For Better DOM Code

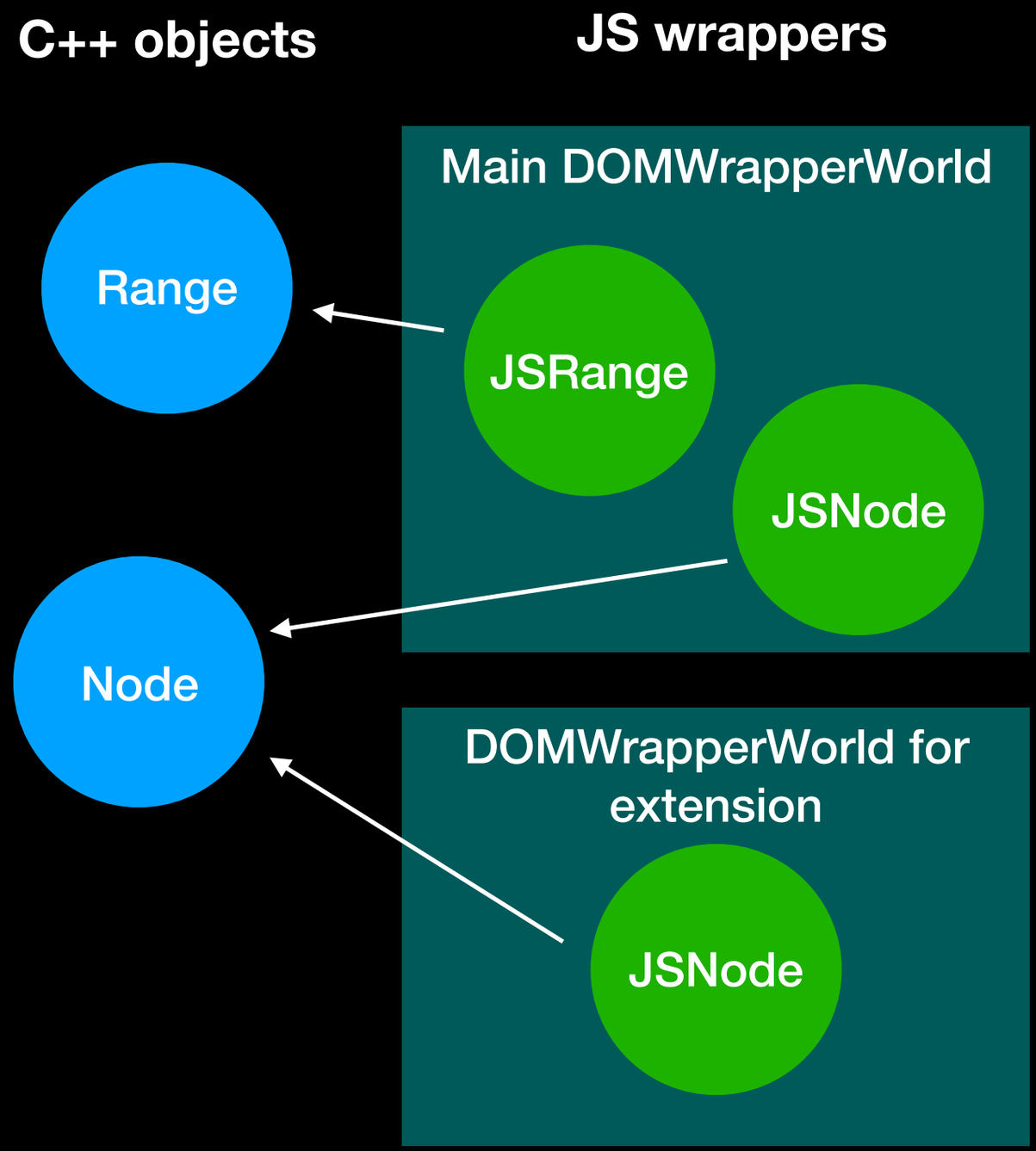
Ryosuke Niwa

C++ and JS Objects

- C++ objects define behavior

- JS wrappers expose them to JavaScript

- Injected scripts have their own DOM wrapper “world”.



