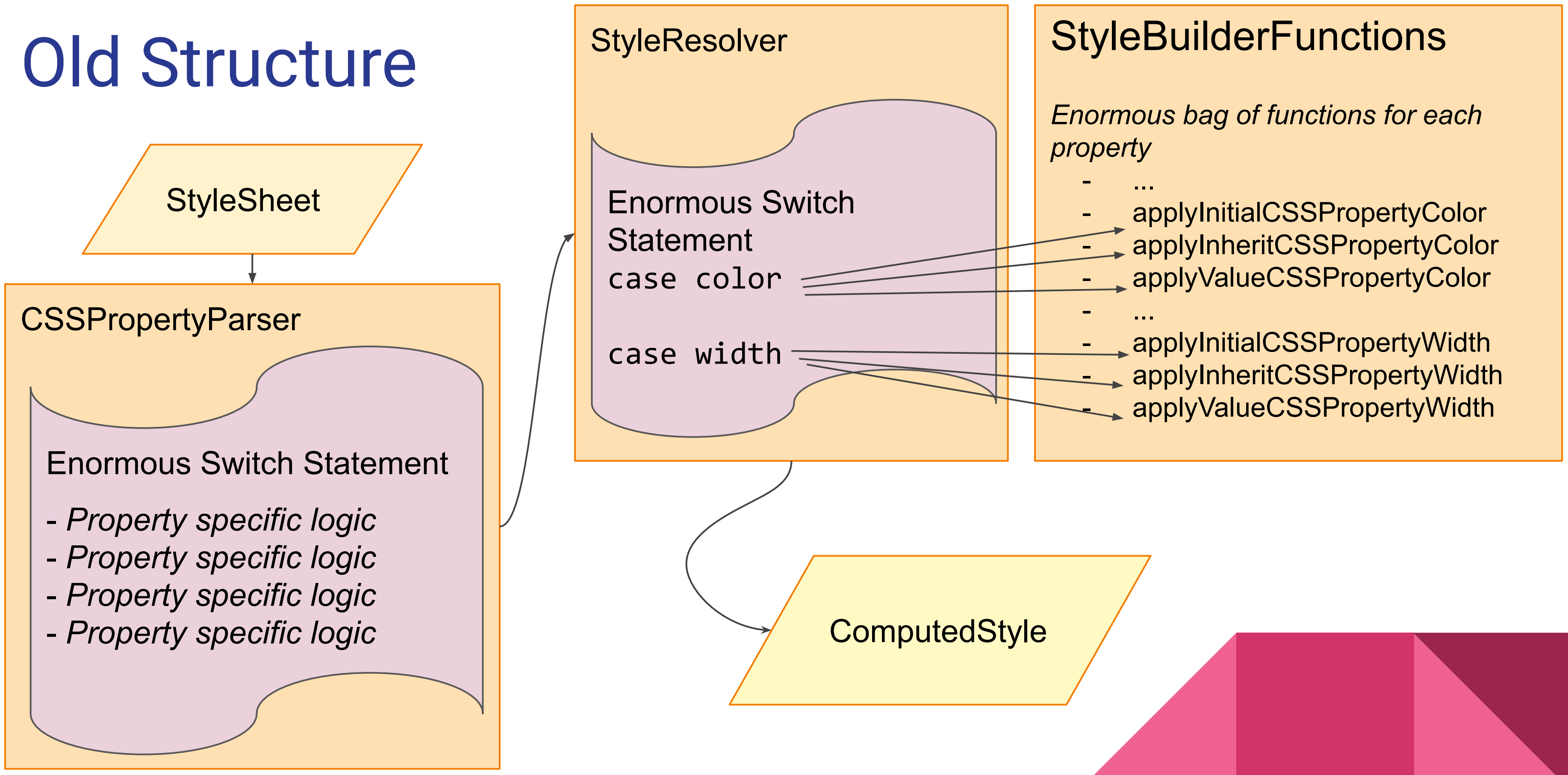
Blinkon 8, September 2017

Outline

- Overview and Motivation
- Ribbon Fundamentals
- Current Status
- Next Steps
- Team and Design Docs

