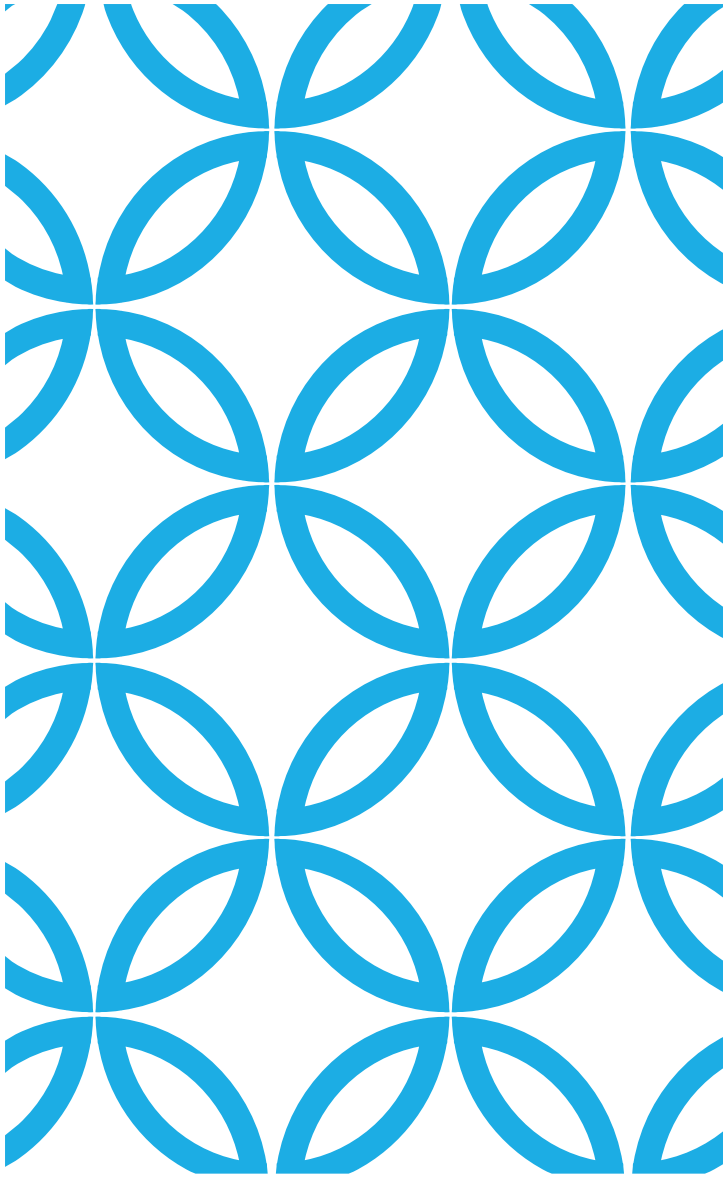


# PERFORMANCE ANALYSIS OF WEBASSEMBLY CALLS

Ross Tate

# BENCHMARK

## List Library

```
class/interface Node {
    boolean isNil();
    int getElement();
    void setElement(int element);
    Node getNext();
}

class Nil extends/implements Node {
    public Nil() { }
    public boolean isNil() { return true; }
    public int getElement() { throw …;
    public void setElement(int element) { throw …; }
    public Node getNext() { throw …; }
}

class Cons extends/implements Node {
    private int element;
    private Node next;
    public Cons(int element, Node next) { … }
    public boolean isNil() { return false; }
    public int getElement() { return element; }
    public void setElement(int element) { … }
    public Node getNext() { return next; }
}
```

## Sort Client

```
public static void sort(Node head) {
    while (!head.isNil()) {
        int element = head.getElement();
        Node old = head;
        Node node = old.getNext();
        while (!node.isNil()) {
            int value = node.getElement();
            if (value < element) {
                node.setElement(element);
                old.setElement(value);
                old = node;
            }
            node = node.getNext();
        }
        if (old == head) {
            head = head.getNext();
        }
    }
}
```

# PERFORMANCE

Median ms of 5 sequential runs on pseudorandomly generated list with 10000 elements