C++ and JS Objects

- toJS(node) to get a JS wrapper

- toWrapped(node) / jsNode.wrapped() to get C++ object

- Cache main world’s wrapper via ScriptWrappable

Lifecycle of DOM Objects

- JS wrapper keeps C++ object alive
  - Ref<> in JSDOMWrapper

- Two ways to keep JS wrappers alive
  - Visit children
  - Reachable from Opaque Roots

Common misconception

C++ objects do NOT keep their JS wrappers alive by default

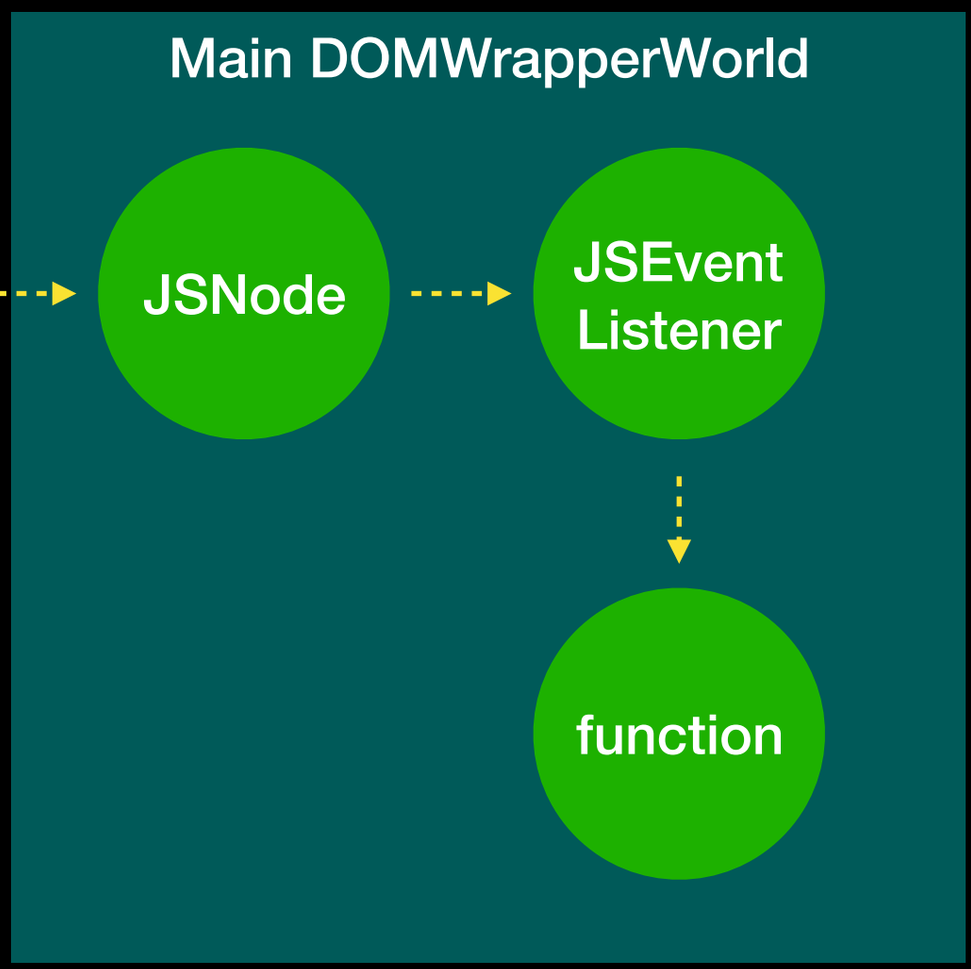
```
class Some : RefCounted<Some> {
    ...
    Ref<Other> m_other; // ← JSOther will still go away
}

class Other : RefCounted<Other> { }
```