Overview and Motivation

Old Structure



StyleSheet
CSSPropertyParser
Enormous Switch Statement
- Property specific logic
- Property specific logic
- Property specific logic
- Property specific logic
StyleResolver
Enormous Switch Statement
case color
case width
StyleBuilderFunctions
Enormous bag of functions for each property
- ...
- applyInitialCSSPropertyColor
- applyInheritCSSPropertyColor
- applyValueCSSPropertyColor
- ...
- applyInitialCSSPropertyWidth
- applyInheritCSSPropertyWidth
- applyValueCSSPropertyWidth
ComputedStyle

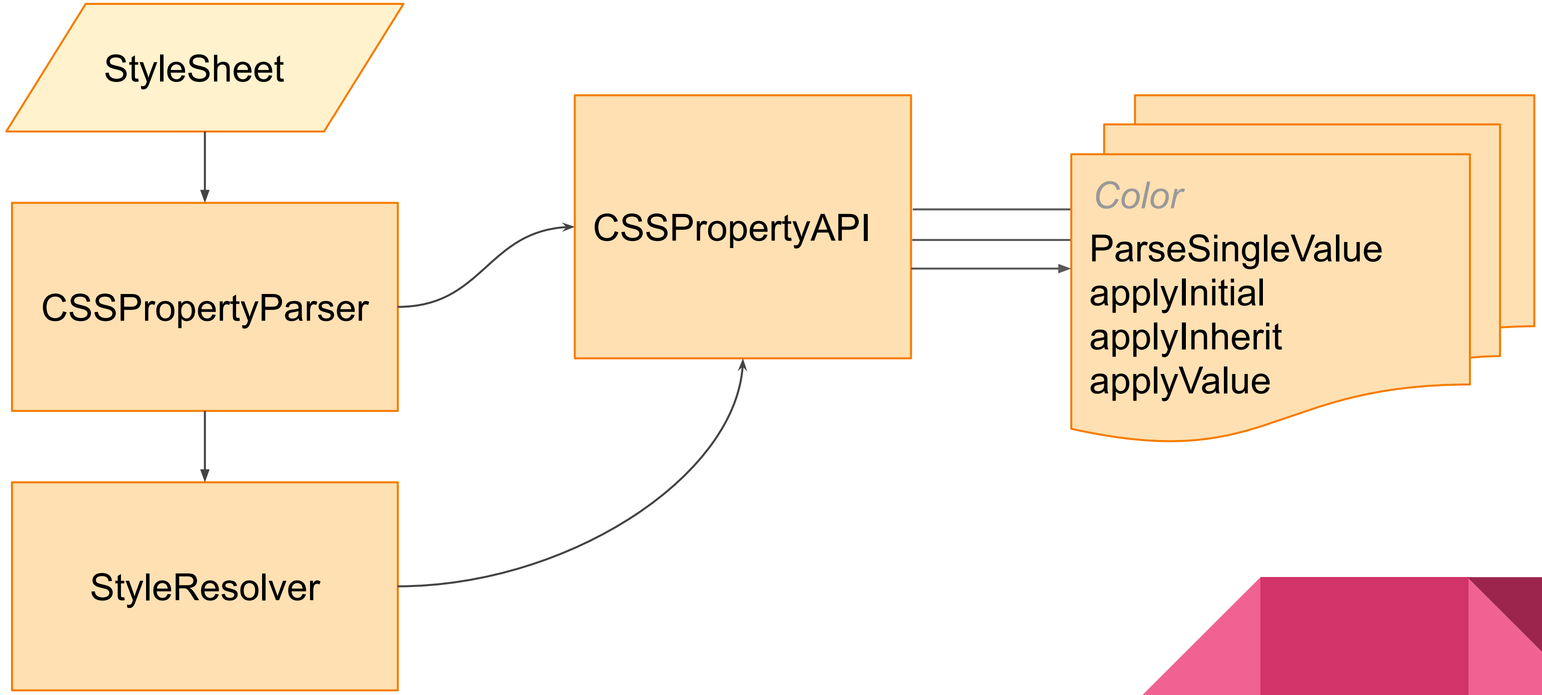
Old Structure

- Problems
    - Hard to maintain and/or add new properties
    - Hard to track down bugs to relevant properties
    - Slow because of switch-case statements

New Structure Under Ribbon

- Goal: separate engine logic from property logic
- Benefits
  - Clearer code
  - Support custom properties
  - Potentially improving performance

