EVERYTHING I KNOW ABOUT REACT I LEARNED FROM TWITTER

JENN CREIGHTON (@GURLCODE)
FRONTEND ENGINEER @ RENT THE RUNWAY

REACT DETECTIVE

@GURLCODE

ON A DARK & STORMY NIGHT...

Dan Abramov
@dan_abramov

"Async React" is common term for the new React capabilities (Time Slicing and Suspense) that I showed at JSConf Iceland. It's not an API. We're actually thinking of calling it "Concurrent React" because people think of different things when they hear "async".

7:33 AM - 4 Sep 2018

@GURLCODE

Dan Abramov
@dan_abramov

Concurrent React is not a set of new APIs. It's more like lifting some restrictions on what React can do. It's also the same thing as "Fiber" architecture you might've heard about. See — many names but ultimately it's about the same things we've been working on for years.

7:36 AM - 4 Sep 2018

@GURLCODE

@GURLCODE

Screen Shot 2019-02-10 at 5.00.38 PM

Dan Abramov @dan_abramov · Feb 8

JS tip: no matter which export style you prefer (default or named) or which function style you use (arrow or declaration), make sure your functions have names!

Especially important for React components to show up named in DevTools and warnings.

```
// 🔴 Default export with no name :-(
export default () => {
  return <button />;
}
```

```
// ✅ Named export (declaration)
export function Button() {
  return <button />;
}

// ✅ Named export (arrow)
export const Button = () => {
  return <button />;
};

// ✅ Defa
export def
  return <
}

// ✅ Defa
const Butt
  return <
};
export def
```

51 · 679 · 2.7K

Sunil Pai
@threepointone

ok so -
react co
bits, the
much n
comple

Screen Shot 2019-02-10 at 5.01.

Dan Abramov
@dan_abramov

Memoization seems easie
consuming component n
useMemo. Instead of imp
change bailouts based o
comparison inside a HOC

9:03 PM - 6 Feb 2019

7 Retweets 34 Likes

3 · 7 · 34

```
      return val;
    } else {
      return newValue;
    }
```

Following

will expose the
xploring larger
he flow of

Screen Shot 2019-02-10 at 4.53.01 PM

Sophie Alpert
@sophiebits

Following

Reminiscing tonight about ideas from 2014 for function components with state. Sometimes we're slow to add stuff, but that's because we want to get it right. @sebmarkbage didn't invent Hooks – the first non-class API we really liked – until 4 years later!

Returning state · reactjs/react-future@344964f
A pure version of state management with no direct

@GURLCODE

WELCOME!

Andrew Clark
@acdlite

If I never heard the term "virtual DOM" again, I'd be happy

10:47 AM - 24 Sep 2016 from Redwood City, CA

@GURLCODE

Dan Abramov
@dan_abramov

I wish we could retire the term "virtual DOM". It made sense in 2013 because otherwise people assumed React creates DOM nodes on every render. But people rarely assume this today.

"Virtual DOM" sounds like a workaround for some DOM issue. But that's not what React is.

8:52 AM - 24 Nov 2018

@GURLCODE

Dan Abramov
@dan_abramov

Myth: React is "faster than DOM". Reality: it helps create maintainable applications, and is *fast enough* for most use cases.

bigmanwalter 11 hours ago [-]

The React documentation says it so it has to be true. DOM slow, VDOM fast. Facebook is out there buying developer mindshare.

reply

* 1 point by danabramov 1 minute ago | edit | delete [-]

We don't claim that in React documentation. React can't be faster than the same DOM mutations written by hand because by definition it has to do more work.

We do think it helps to create maintainable apps though, and it is *fast enough* for practical use cases despite its immutable design. That's why we use it at Facebook.

reply

7:01 AM - 16 Mar 2017

@GURLCODE

Dan Abramov
@dan_abramov

Reconciliation takes longer than commit. That's exactly why we're making reconciliation pausable in Fiber

7:21 AM - 16 Mar 2017

<BUTTON />

```
react.CreateElement("button",null);
```

REACT ELEMENT

```
{
    type: button,
    props: { … }
}
```

Dan Abramov
@dan_abramov

React Elements are immutable objects describing the DOM. { type: 'p', props: { className: 'Paragraph' } }

10:39 AM - 14 Dec 2015

@GURLCODE

Dan Abramov
@dan_abramov

The more expensive part is calling React components recursively to learn which elements they render to, and diffing child lists

7:19 AM - 16 Mar 2017

@GURLCODE

Dan Abramov
@dan_abramov

The main difference between how Stack and Fiber reconcilers work: Fiber doesn't go deep into the JavaScript stack.

10:31 AM - 3 Dec 2016

@GURLCODE

Dan Abramov
@dan_abramov

Our goal is to make reconciliation run in chunks in idle time, but make commit phase thin so it doesn't miss a frame

7:24 AM - 16 Mar 2017

Dan Abramov
@dan_abramov

Fiber gives us architecture that is flexible enough for many cool things but it'll take a while to take advantage of it

12:35 PM - 8 Apr 2017

Sebastian Markbåge
@sebmarkbage

Fun fact: React Fiber doesn't have any JavaScript function recursion in its implementation because it reimplements the stack.

3:41 PM - 26 Sep 2016

@GURLCODE

Dan Abramov
@dan_abramov

If React is a horse then Fiber is a horse on acid.

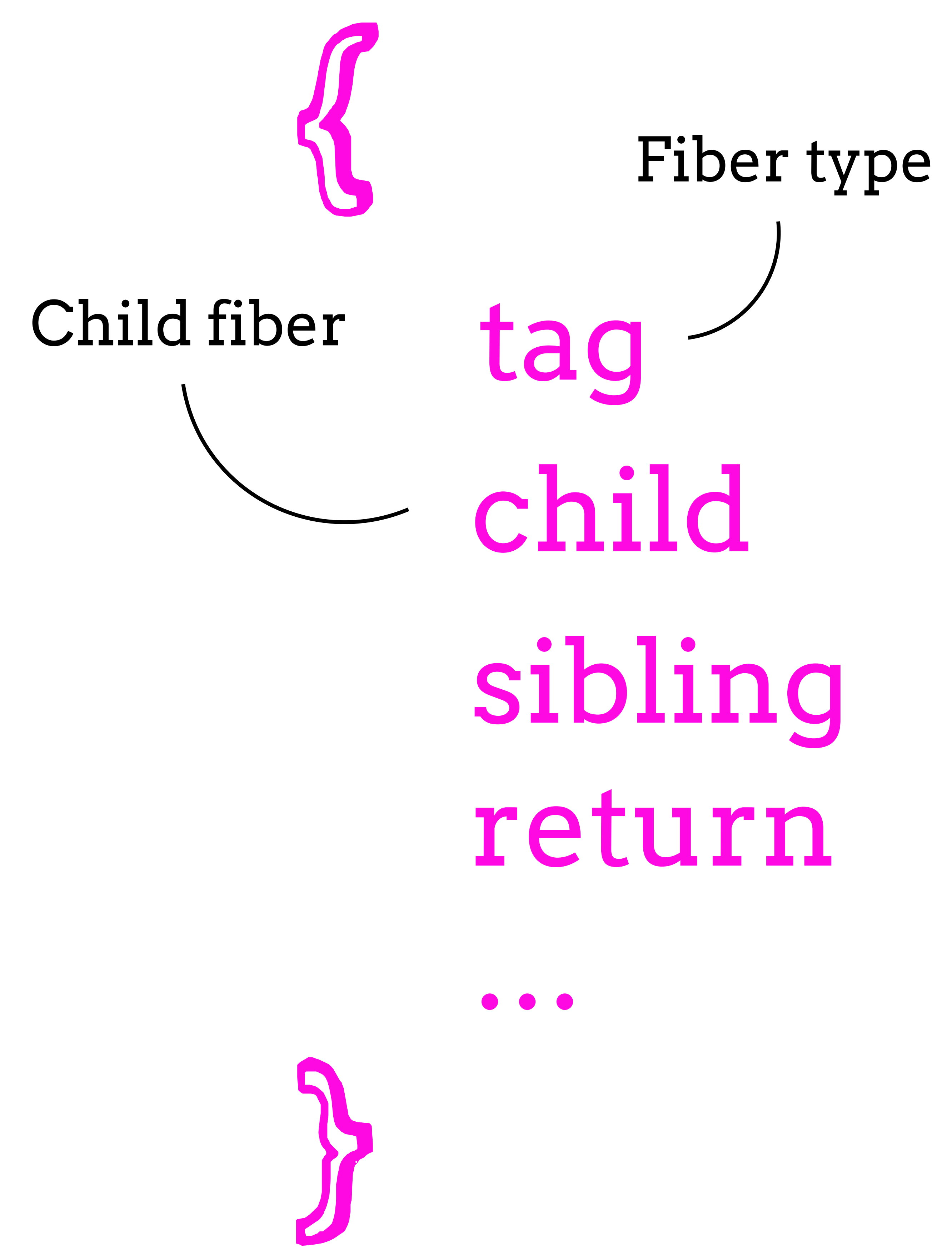
2:03 PM - 31 Oct 2016

```
{
    tag
    child
    sibling
    return
    ...
}
```

FIBER

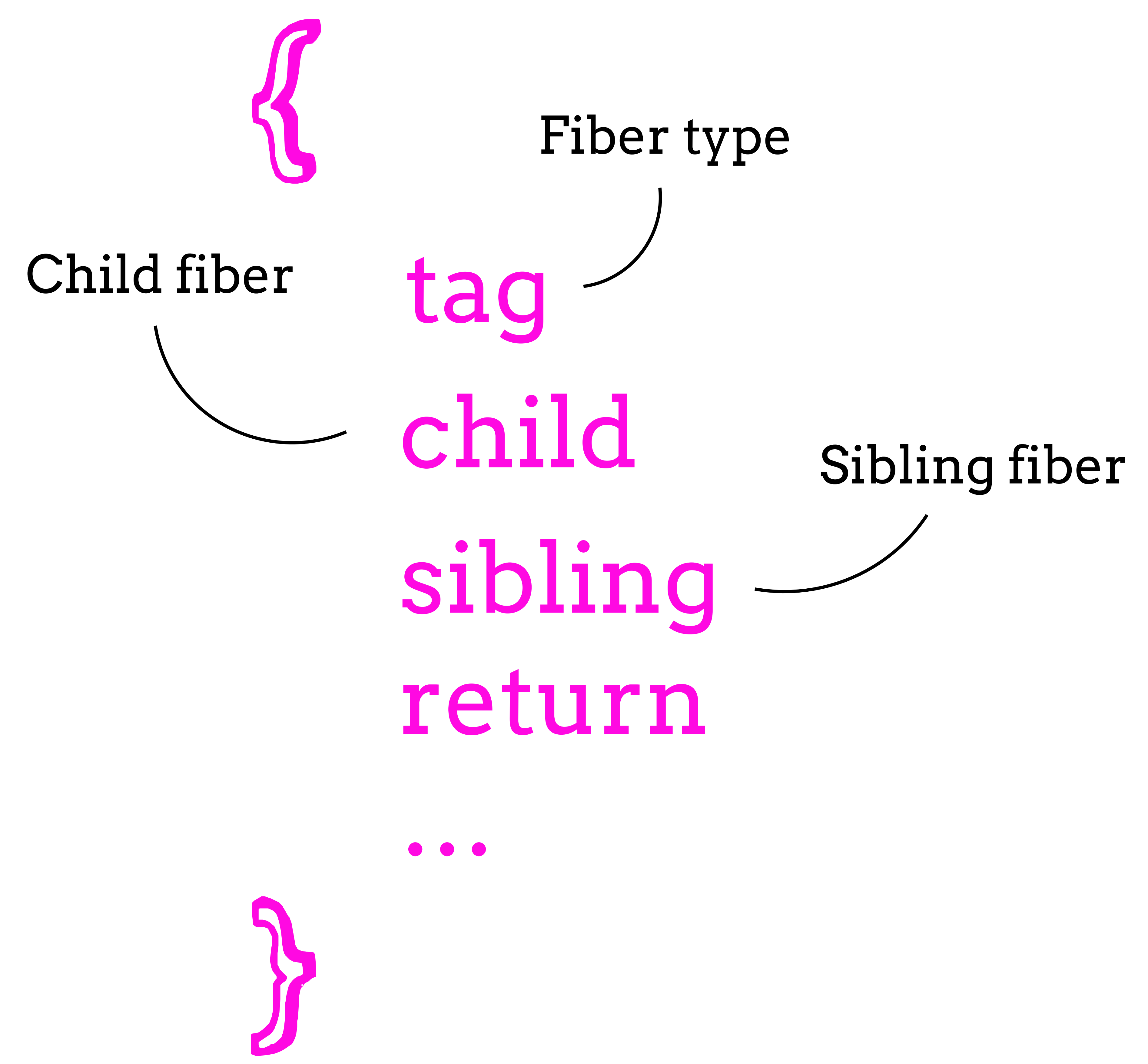
@GURLCODE

FIBER



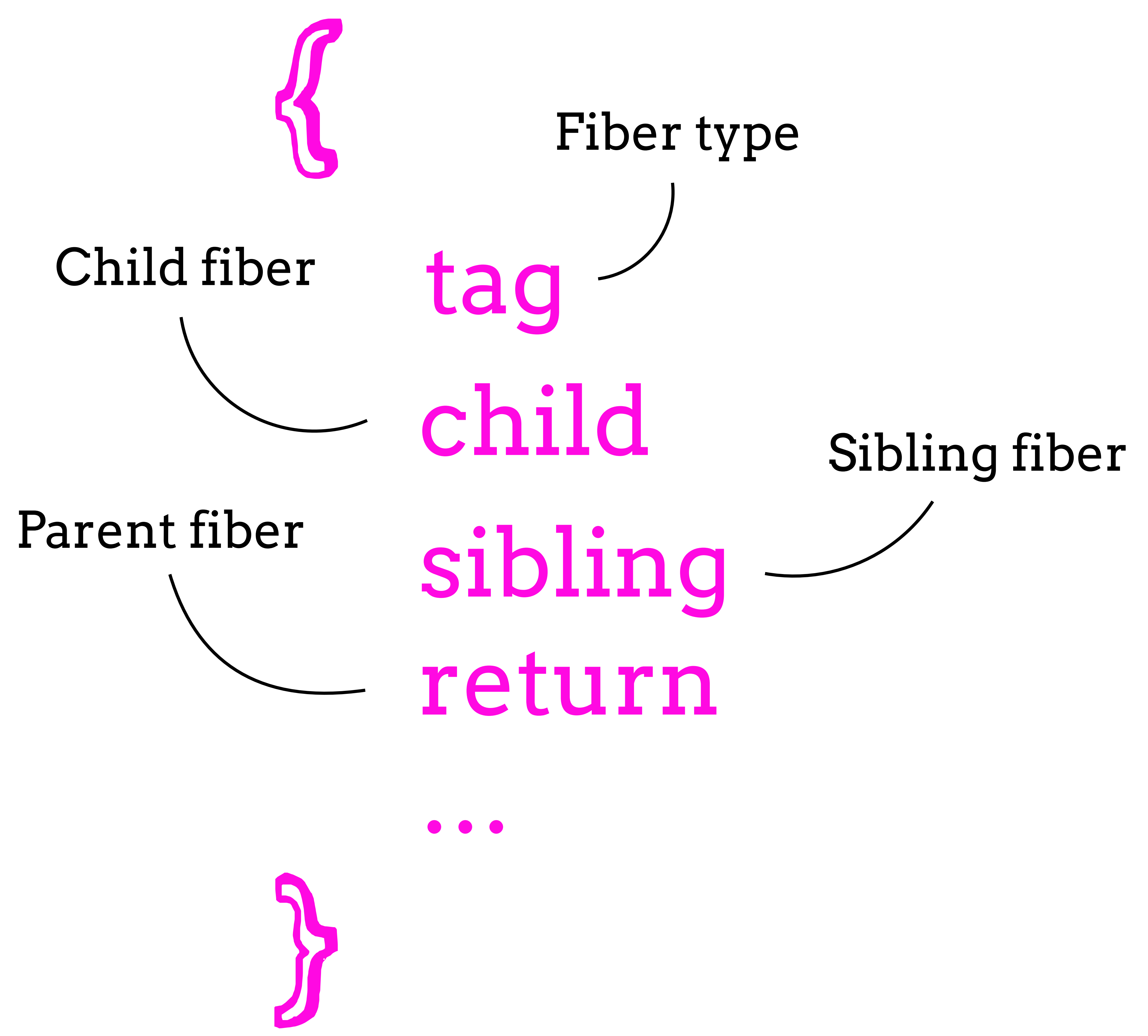
@GURLCODE

FIBER



@GURLCODE

FIBER



@GURLCODE

Fiber Principles: Contributing To Fiber #7942

sebmarkbage opened this Issue on Oct 11, 2016 · 7 comments

- If I can't use recursion, how do I traverse through the tree? Learn to use the singly linked list tree traversal algorithm. E.g. parent first, depth first:

```
let root = fiber;
let node = fiber;
while (true) {
  // Do something with node
  if (node.child) {
    node = node.child;
    continue;
  }
  if (node === root) {
    return;
  }
  while (!node.sibling) {
    if (!node.return || node.return === root) {
      return;
    }
    node = node.return;
  }
  node = node.sibling;
}
```

@GURLCODE

reactInternalInstance

Overreacted

Personal blog by Dan Abramov.
I explain with words and code.