Lifecycle: Visit Children

- JSCustomMarkFunction in IDL

- Add JS*::visitAdditionalChildren in JS*Custom.cpp

- Visit JS object kept by WebCore

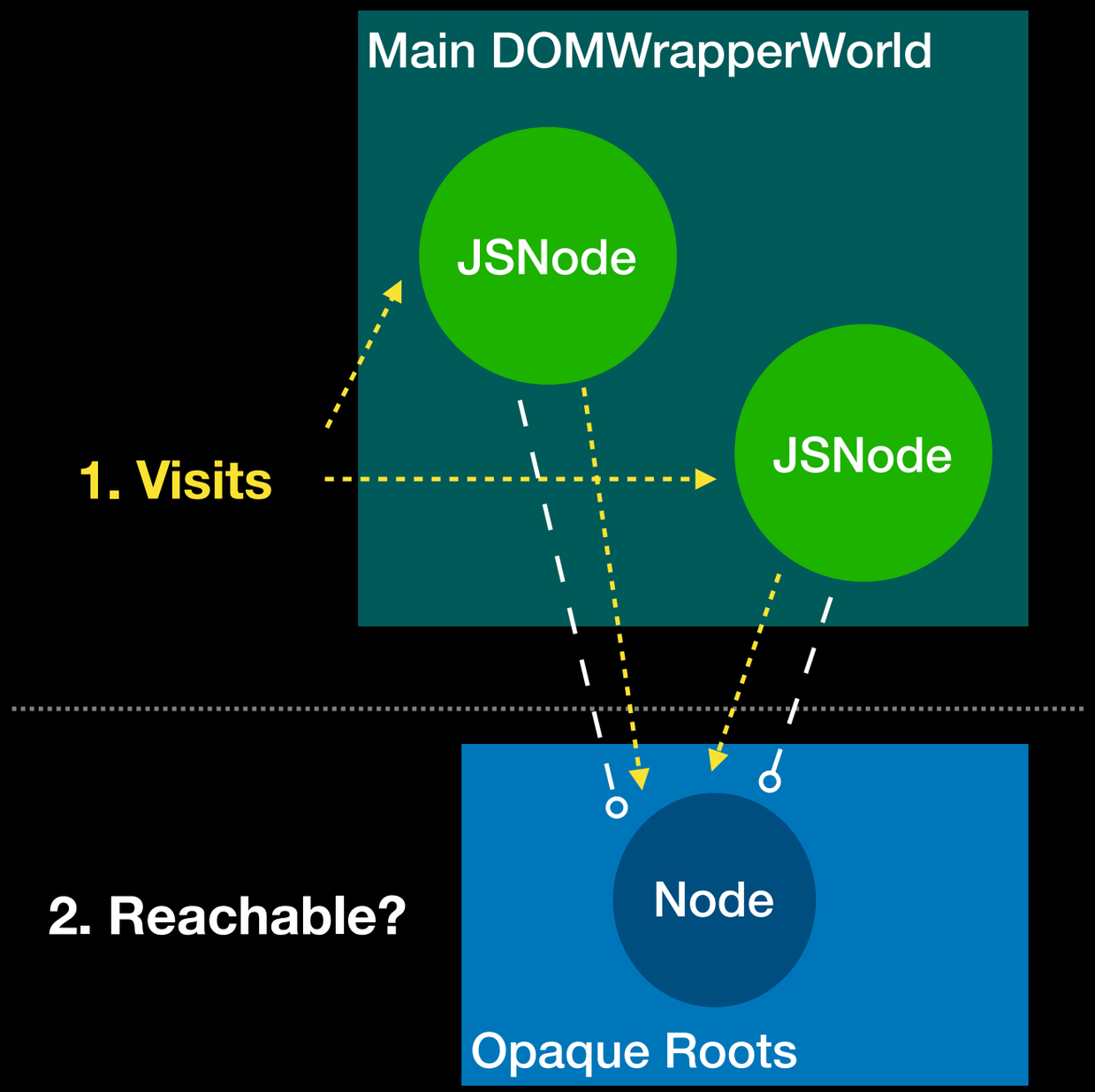


Lifecycle: Opaque Roots

- GenerateIsReachable=Impl* or CustomIsReachable in IDL

- addOpaqueRoot in visitAdditionalChildren

- JS*::isReachableFromOpaqueRoots



Lifecycle: Concurrency

- Visiting & opaque root checks happen in non-main threads