New Structure Under Ribbon

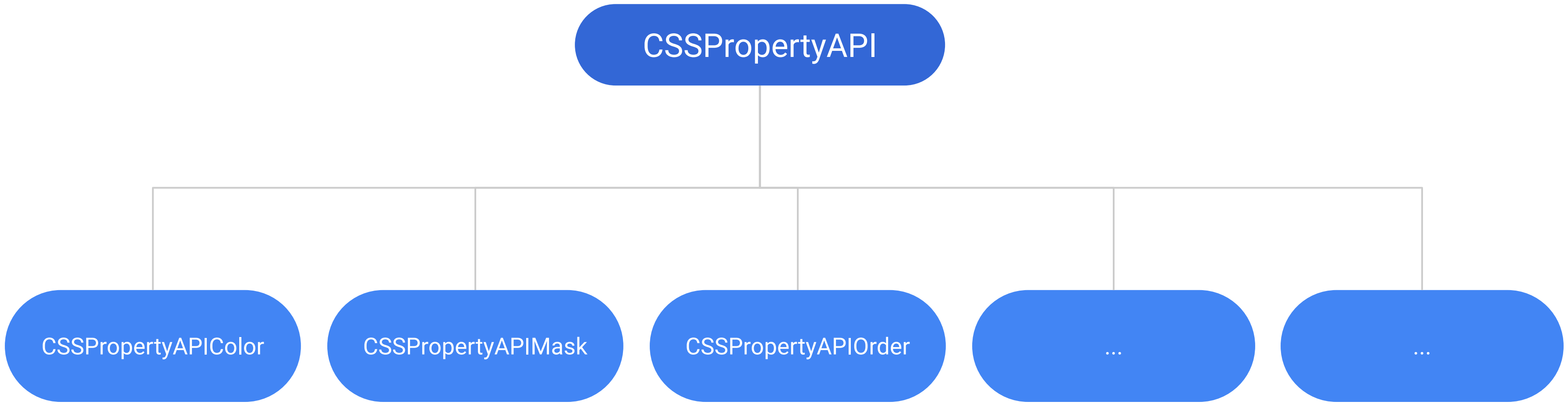


New Structure Under Ribbon

- Fundamentals of Ribbon
    - API class structure
    - Property dispatch via virtual class pointers

Ribbon Fundamentals

API Class Structure



API Class Structure

- Common interface (ParseSingleValue, ApplyInitial, IsInherited etc)
    - Subclasses implement property-specific logic for ribbonized properties
    - Base class implements default logic or unribbonized properties
- Designed to work with property dispatch via virtual class pointers (see next)

Property Dispatch

- Call site needs to call suitable property API for parsing, style building etc
- Which property API to call?
- Two options
  - Descriptors of function pointers
  - Arrays of virtual class pointers

Descriptors of Function Pointers

```
static constexpr CSSPropertyDescriptor descriptors[] = {
  {
    CSSPropertyAPIColor::ParseSingleValue,
    nullptr,
  },
  {
    nullptr,
    CSSPropertyAPIMask::ApplyInitial,
  },
...
};

static size_t descriptorIndices[] = { 0, 1, 2 … };

const CSSPropertyDescriptor& CSSPropertyDescriptor::Get(CSSPropertyID id) {
  return descriptors[descriptorIndices[id]];
}
```

One per property

Unimplemented functions

Property ids

Descriptors of Function Pointers

```
CSSValue* ParseSingleValue(CSSPropertyID id, ...) {
  …
  const CSSPropertyDescriptor& d = CSSPropertyDescriptor::Get(id);
  if (d.ParseSingleValue) {
    return d.ParseSingleValue();
  }
  switch (id) {
    …
  }
}
```

Properties without API implementations

Descriptors of Function Pointers

- Problems
  - Difficult to maintain
  - Slow to get required functions

Array of Virtual Class Pointers

```
static constexpr CSSPropertyAPI api_default;
static constexpr CSSPropertyAPIColor api_color;
static constexpr CSSPropertyAPIMask api_mask;

constexpr const CSSPropertyAPI* const property_apis[] = {
  &api_default,
  &api_color,
  &api_mask,
…
}

const CSSPropertyAPI& CSSPropertyAPI::Get(CSSPropertyID id) {
  return *property_apis[id];
}
```