How Are Function Components Different from Classes?

March 3, 2019 • ☕☕☕ 14 min read

They're a whole different Pokémon.

Coping with Feedback

March 2, 2019 • ☕ 3 min read

Sometimes I can't fall asleep.

Fix Like No One's Watching

February 15, 2019 • ☕ 1 min read

The other kind of technical debt.

Elements Memory Network Console »

top ▼ Filter All levels

```
> const toggle = document.querySelector('.react-toggle');
< undefined
>
```

@GURLCODE

Dan Abramov
@dan_abramov

Replying to @dan_abramov @kentcdodds and 2 others

Let's put some breakpoints in Fiber and I'll show you how it works

9:09 PM - 1 Dec 2016

ColorBox

@GURLCODE

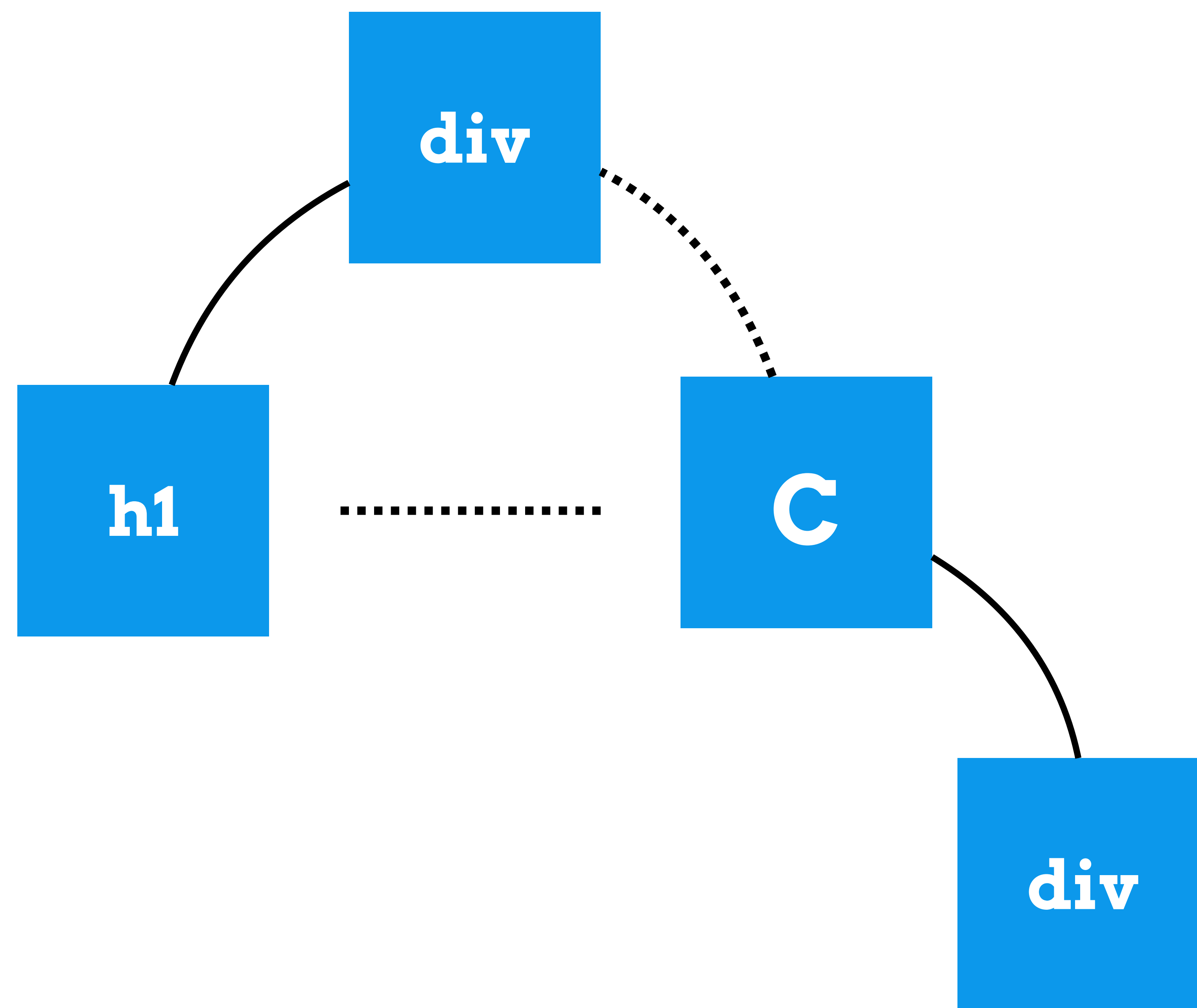
```
class ColorBox extends React.Component {
  toggleColor() {
    this.setState({
      color: this.state.color === 'blue' ? 'tomato' : 'blue',
    });
  }

  render() {
    return (
      <div
        style={{ backgroundColor: this.state.color }}
        onClick={this.toggleColor}
      >
        {this.state.color}
      </div>
    );
  }
}
```

@GURLCODE

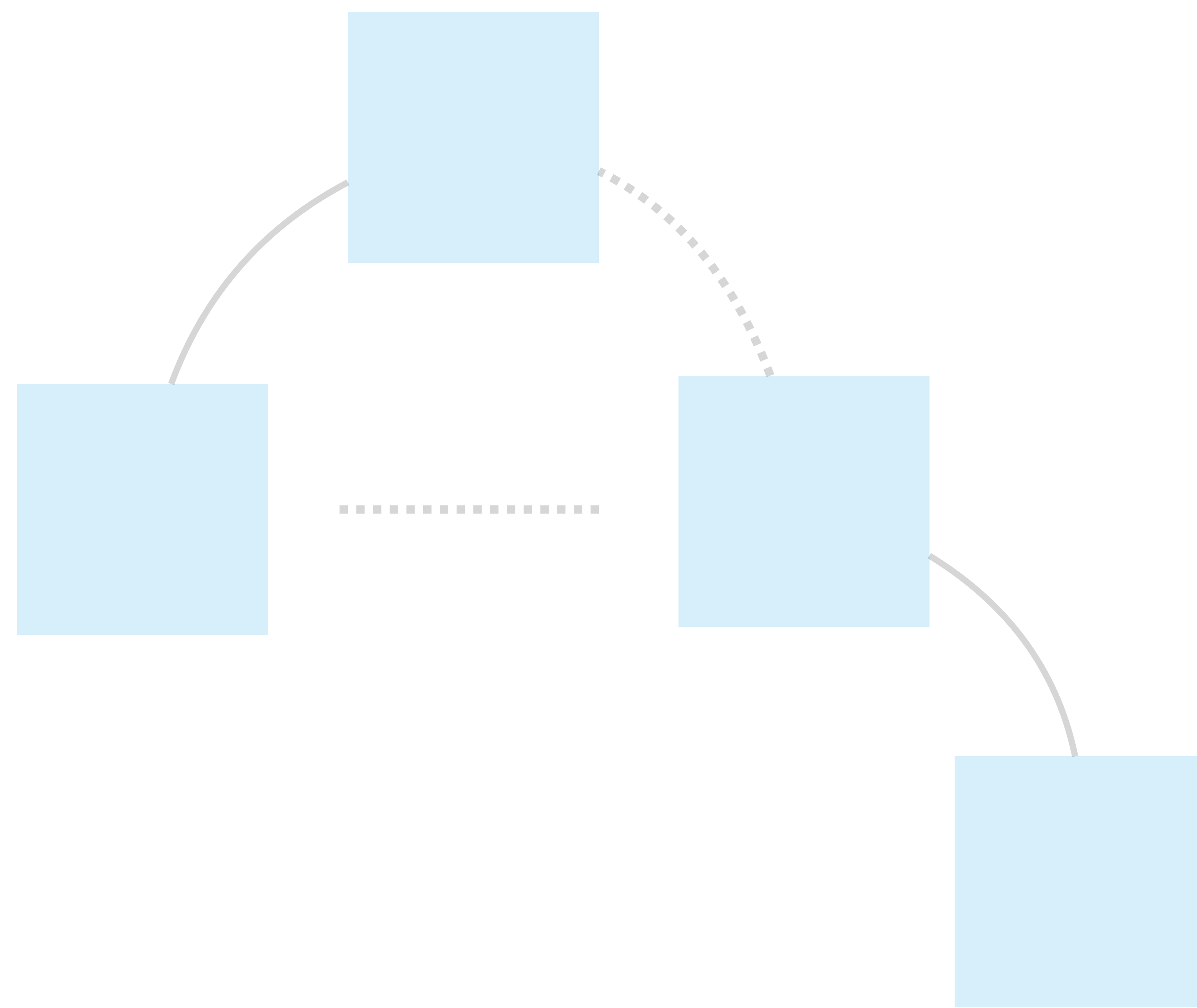
```
const container = document.getElementById('container');
const root = ReactDOM.unstable_createRoot(container);
root.render(
  <div>
    <h1>ColorBox</h1>
    <ColorBox />
  </div>
);
```