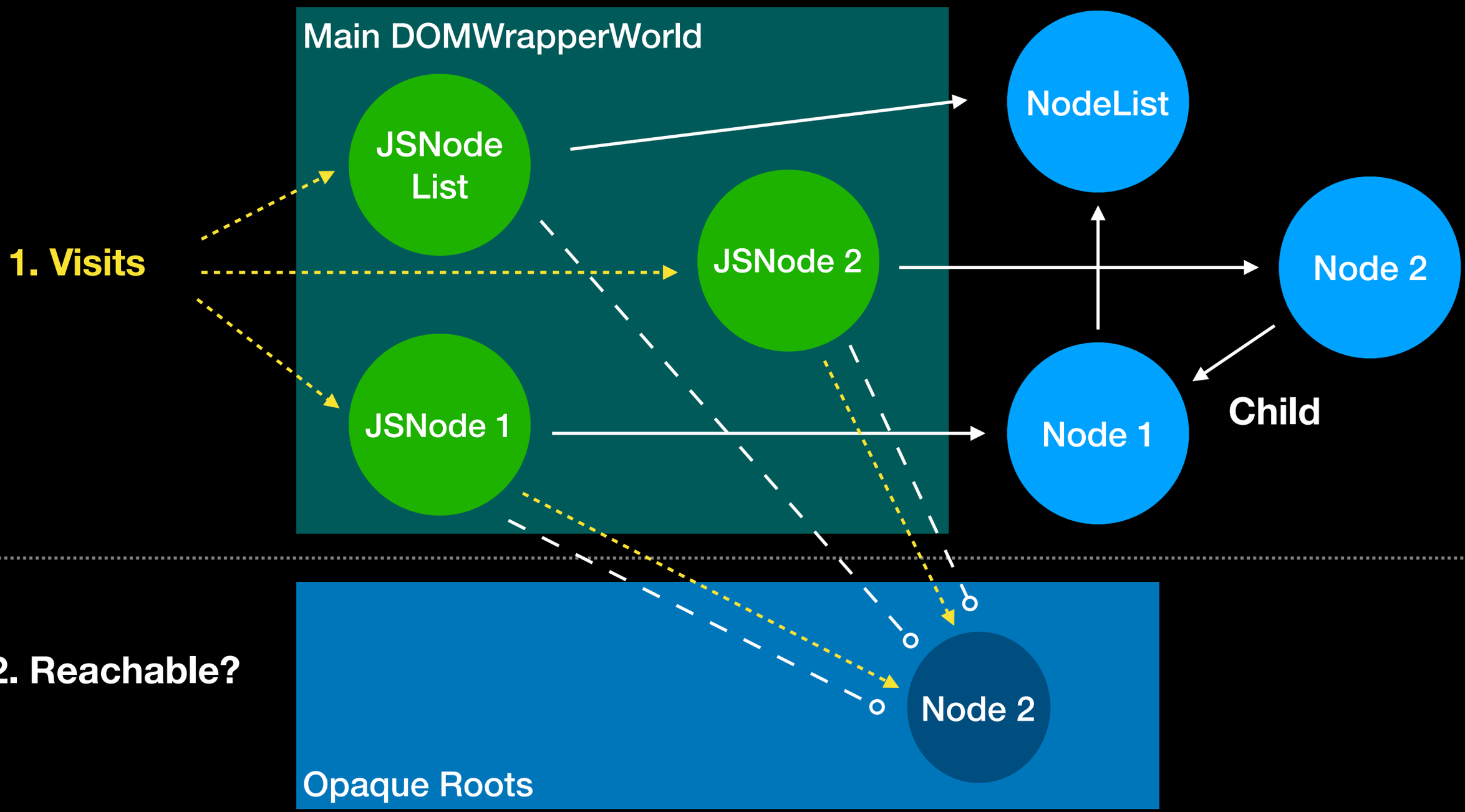
- Can’t make createWeakPtr or ref / deref RefCounted objects

- Can’t look up HashMap

Lifecycle: Common Cases

- Keeping JS object alive → Visit Children
  - Store JSC::Weak<JSC::JSObject>
  - ActiveDOMCallback for callbacks
- C++ object relationship → Opaque Roots
  - Agree on opaque root; typically root Node
  - Write thread safe code to get opaque root