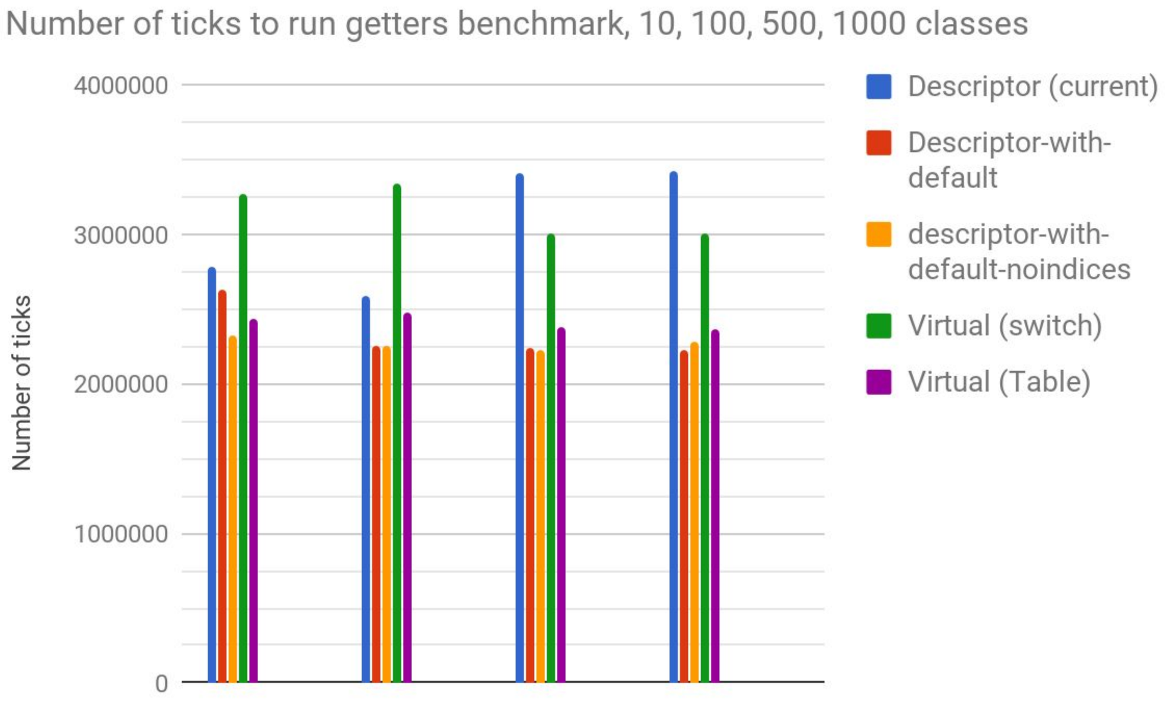
Array of Virtual Class Pointers

```
// Callsites
CSSValue* ParseSingleValue(CSSPropertyID id, ...) {
 …
 return CSSPropertyAPI::Get(id).ParseSingleValue()
}
```