@GURLCODE



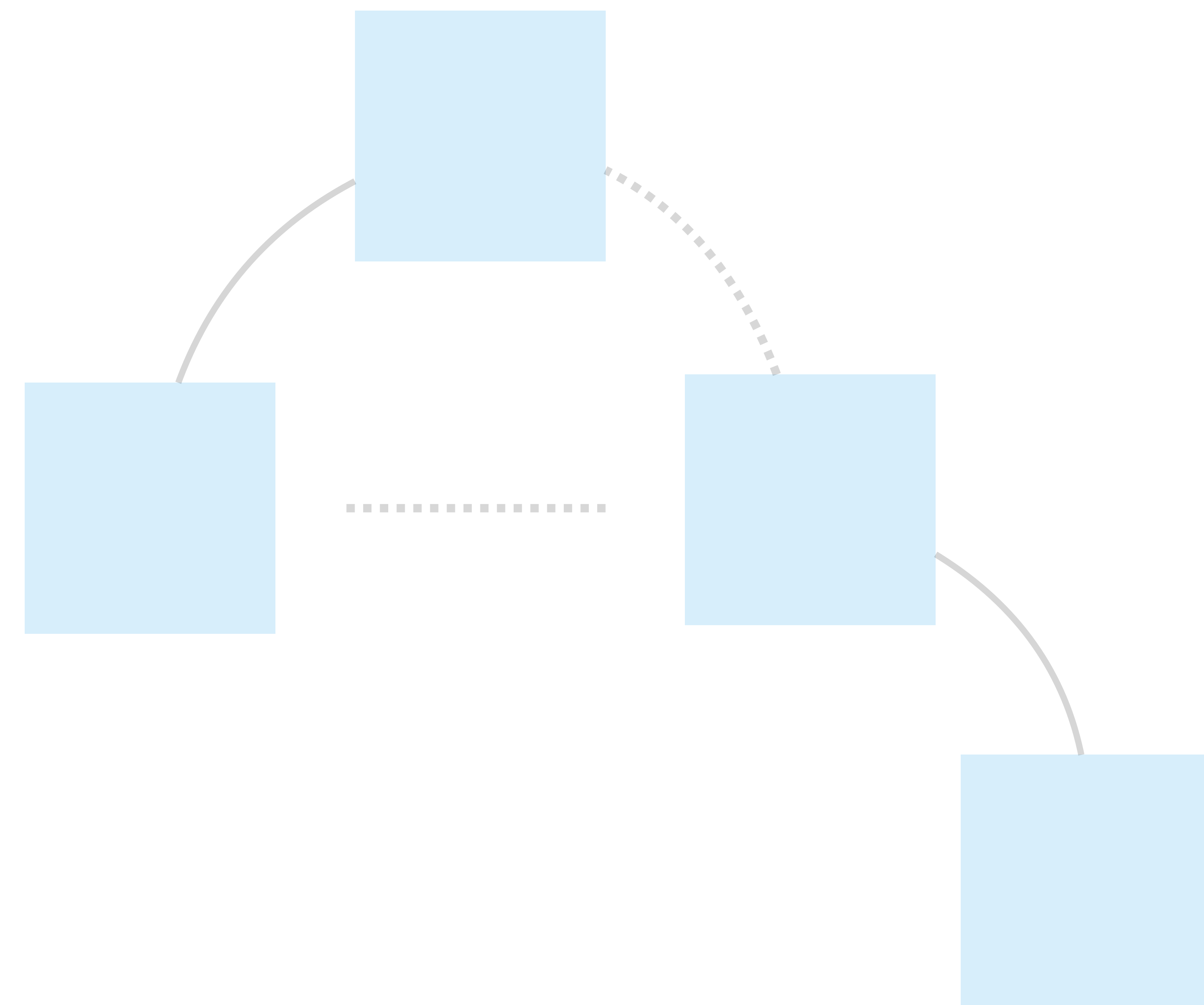
one-way
two-way

@GURLCODE



@GURLCODE

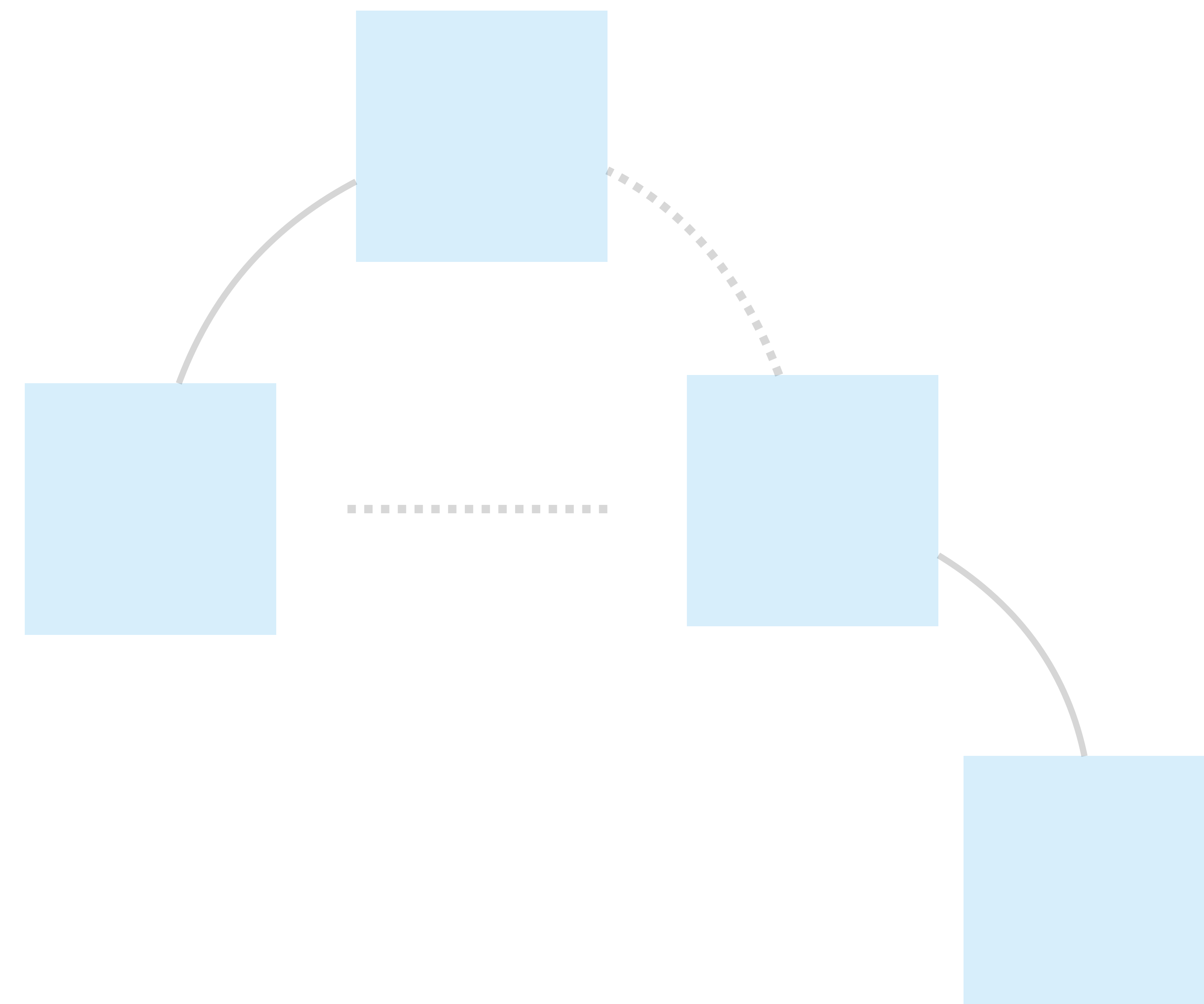
ReactDOM.unstable_createRoot(node)



@GURLCODE

new ReactRoot

ReactDOM.unstable_createRoot(node)

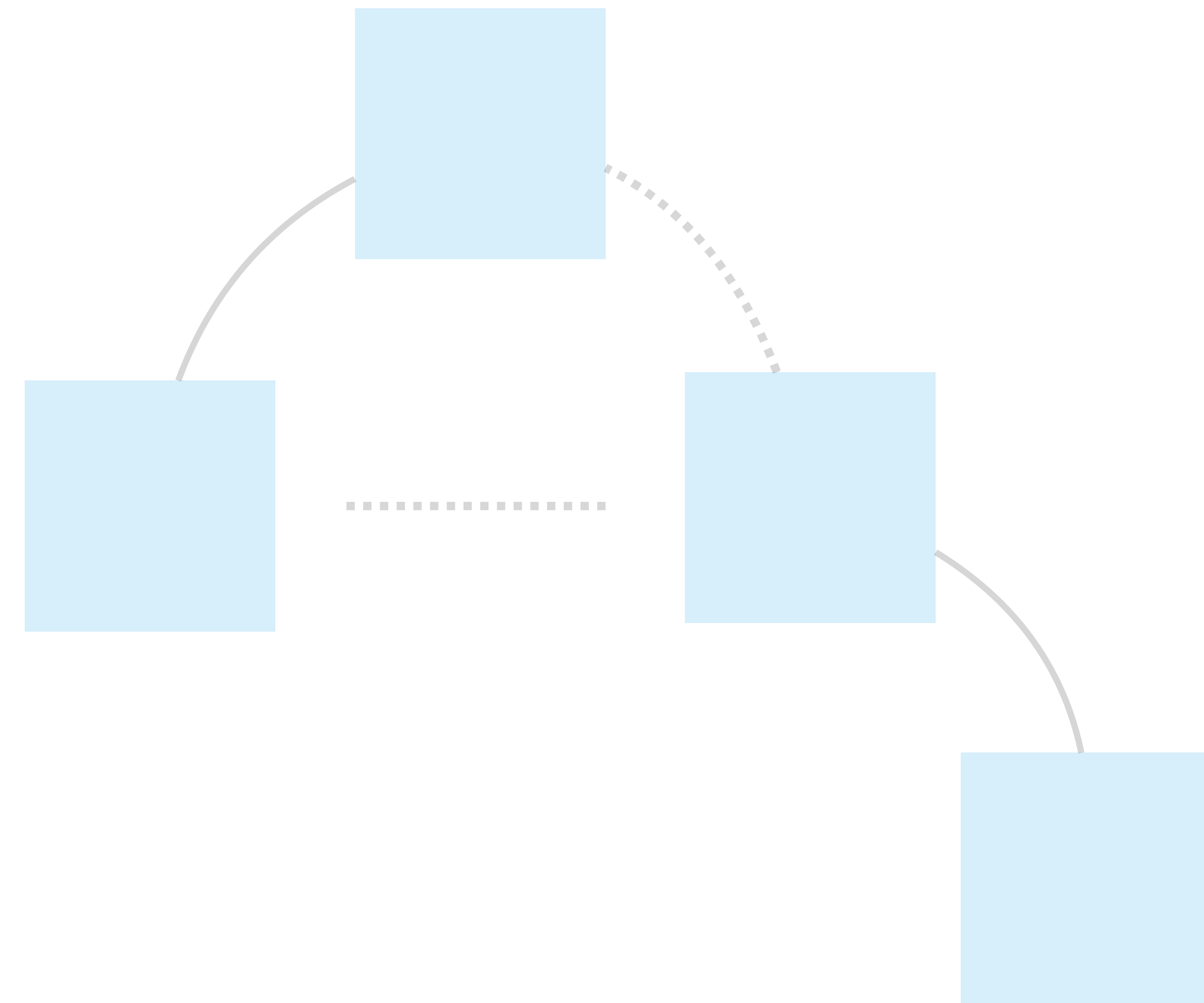


@GURLCODE

createFiberRoot

new ReactRoot

ReactDOM.unstable_createRoot(node)



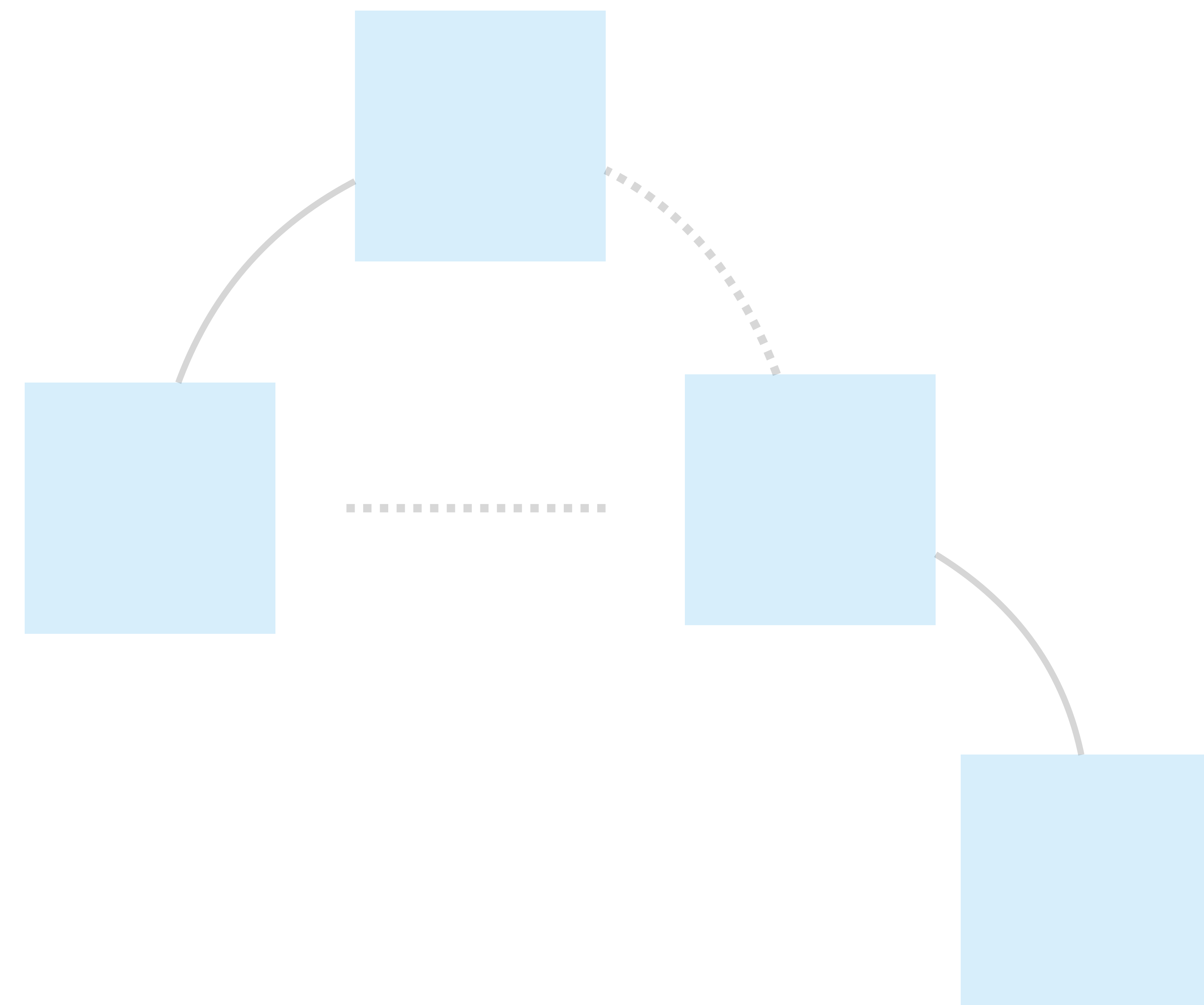
@GURLCODE

createHostRootFiber

createFiberRoot

new ReactRoot

ReactDOM.unstable_createRoot(node)



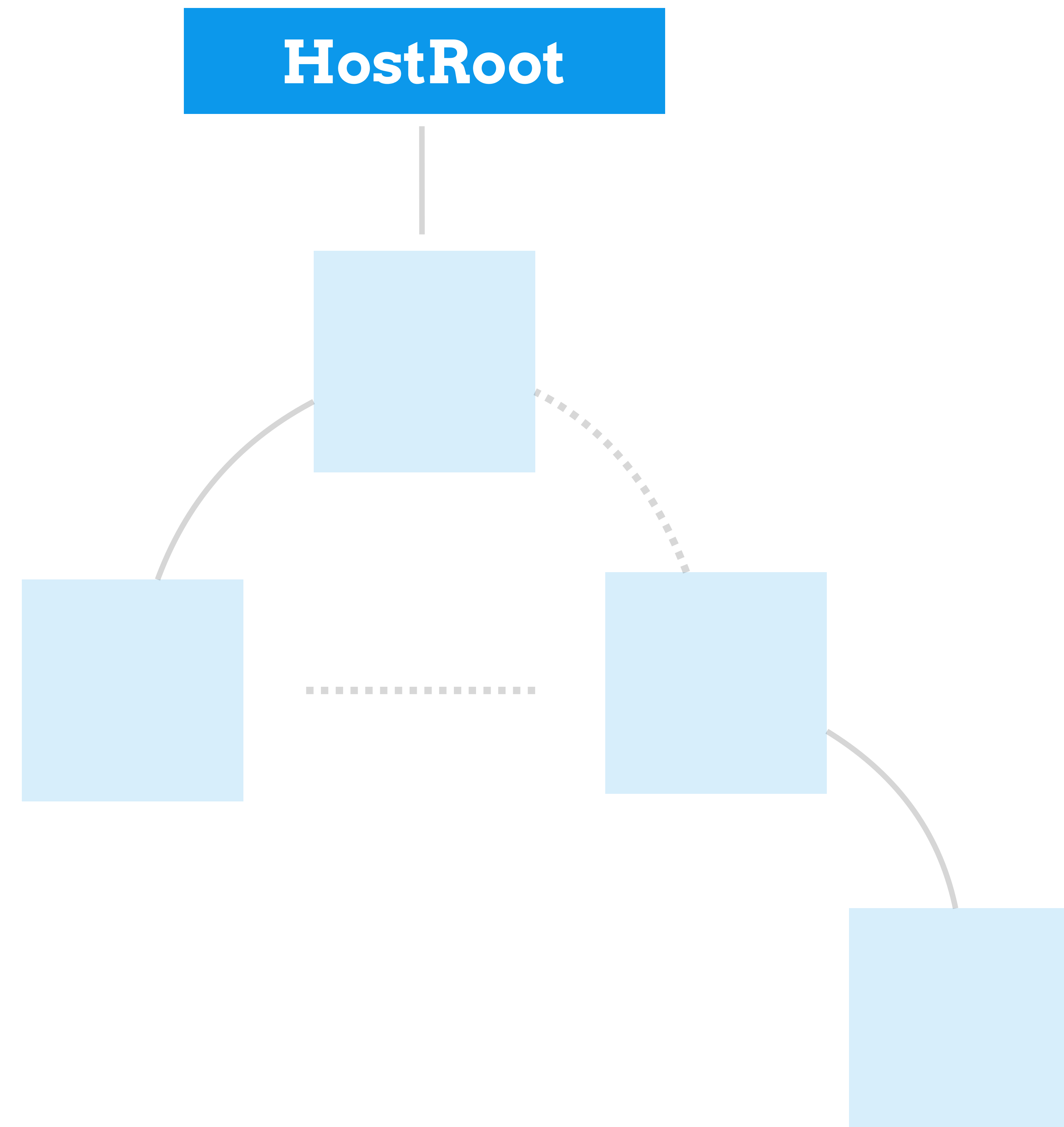
@GURLCODE

fiber { tag: HostRoot }

createFiberRoot

new ReactRoot

ReactDOM.unstable_createRoot(node)