380


660 1090 3040

640


1970 1760 1700

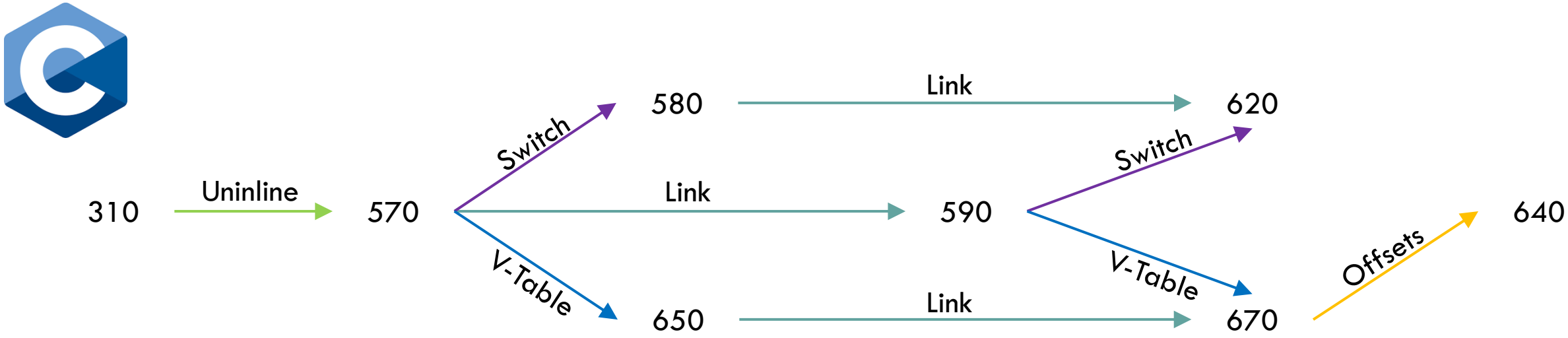
# OPTIMIZED BENCHMARK

## List Library

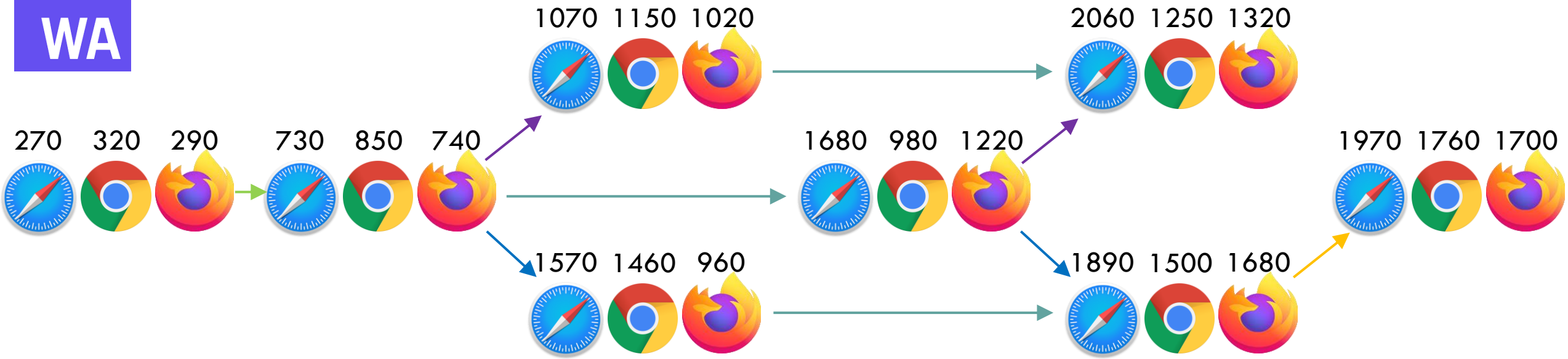
```
struct Node { unsigned int class; };

#define NIL 0
struct Nil { Node base; };
Node* newNil() {…}

#define CONS 1
struct Cons {
    Node base;
    unsigned int element;
    Node* next;
};
Node* newCons(unsigned int element,
              Node* next) {…}
```