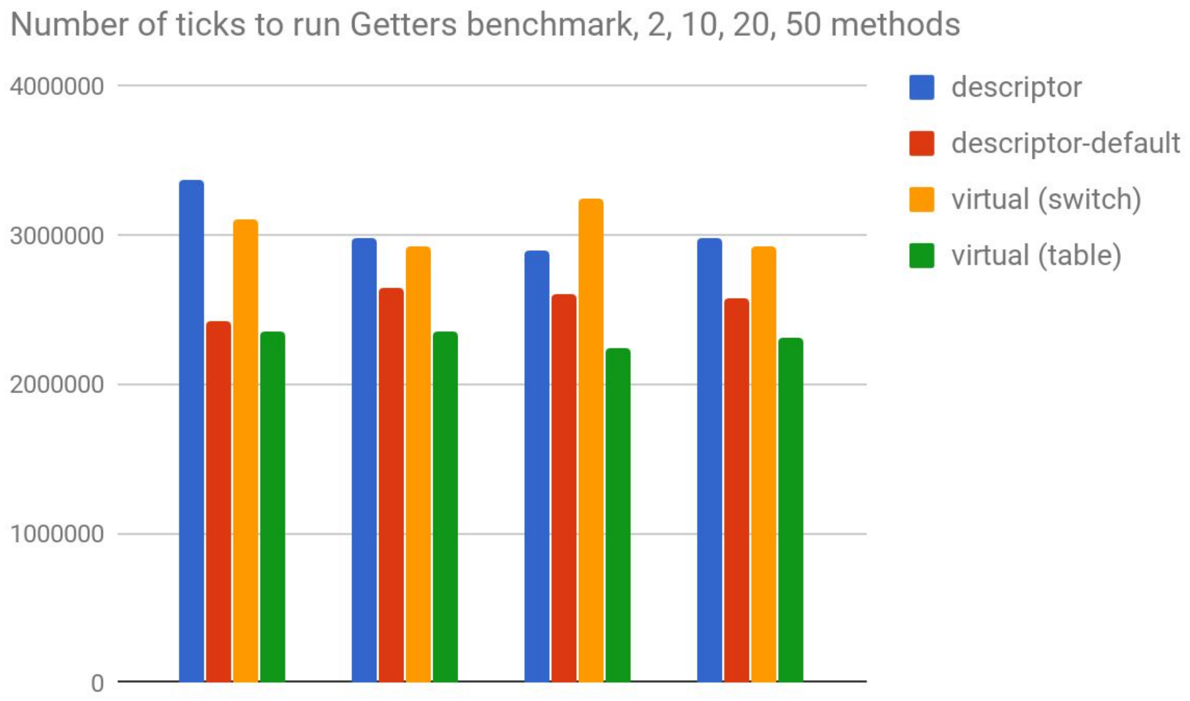
Array of Virtual Class Pointers

- Main benefits
  - Never nullptr
  - Cleaner code
- Performance boost
  - Fast to fetch property classes and implemented functions (according to the micro benchmark)

Array of Virtual Class Pointers



Array of Virtual Class Pointers



Current Status

Ribbon in the Parser

- Implement API class functions for parsing
- Completed:
  - all longhand properties, except fast-path
  - all shorthand properties
- Remaining
  - Style rule

Style Building

- Implement API class functions to build style rules
- Completed generated StyleBuilderFunctions
  - Generated class functions in header files
- Working on custom StyleBuilderFunctions
  - Generated class functions in .cpp files

CSSPropertyMetadata

- CSSPropertyMetadata contains metadata about each property, e.g whether it's runtime-enabled, interpolable, inherited, or supports percentages.
- Implement API class functions to return metadata value or state (true/false).
- Almost complete for all properties.

Next Steps

Next steps

- Ribbon in the parser
- Style building
- Future work
  - StylePropertyShorthand
  - StylePropertyMetadata / CSSProperty
  - ComputedStyleCSSValueMapping
  - And more
  - Longer term: restructure API classes for greater code sharing