Lifecycle: NodeLists

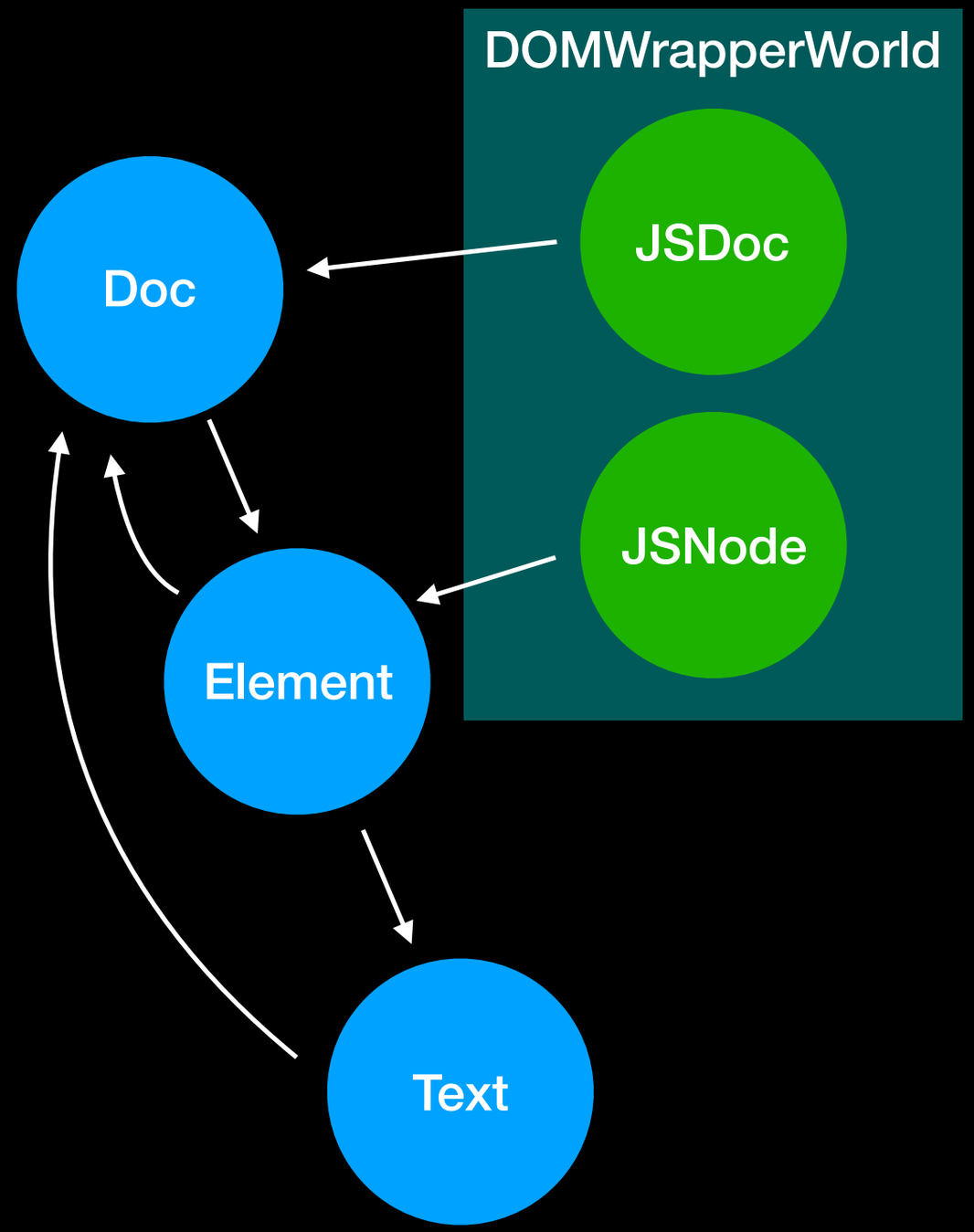


Lifecycle: DOM Nodes

- Node is alive if it has refCount > 0 or has parent node

- Node increments Document's m_referencingNodeCount

- Document is alive if refCount > 0 or m_referencingNodeCount > 0

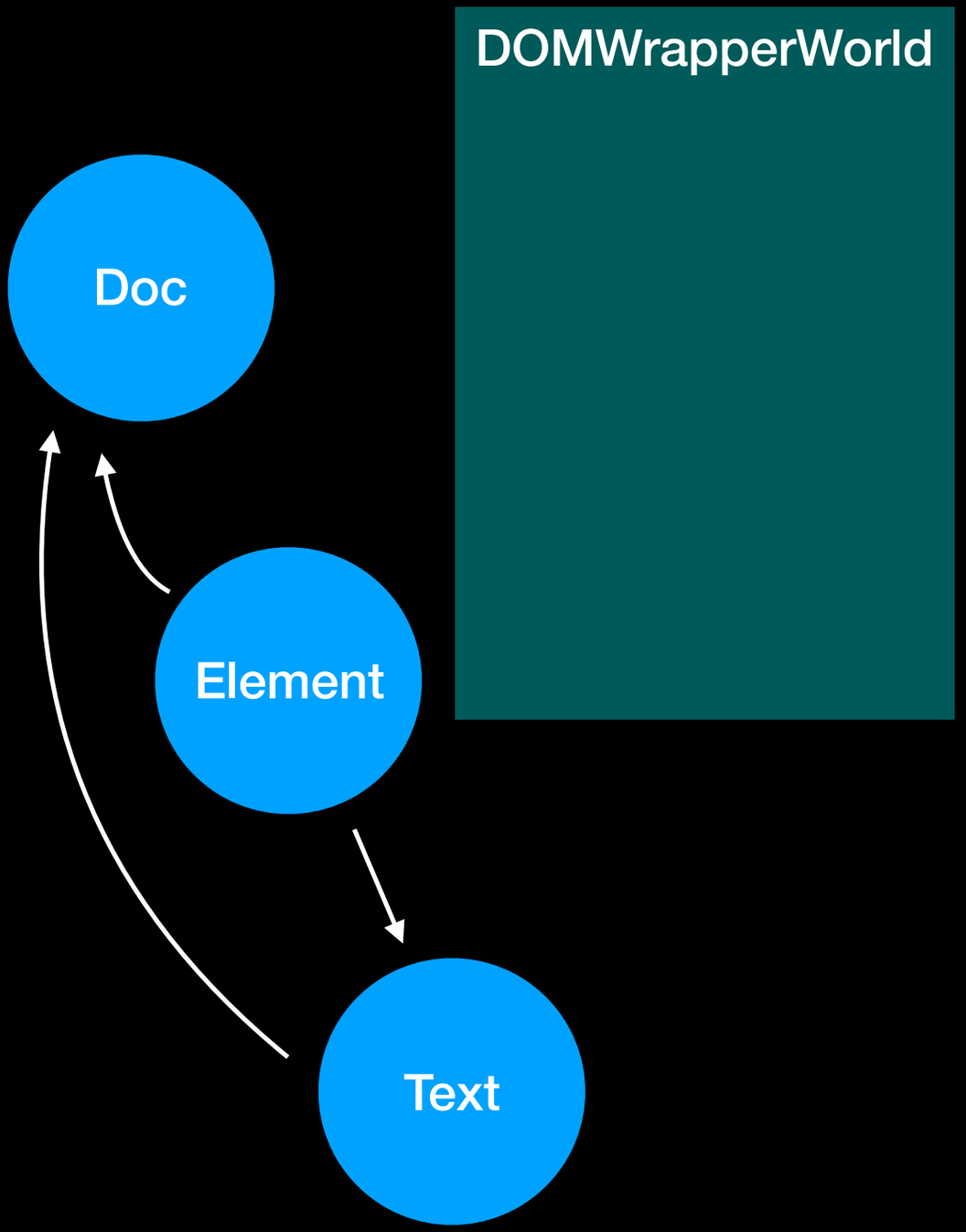


Lifecycle: DOM Nodes

- Node::removedLastRef on Element

- ContainerNode::removeDetached Children in ~ContainerNode

- Turn into flat linked list in deletion queue



Lifecycle: DOM Nodes

- Document::removedLastRef() must clear any Ref / RefPtr to Node
- Not safe to traverse DOM tree during destruction