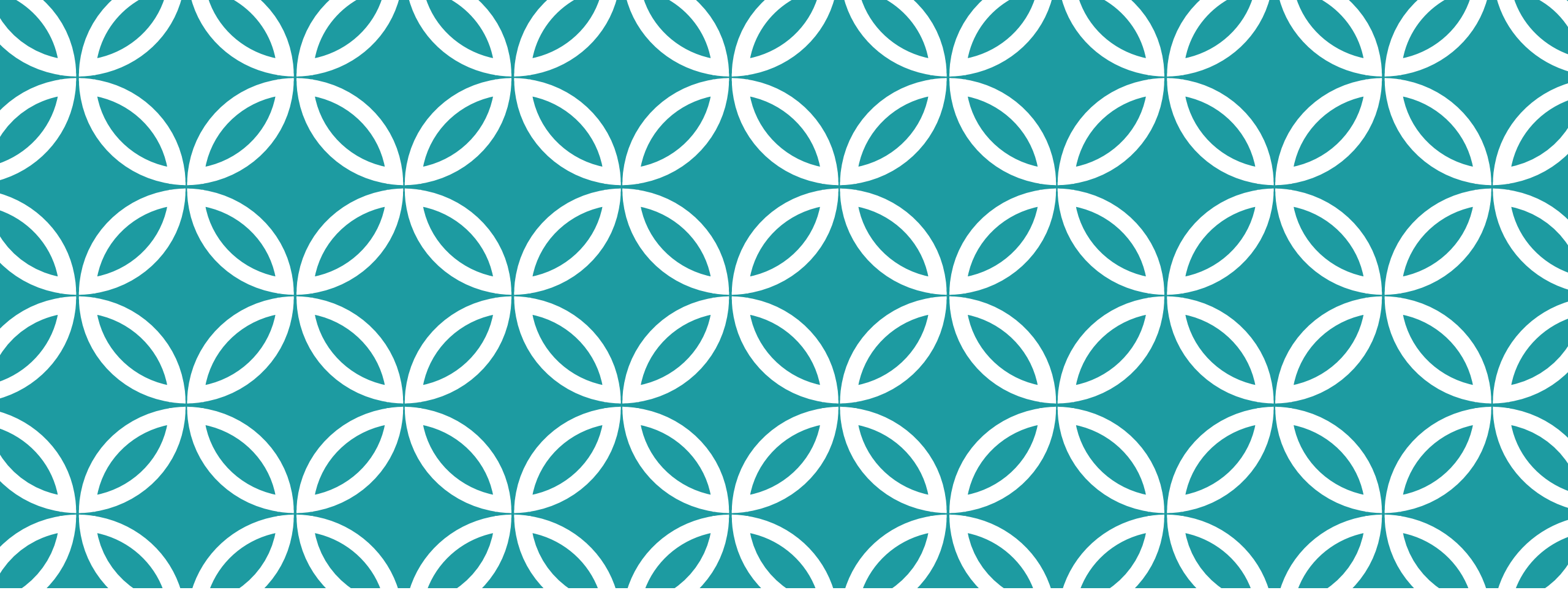
## Sort Client

```
void sort(Node* head) {
    while (head->class != NIL) {
        Cons* old = (Cons*)head;
        unsigned int element = old->element;
        Node* node = old->next;
        while (node->class != NIL) {
            Cons* cons = (Cons*)node;
            unsigned int value = cons->element;
            if (value < element) {
                cons->element = element;
                old->element = value;
                old = cons;
            }
            node = cons->next;
        }
        if (head == (Node*)old) {
            head = old->next;
        }
    }
}
```




# BREAKING DOWN PERFORMANCE





# TAKEAWAYS

# IMPROVE SWITCHING

Both br_table and cascading branches are oddly slow

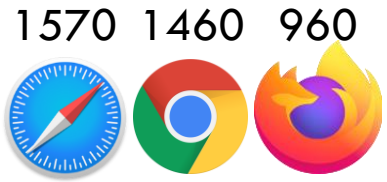
When doing whole-program compilation:

- Compile class/interface methods to use switching rather than call_indirect

# IMPROVING INDIRECT CALLS

Firefox performs much better than Chrome and Safari

- Chrome (and Safari?) uses caller-side signature checking
- Firefox uses callee-side signature checking




Callee-side signature checking is more expressive

- #1408: Allow JS functions to be directly added via table.set
- Efficient interface methods, closure invocation, method_missing

Avoid baking-in caller-side signature checking (until evaluated)

- Typed function references likely have no performance benefit
- Downcasting forces caller-side representation and semantics

# IMPROVING SEPARATE COMPILATION

Change to traditional compilation model

- Instance objects are too inefficient at this scale
- Use linking techniques to still provide fast loading

Avoid expecting instantiation to be very cheap

- E.g. each Web Worker having its own instance of a module
  - Essentially using instance globals as thread-local storage
  - Instead, design a feature for thread-local storage