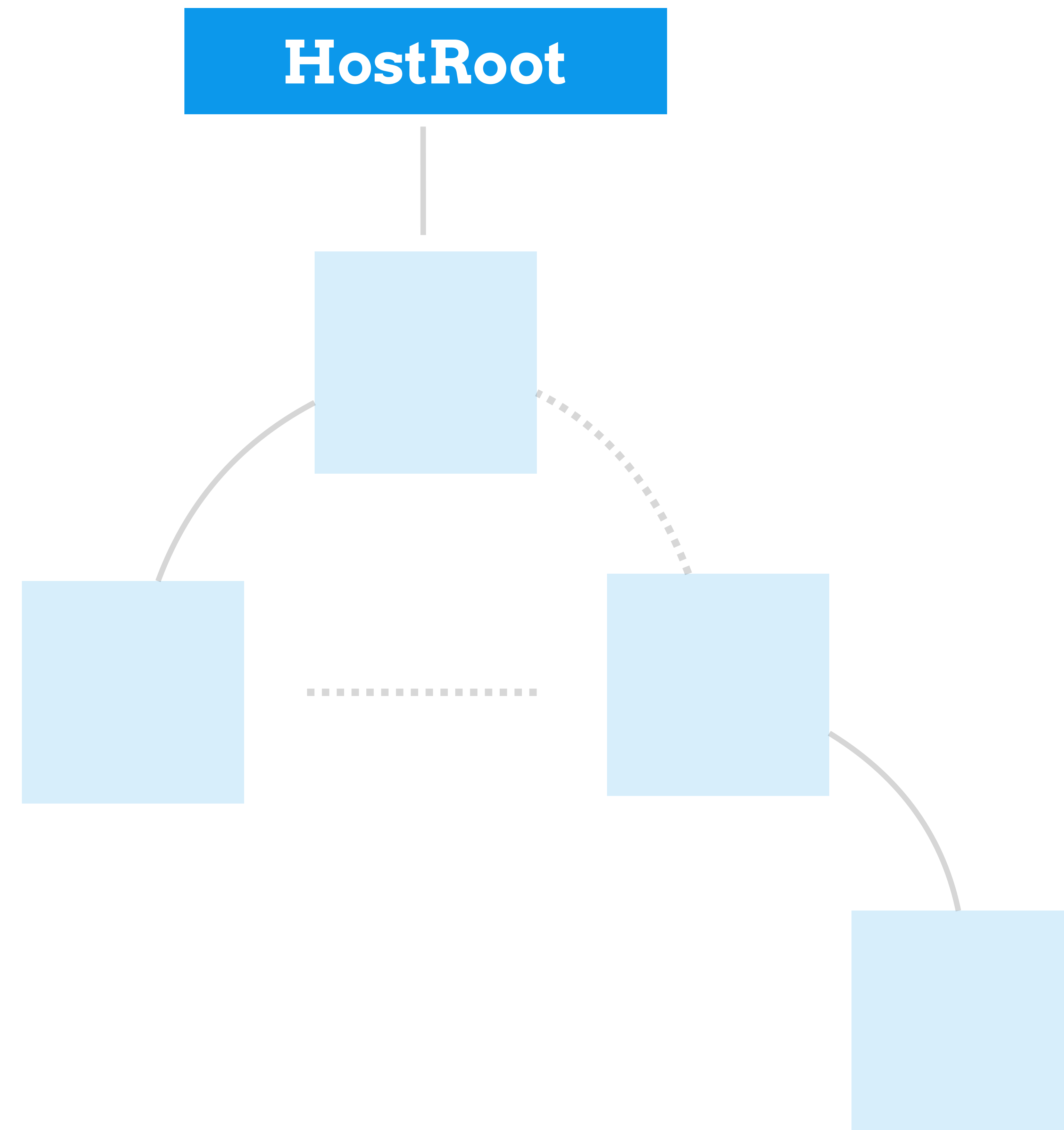


@GURLCODE

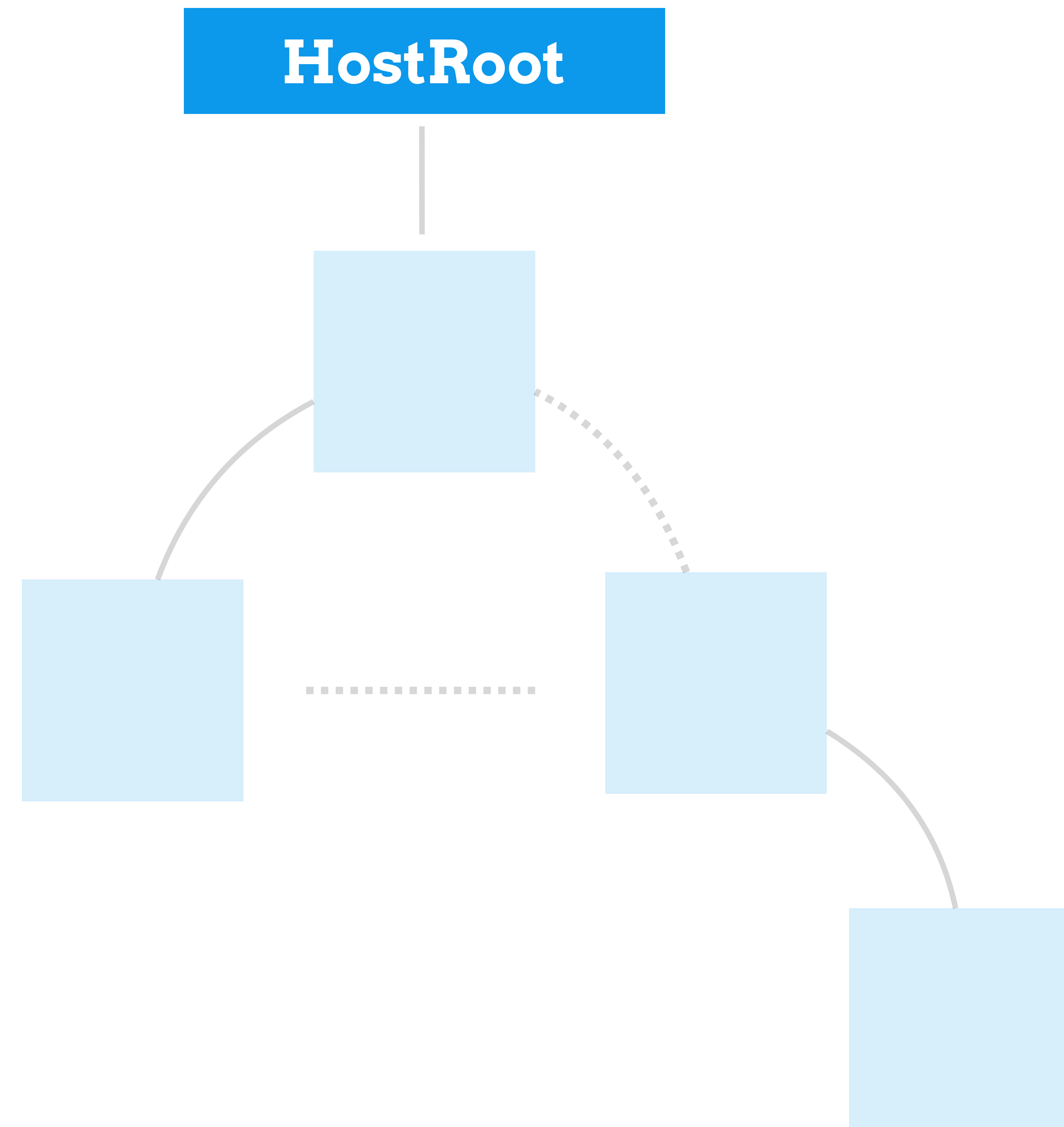
root.current = fiber

new ReactRoot

ReactDOM.unstable_createRoot(node)



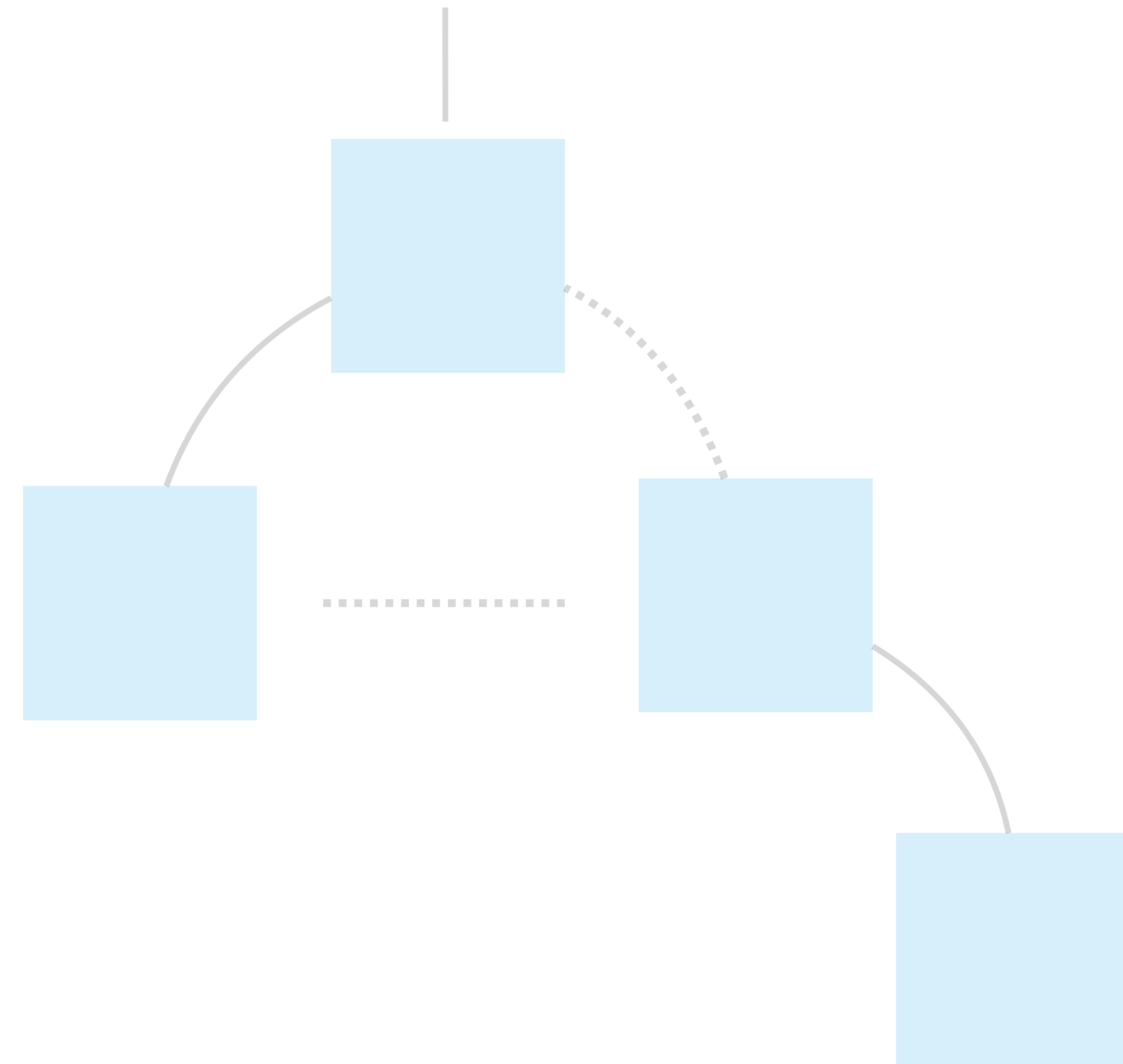
@GURLCODE



new ReactRoot

ReactDOM.unstable_createRoot(node)

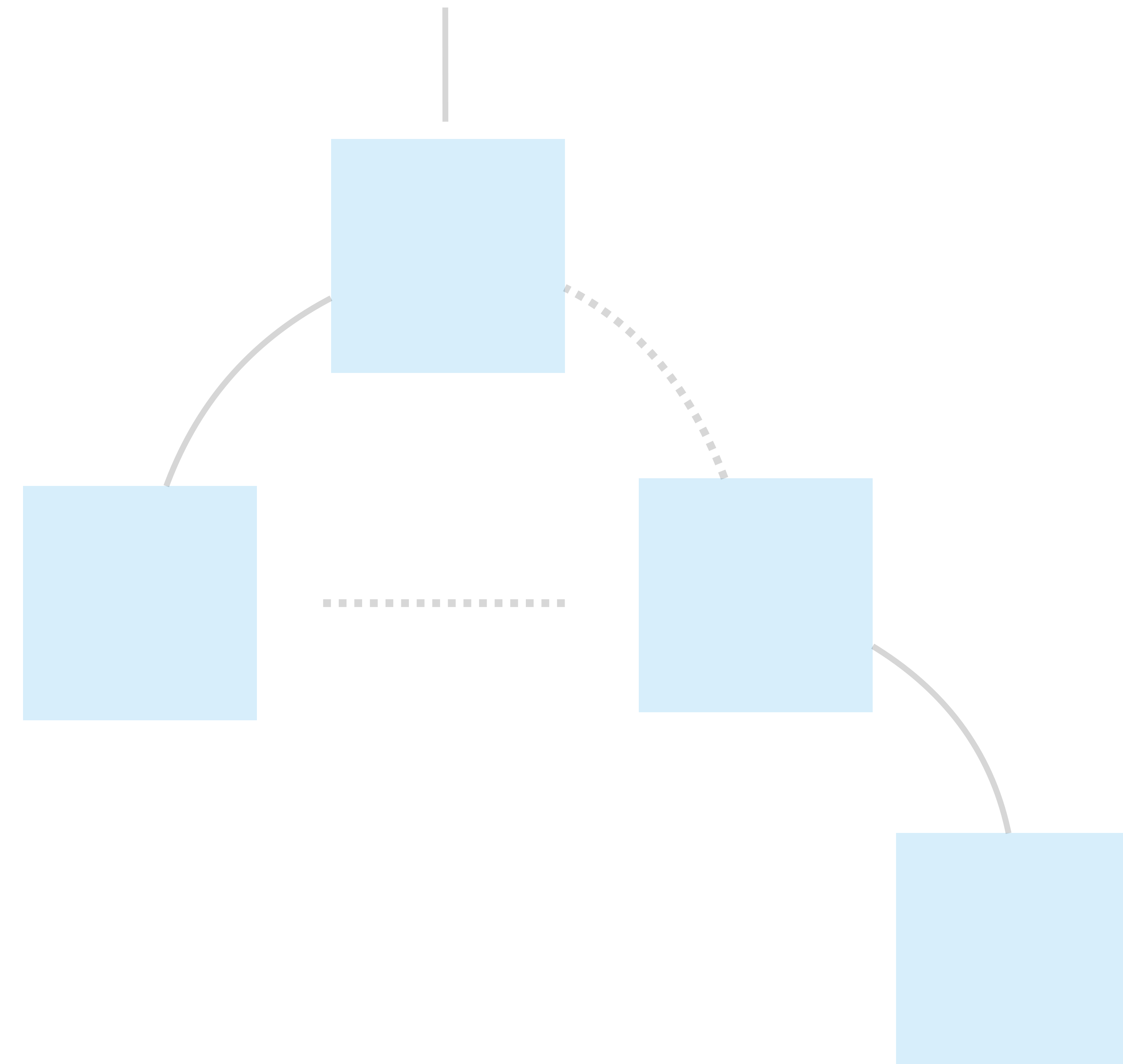
@GURLCODE



this._internalRoot = root

ReactDOM.unstable_createRoot(node)

@GURLCODE



root = ReactRoot

@GURLCODE

```
const container = document.getElementById('container');
const root = ReactDOM.unstable_createRoot(container);
root.render(
  <div>
    <h1>ColorBox</h1>
    <ColorBox />
  </div>
);
```