Node Insertion & Removal

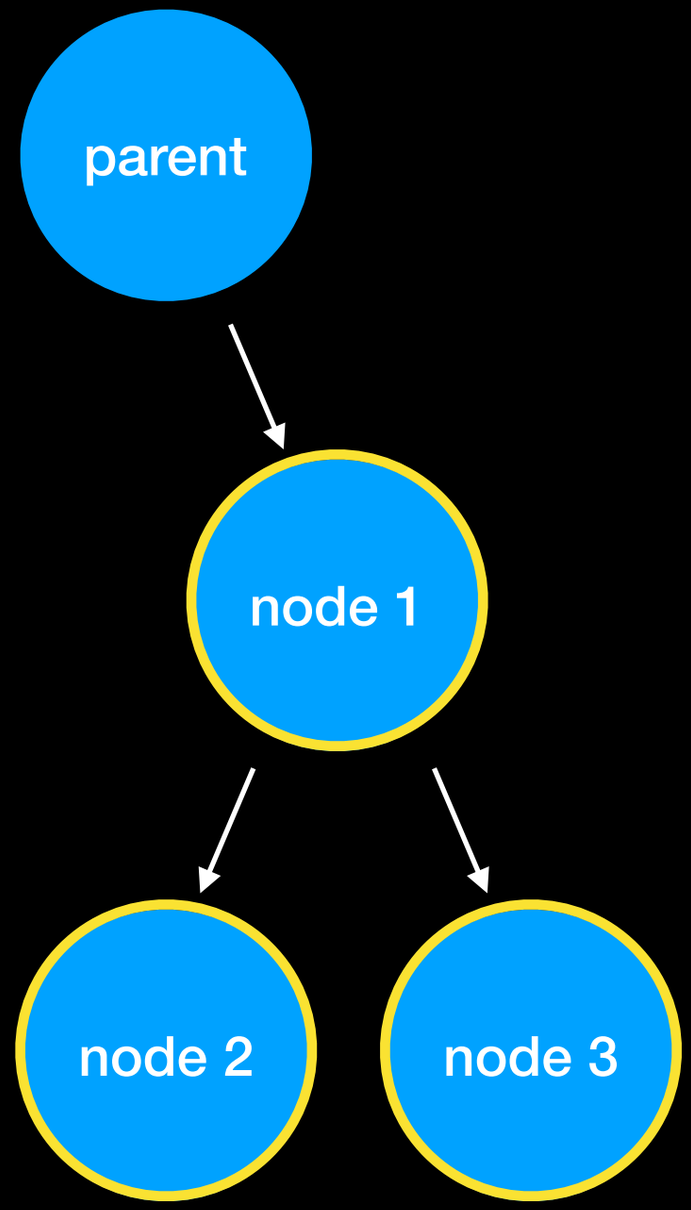
- Node::insertedIntoAncestor / removedFromAncestor
  - Called whenever node's ancestor changes
  - Either "this" or its ancestor got inserted or removed
  - Don't assume tree scope or document change

- No script execution in insertedIntoAncestor or removedFromAncestor
  - Will hit release assertion
  - Use didFinishInsertingNode instead

Node Insertion Order

- insertedIntoAncestor called in tree order

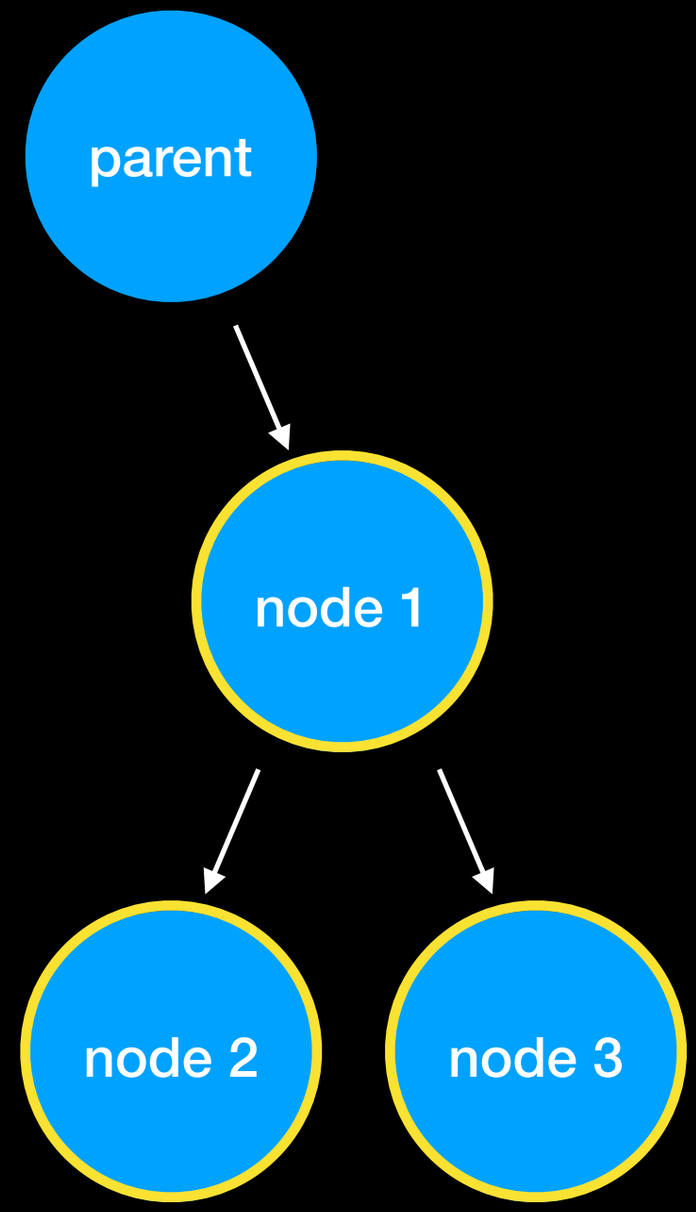
- Only talk to nodes earlier in tree order