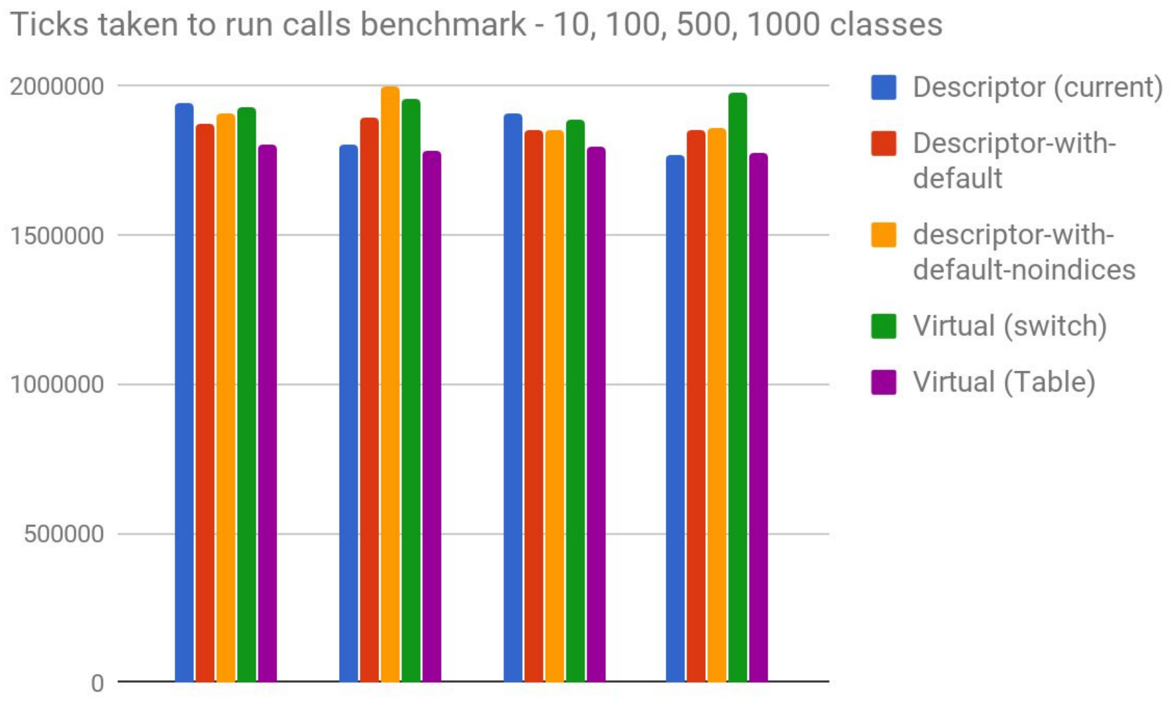
Team

- Bugs Nash (bugsnash@)
- Jia Meng (jiameng@)
- Eddy Mead (meade@)
- Renée Wright (rjwright@)

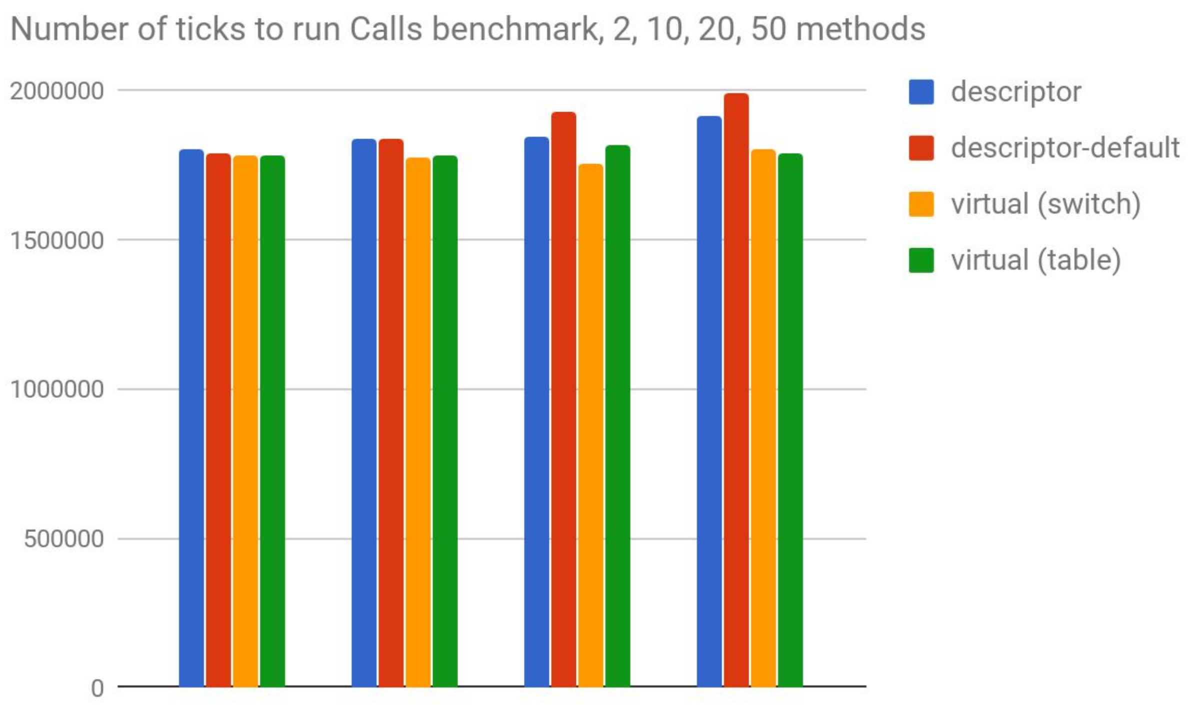
Resources

- Project Ribbon
- Project Ribbon in the Parser
- Parsing Shorthand Properties in Property APIs
- Ribbonizing StyleBuilder and StyleBuilderFunctions
- Ribbon Dispatch
- CSSPropertyMetadata Ribbonization One-Pager

Appendix: benchmark on calling functions



Appendix: benchmark on calling functions


Number of ticks to run Calls benchmark, 2, 10, 20, 50 methods
descriptor
descriptor-default
virtual (switch)
virtual (table)