@GURLCODE

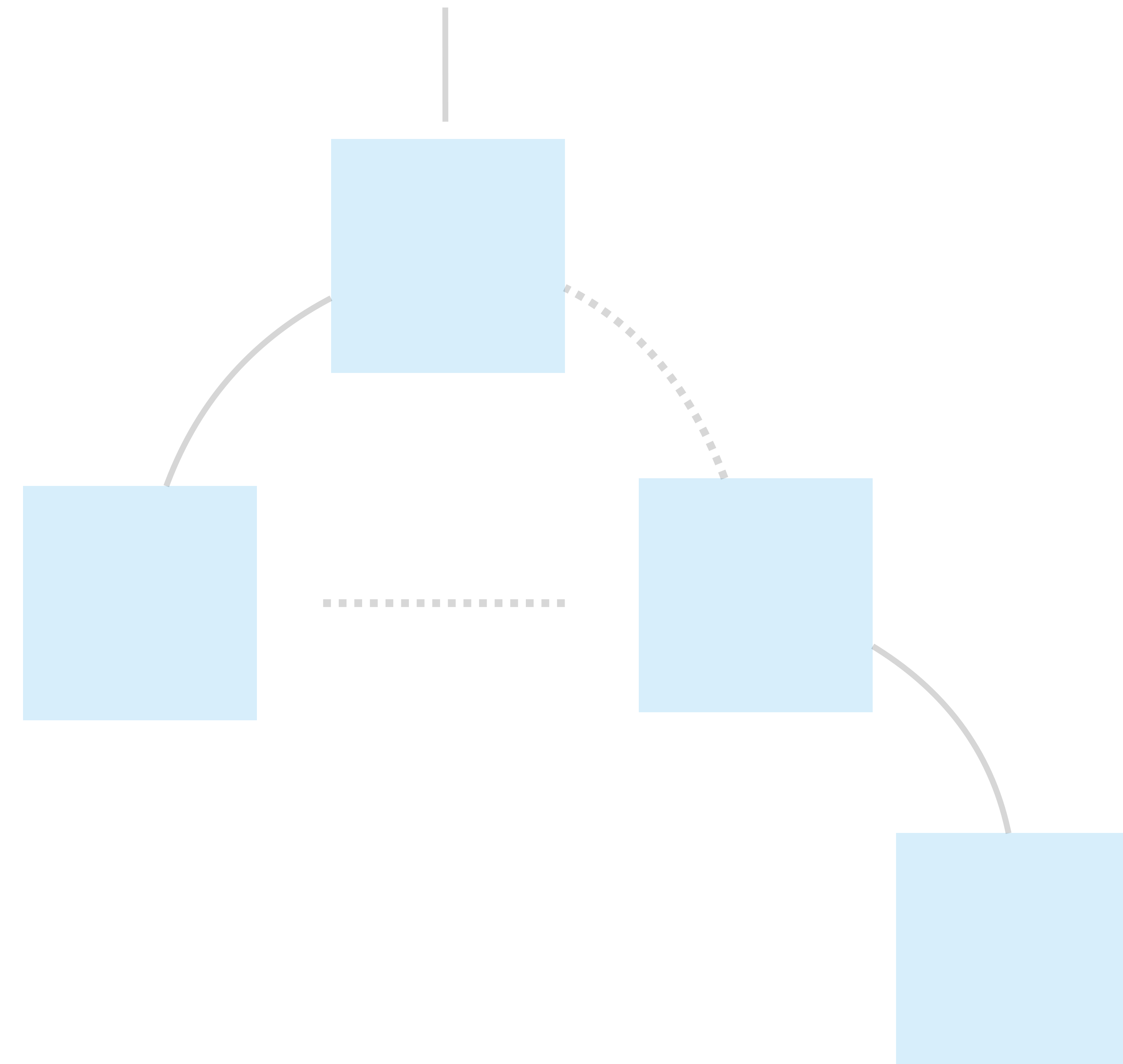
this.__InternalRoot

{

current: { tag: HostRoot }

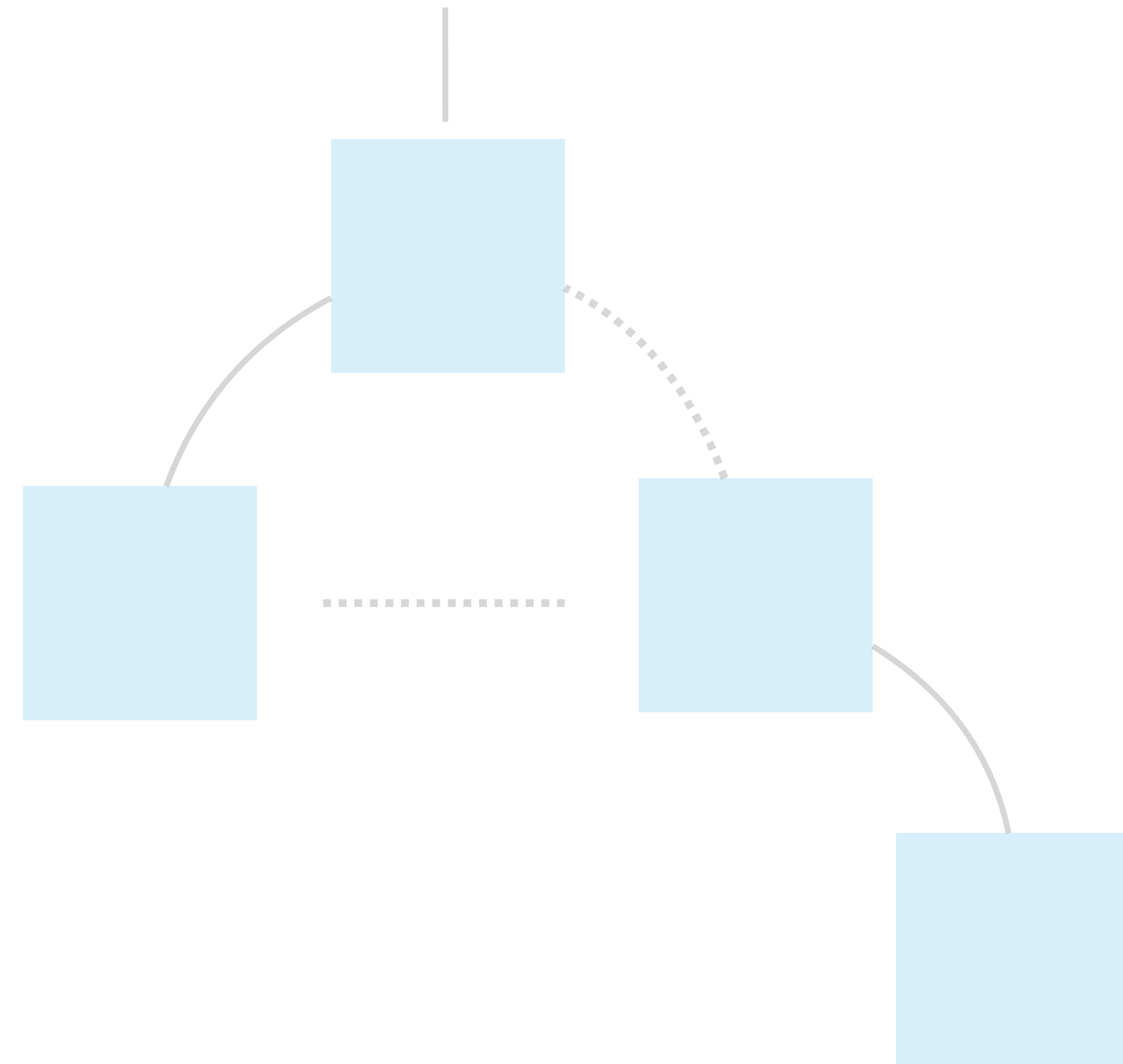
...

}



root.render(ReactElement)

@GURLCODE

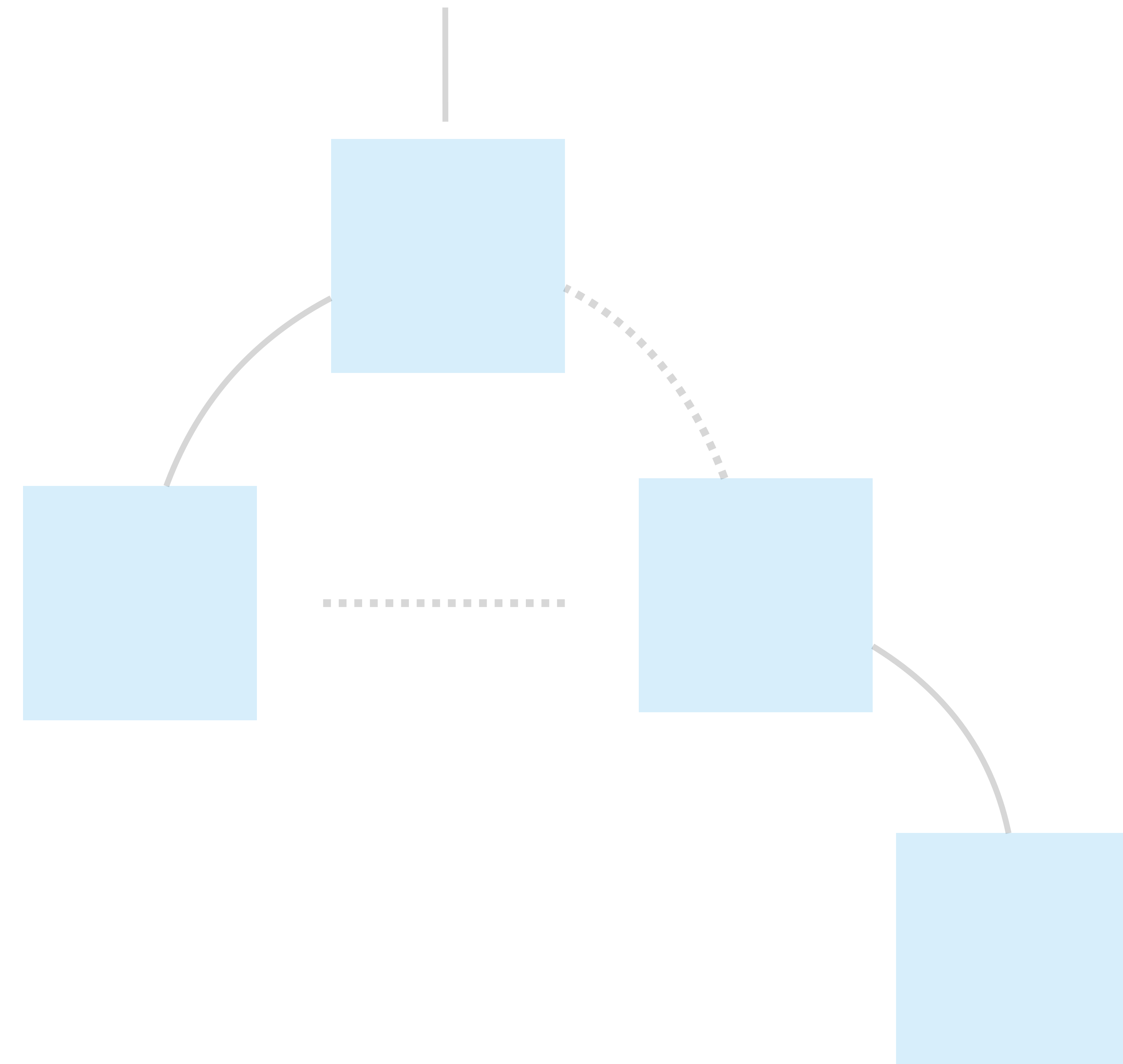


ReactRoot.render(ReactElement)

@GURLCODE

scheduleRootUpdate(current, element)

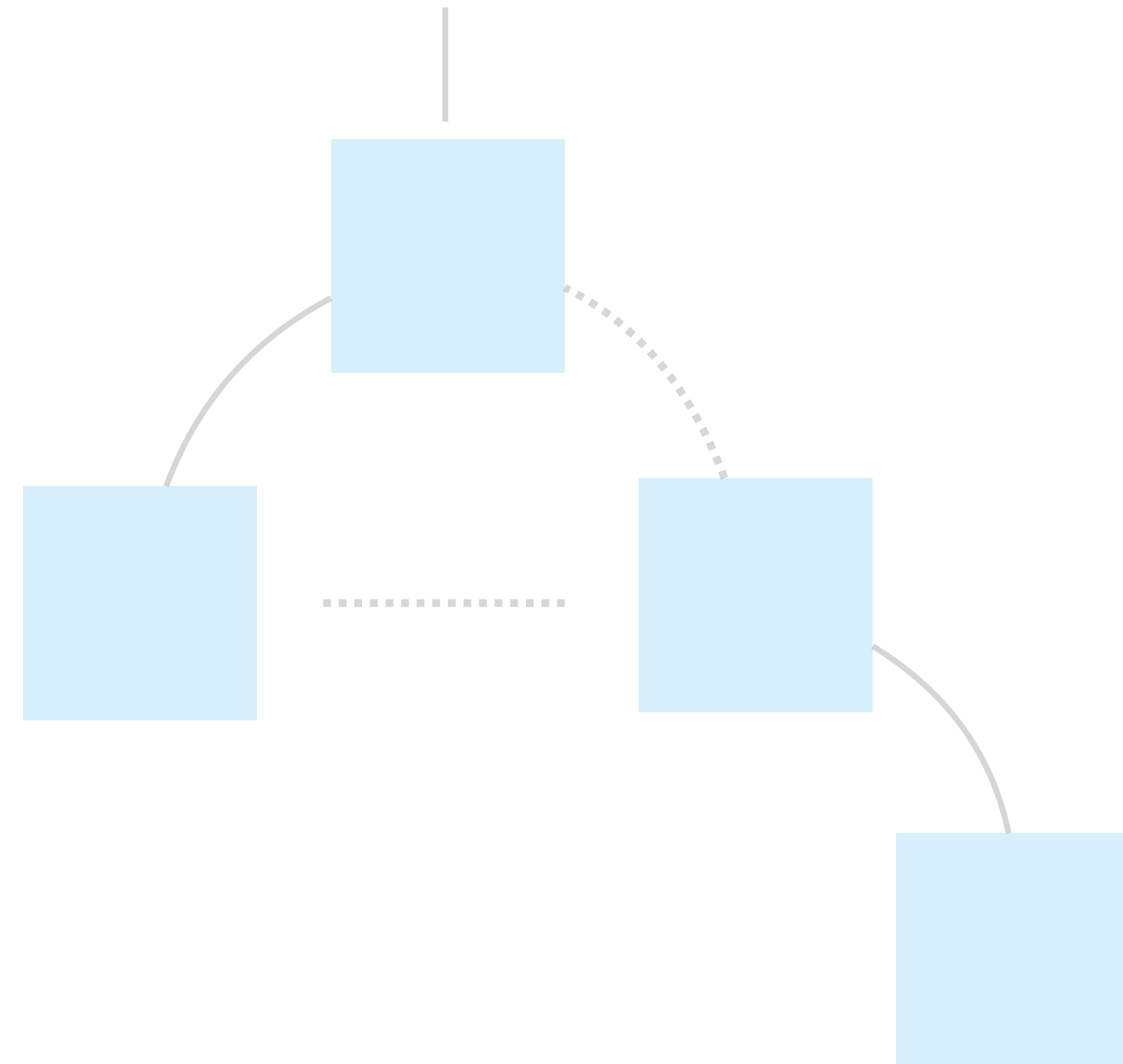
ReactRoot.render(ReactElement)



@GURLCODE

update = { element: element }

ReactRoot.render(ReactElement)