Node Removal Order

- removedFromAncestor called in tree order

- Only talk to nodes earlier in tree order

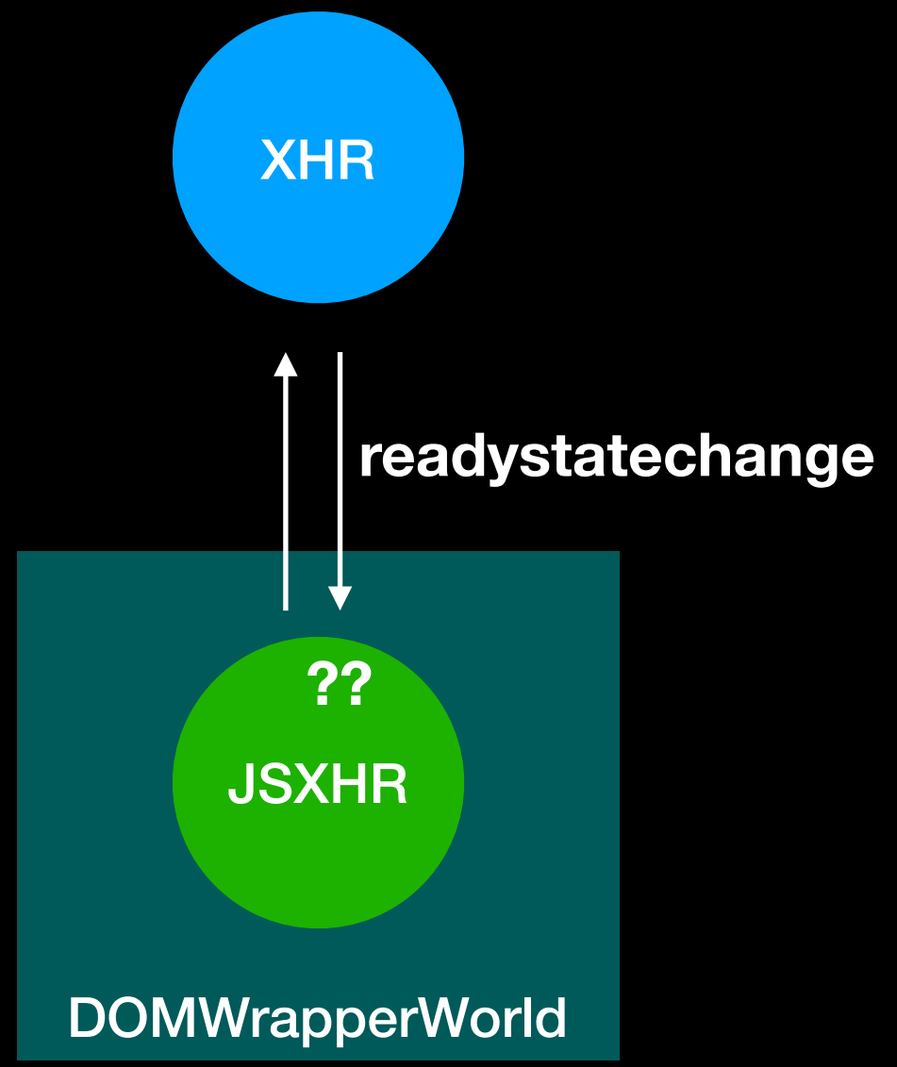


Lifecycle: Delayed Use

- Asynchronous use of “this” - XHR, media, …
  - Make “this” ActiveDOMObject
- Asynchronous use of Node - MutationObserver, ResizeObserver, …
  - GCReachableRef ← This is a leak!

Lifecycle: ActiveDOMObject

- Async work → dispatchEvent on this
- Reachable if hasPendingActivity is true
- Suspendable for back-forward cache



HTML5 Event Loop

- WindowEventLoop has been added

- WorkerEventLoop is coming

- WindowEventLoop is shared across documents of similar origins

Event Loop: In New Code

- Do NOT USE
- Timer / SuspendableTimer
- GenericEventQueue / GenericTaskQueue