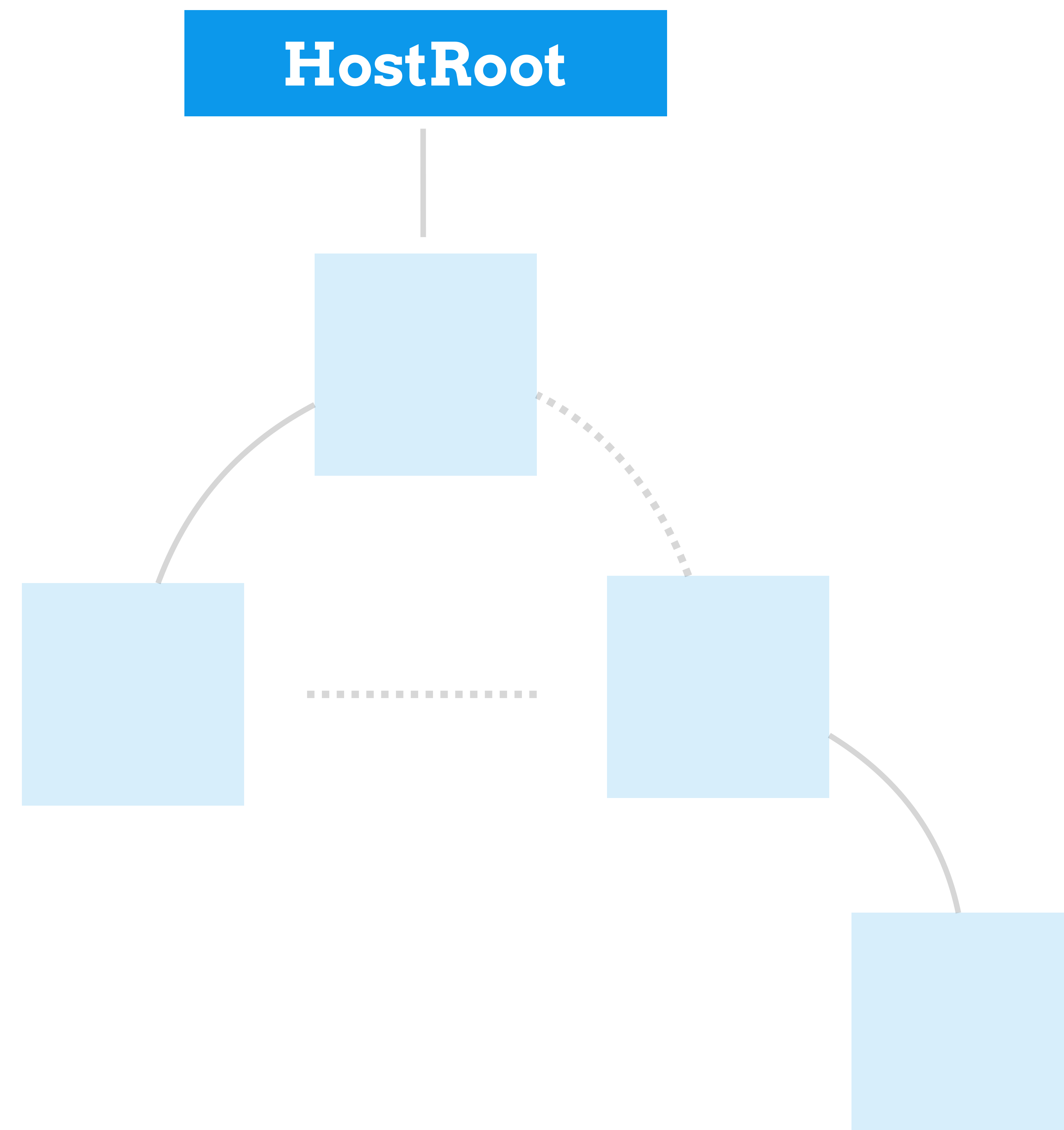


@GURLCODE

enqueueUpdate(update)

scheduleRootUpdate(current, element)

ReactRoot.render(ReactElement)



@GURLCODE

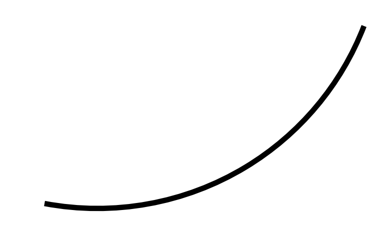
FIBER

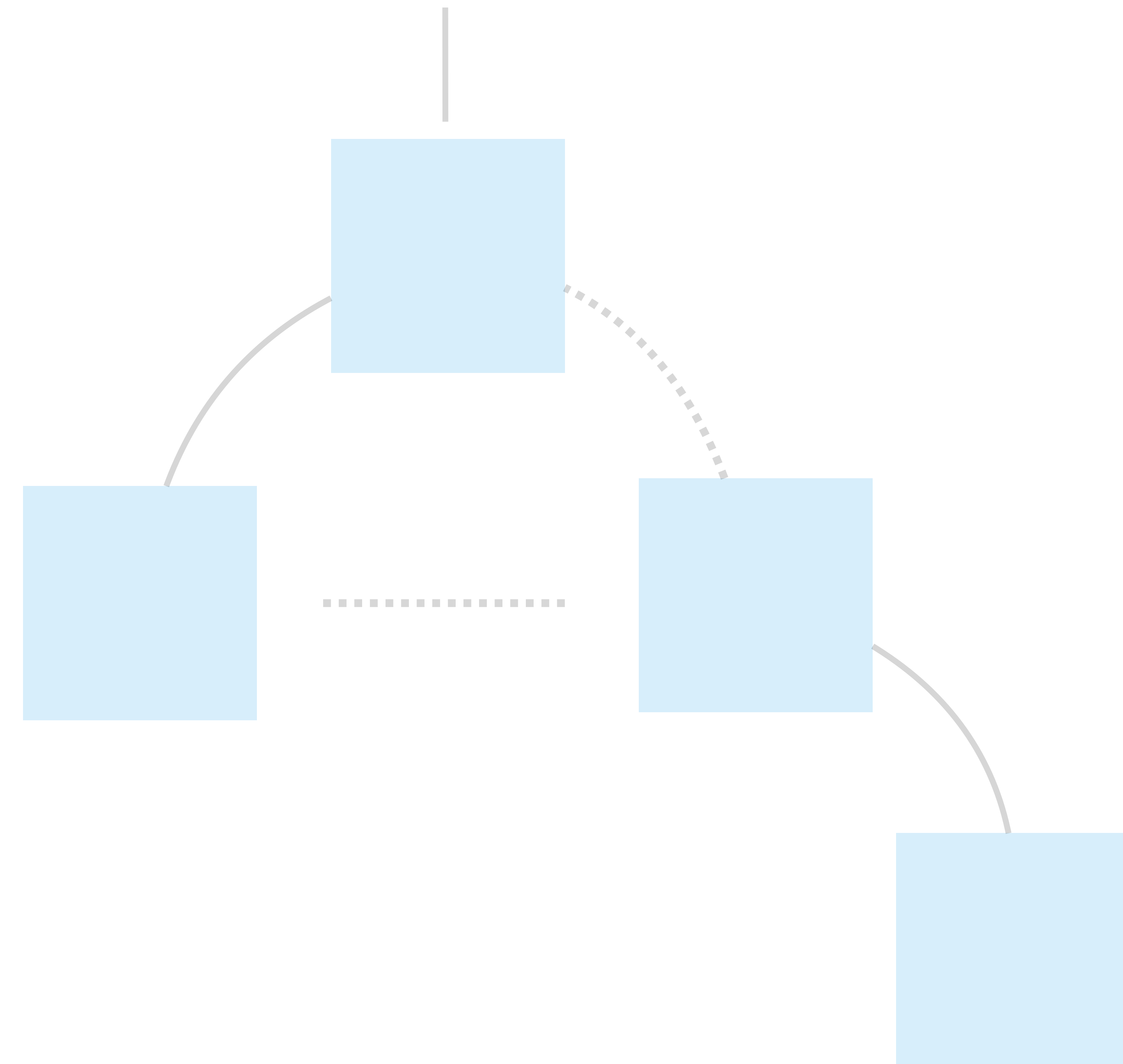
{

alternate
tag
child
sibling
return
updateQueue

}

list of state updates





scheduleRootUpdate(current, element)

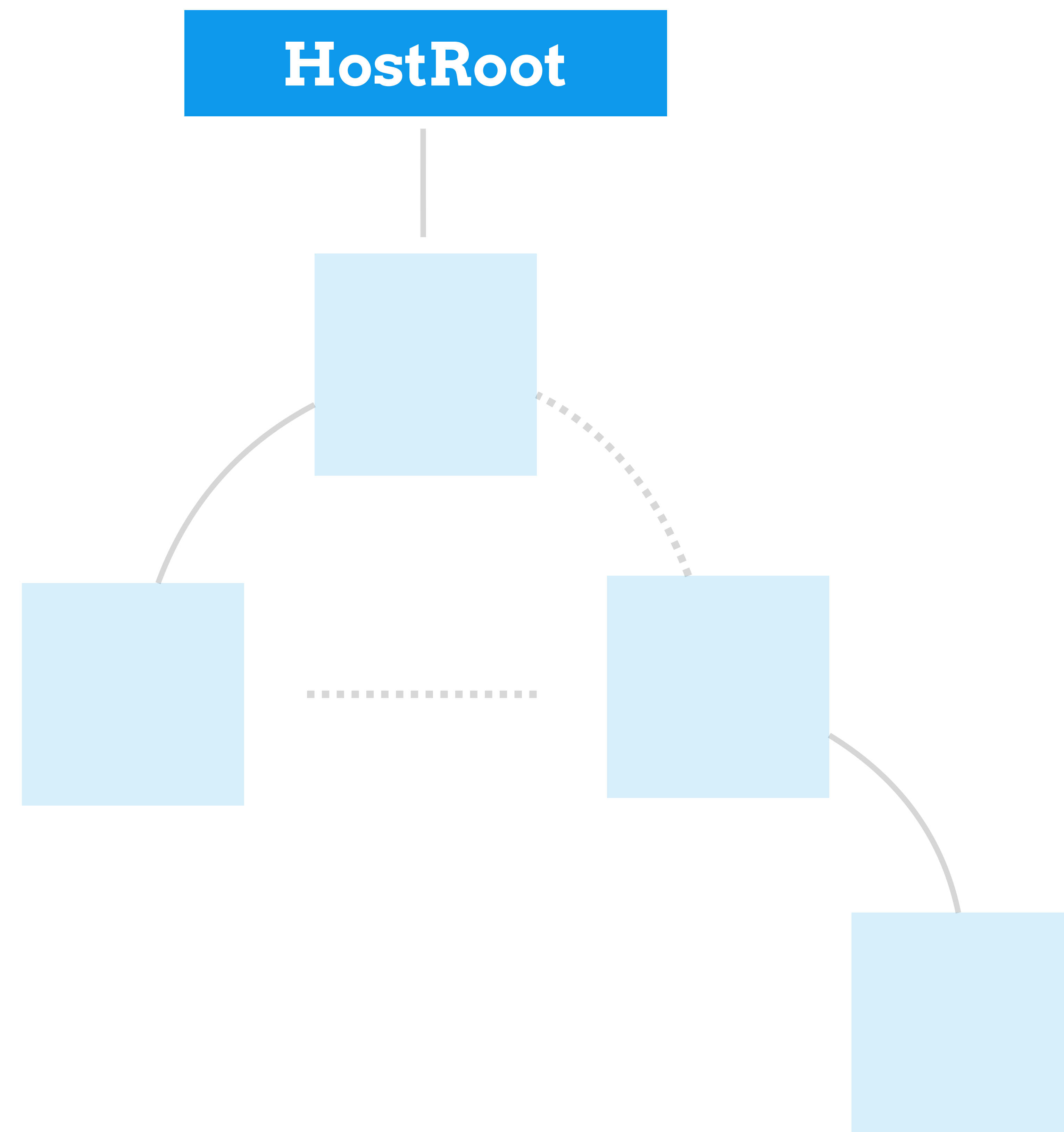
ReactRoot.render(ReactElement)

@GURLCODE

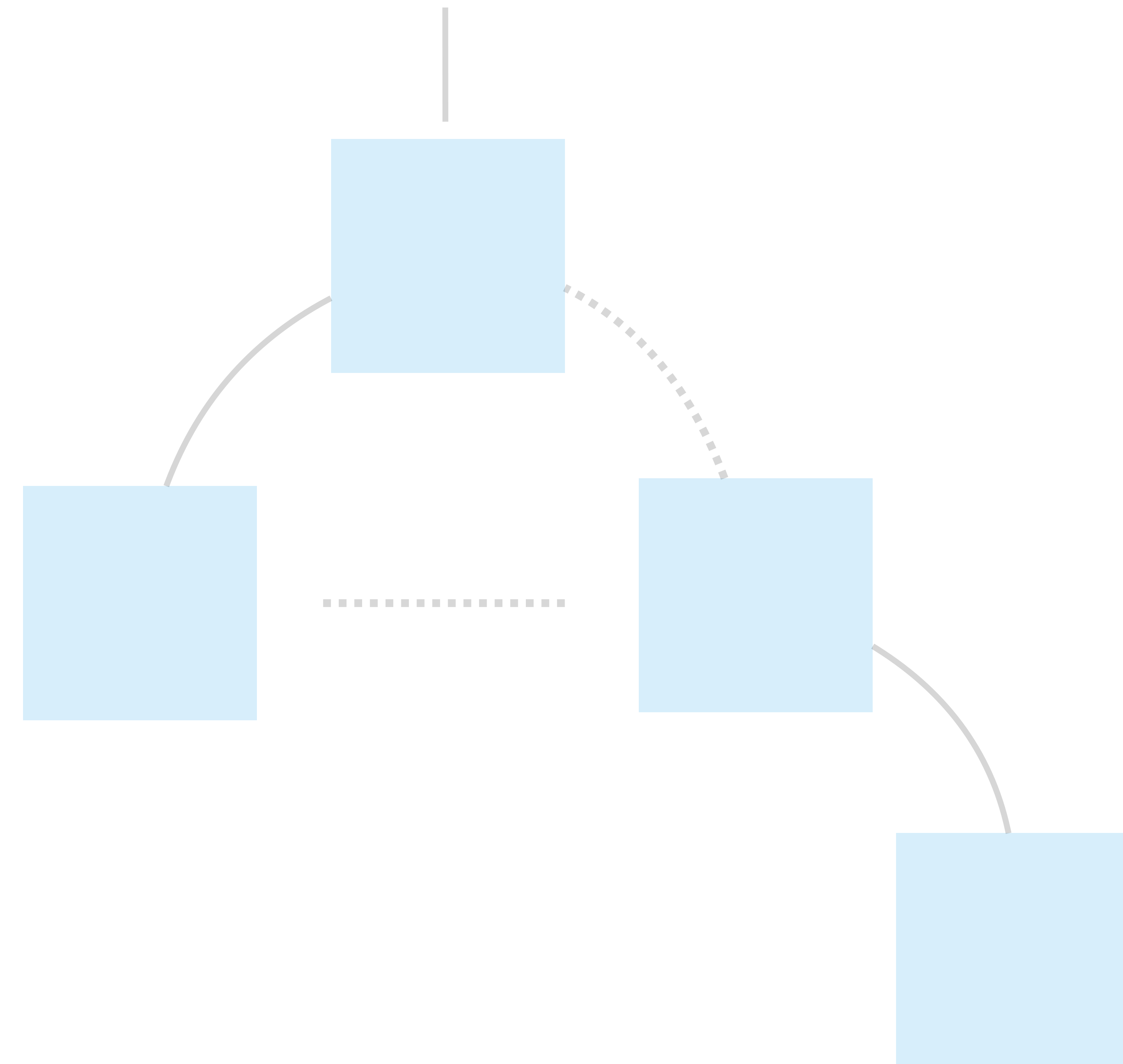
scheduleWork

scheduleRootUpdate(current, element)

ReactRoot.render(ReactElement)



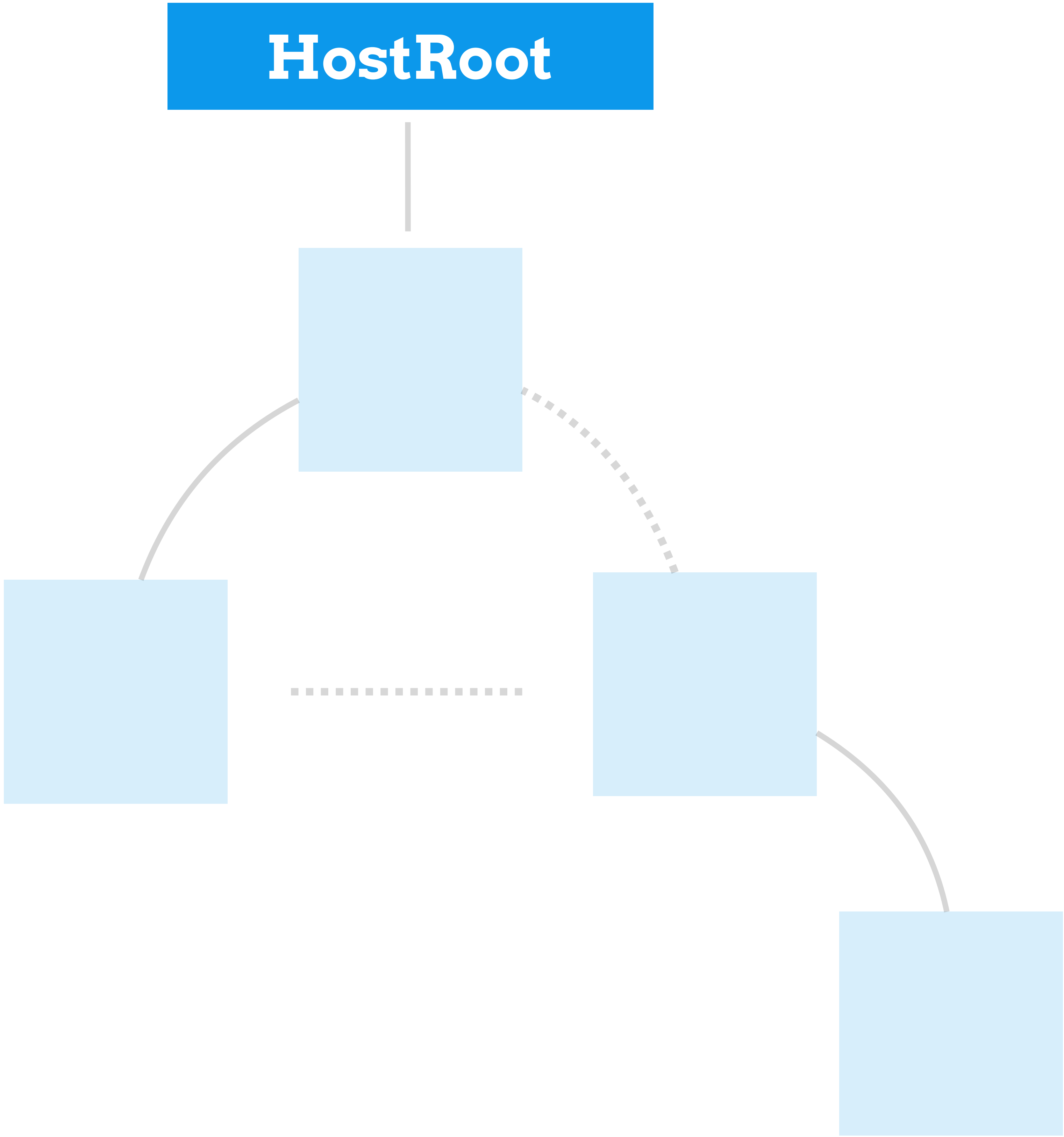
@GURLCODE



scheduleRootUpdate(current, element)

ReactRoot.render(ReactElement)

@GURLCODE

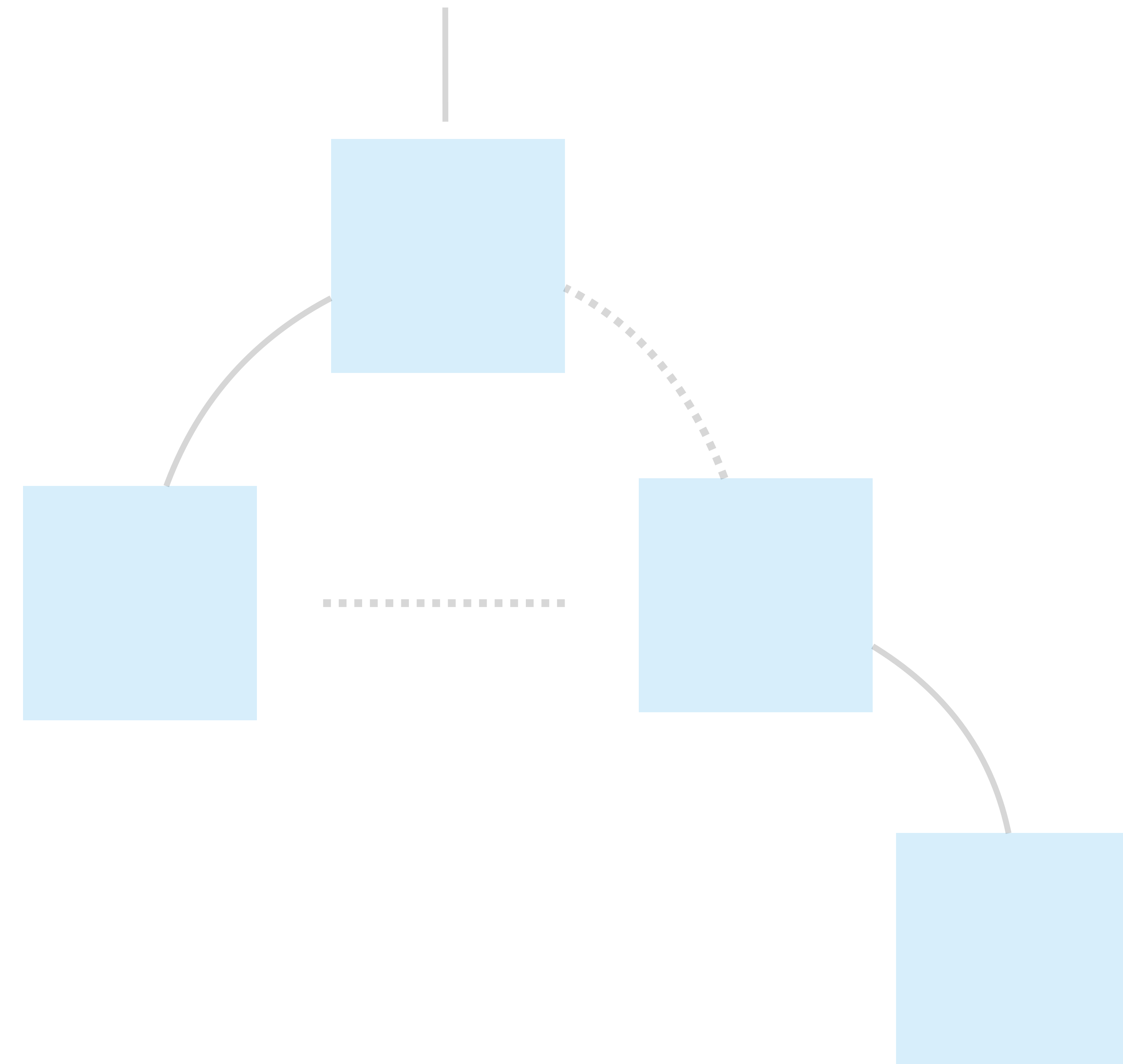


ReactRoot.render(ReactElement)

@GURLCODE

```
const container = document.getElementById('container');
const root = ReactDOM.unstable_createRoot(container);
root.render(
  <div>
    <h1>ColorBox</h1>
    <ColorBox />
  </div>
);
```

@GURLCODE

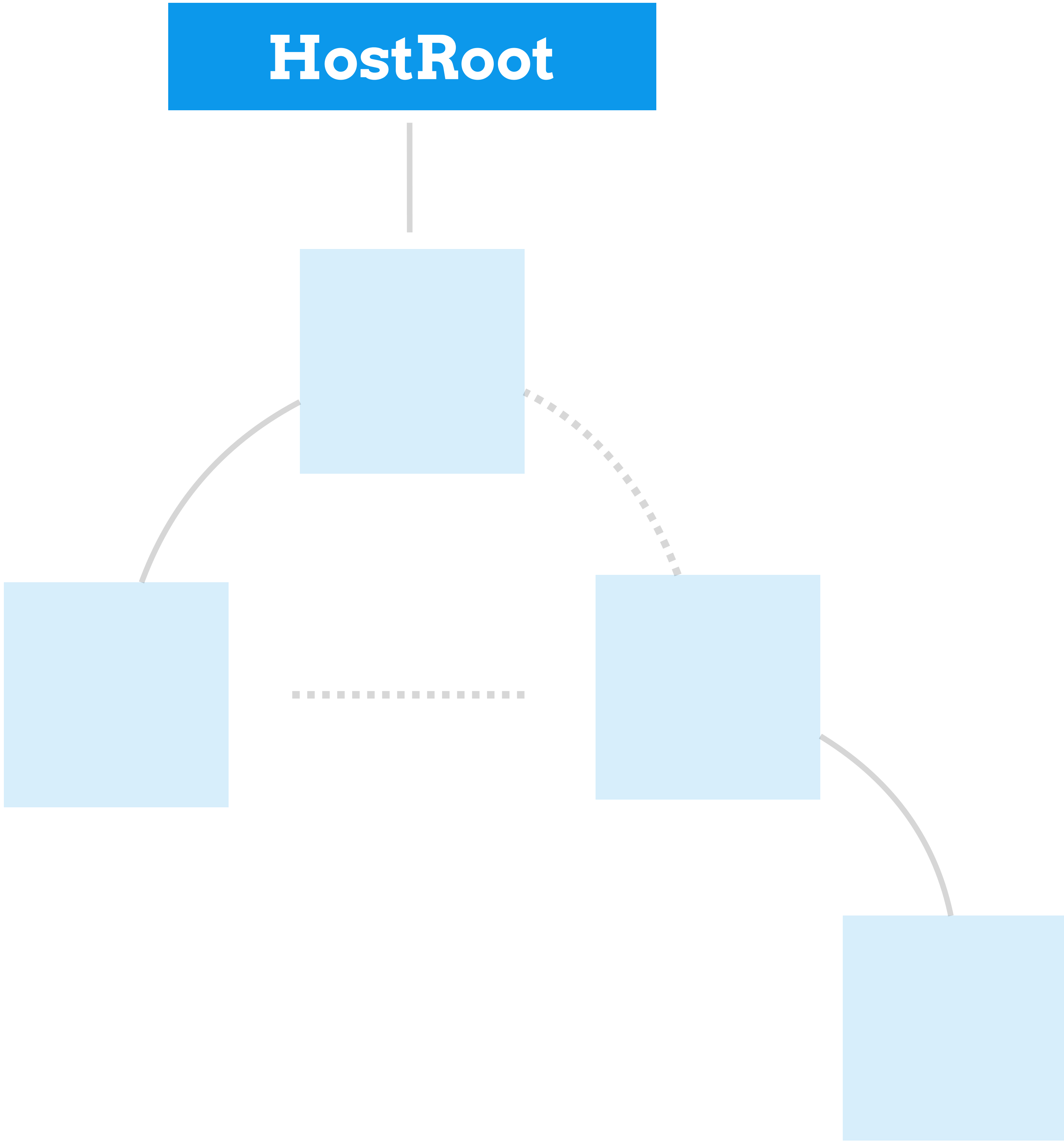


performWork

@GURLCODE

renderRoot

performWork

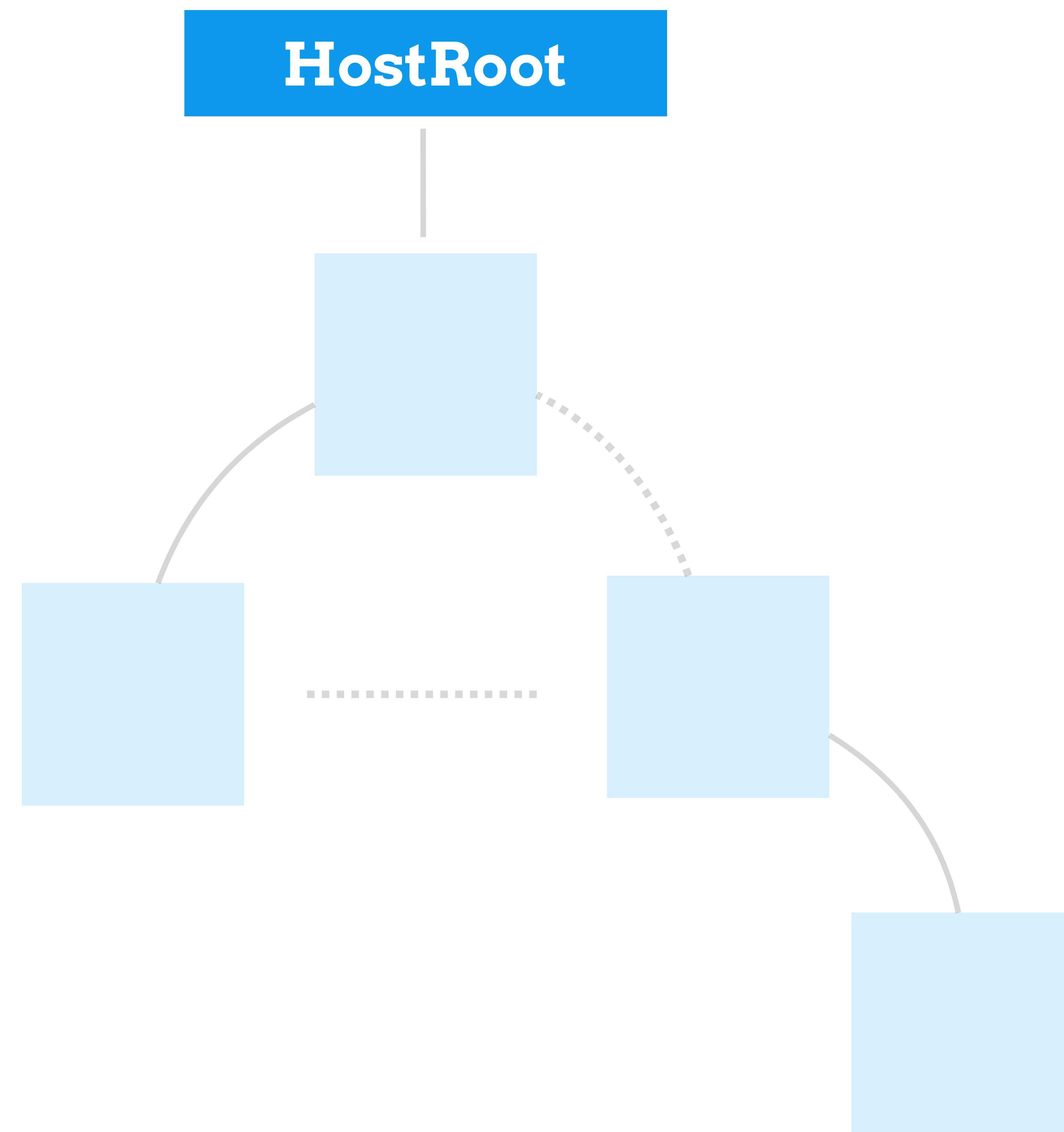


@GURLCODE

createWorkInProgress(current)

renderRoot

performWork

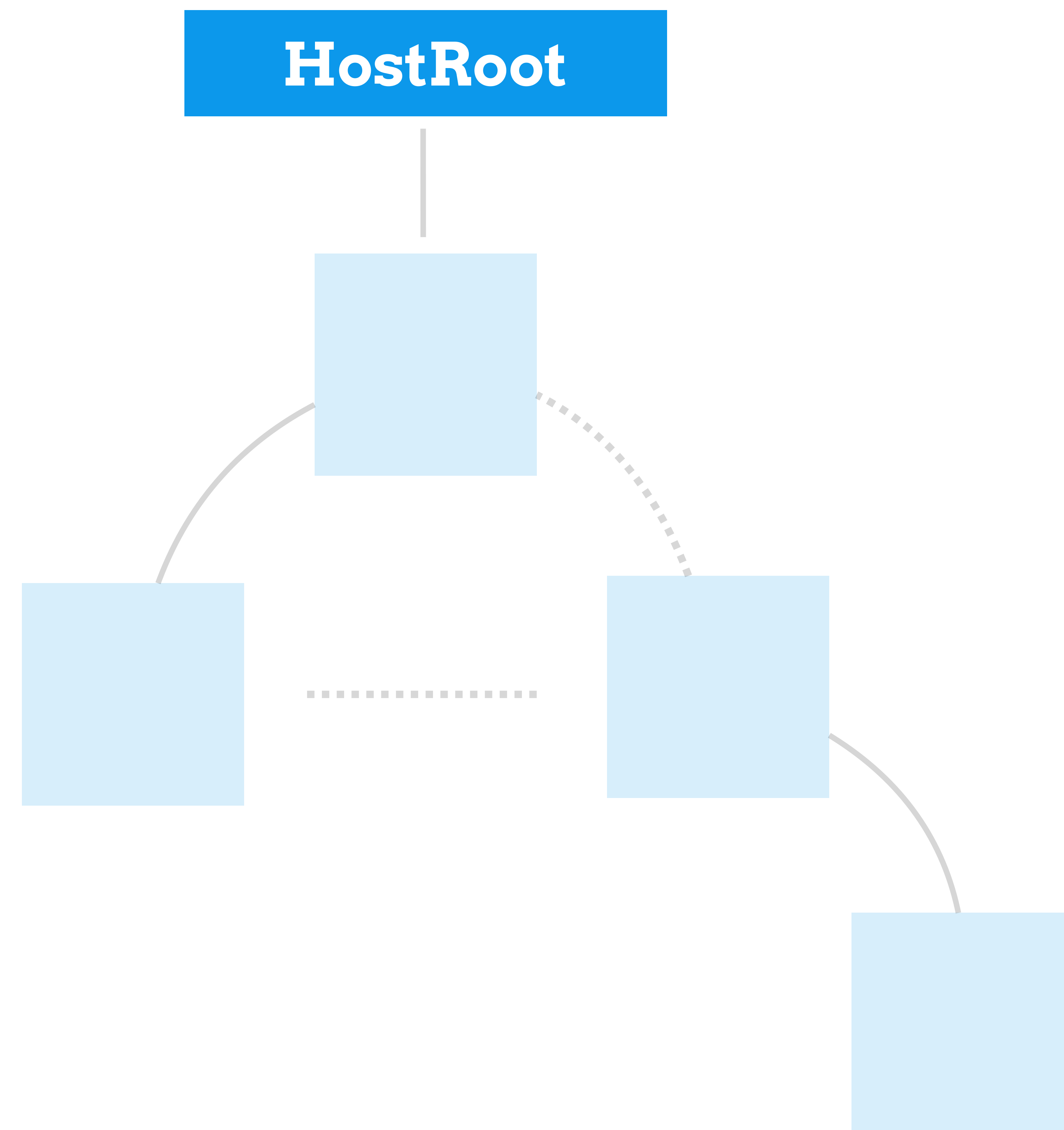


@GURLCODE

workInProgress = current.alternate

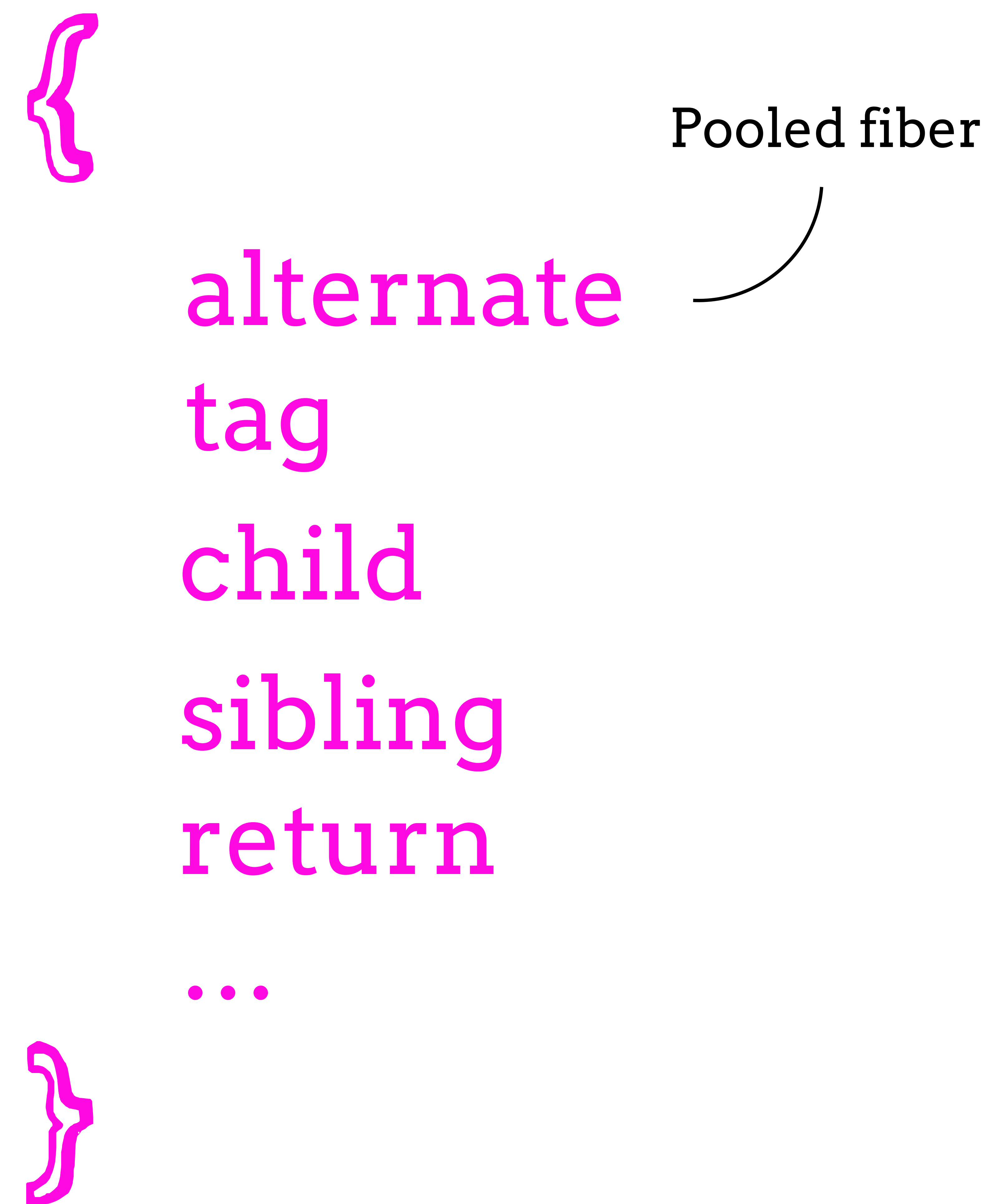
renderRoot

performWork



@GURLCODE

FIBER



@GURLCODE

Andrew Clark
@acdlite

So much of React's architecture is based on stuff game developers figured out decades ago.

Like double buffering. React has a current tree and a work-in-progress tree. When the WIP tree is finished rendering, we swap. The WIP becomes current.

Read here: gameprogrammingpatterns.com/double-buffer.…

2:14 PM - 27 Mar 2018

@GURLCODE

Overreacted

Personal blog by Dan Abramov.
I explain with words and code.

How Are Function Components Different from Classes?

March 3, 2019 • ☕☕☕ 14 min read

They're a whole different Pokémon.

Coping with Feedback

March 2, 2019 • ☕ 3 min read

Sometimes I can't fall asleep.

Elements Memory Network Performance Console

top ▼ Filter All level

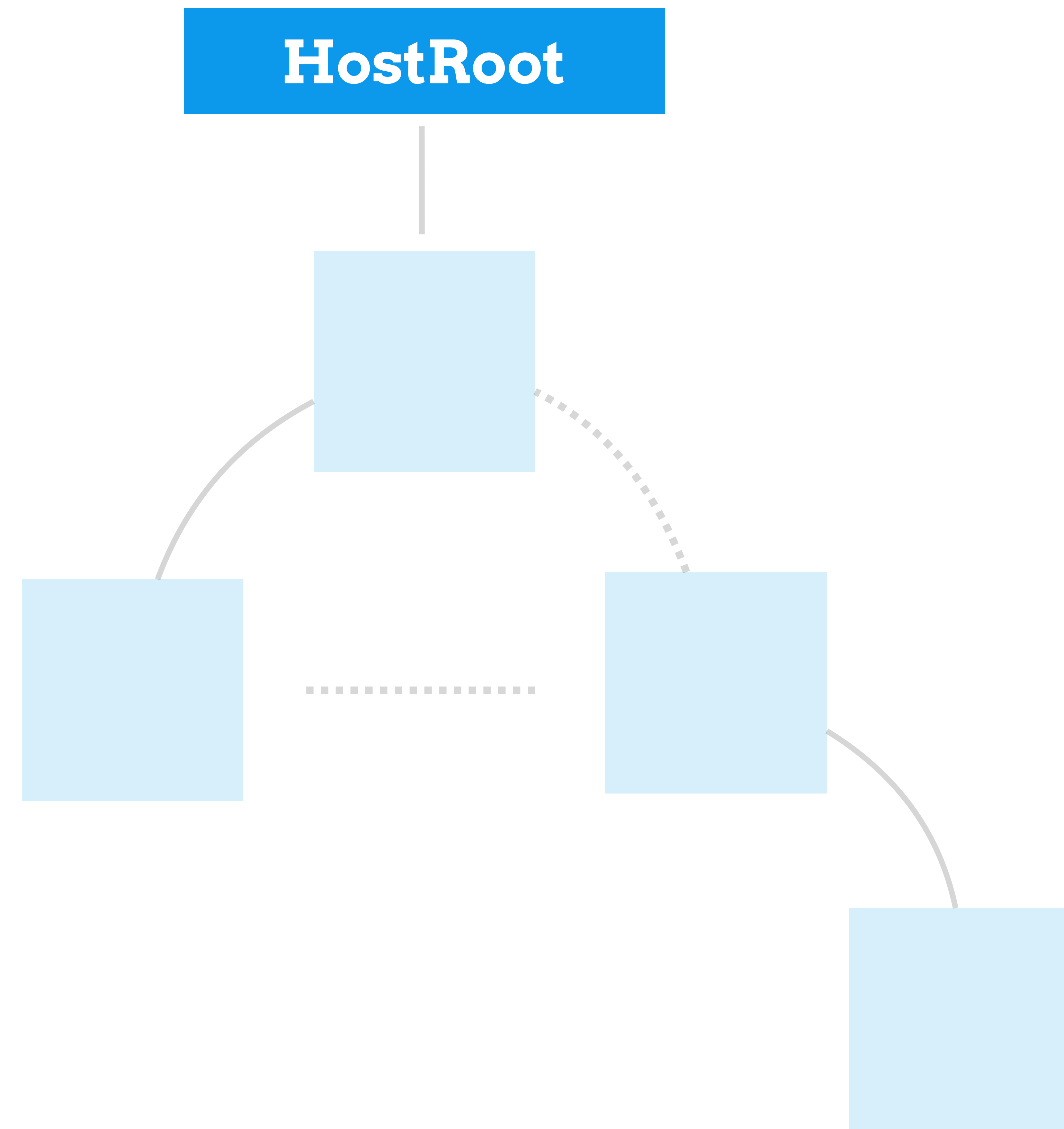
```
[Violation] 'setTimeout' handler took 69ms
> const toggle = document.querySelector('.react-toggle');
< undefined
> toggle.__reactInternalInstance$yondtp9ujre
< ▶ Wr {tag: 5, key: null, elementType: "div", type: "div", stateN
  , …}
>
```

@GURLCODE

workInProgress === null

renderRoot

performWork



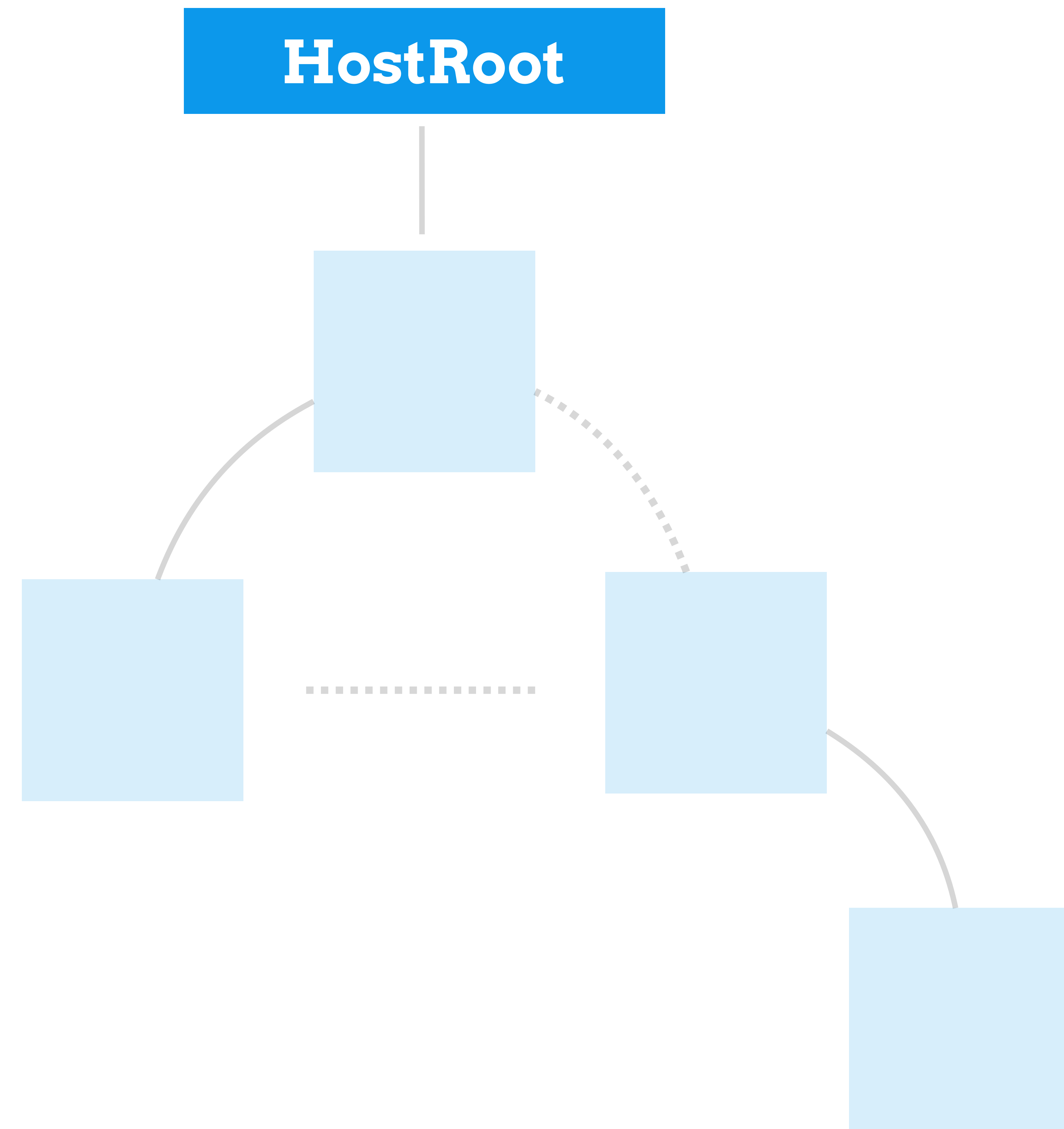
@GURLCODE

createFiber(current)

workInProgress = current.alternate

renderRoot

performWork

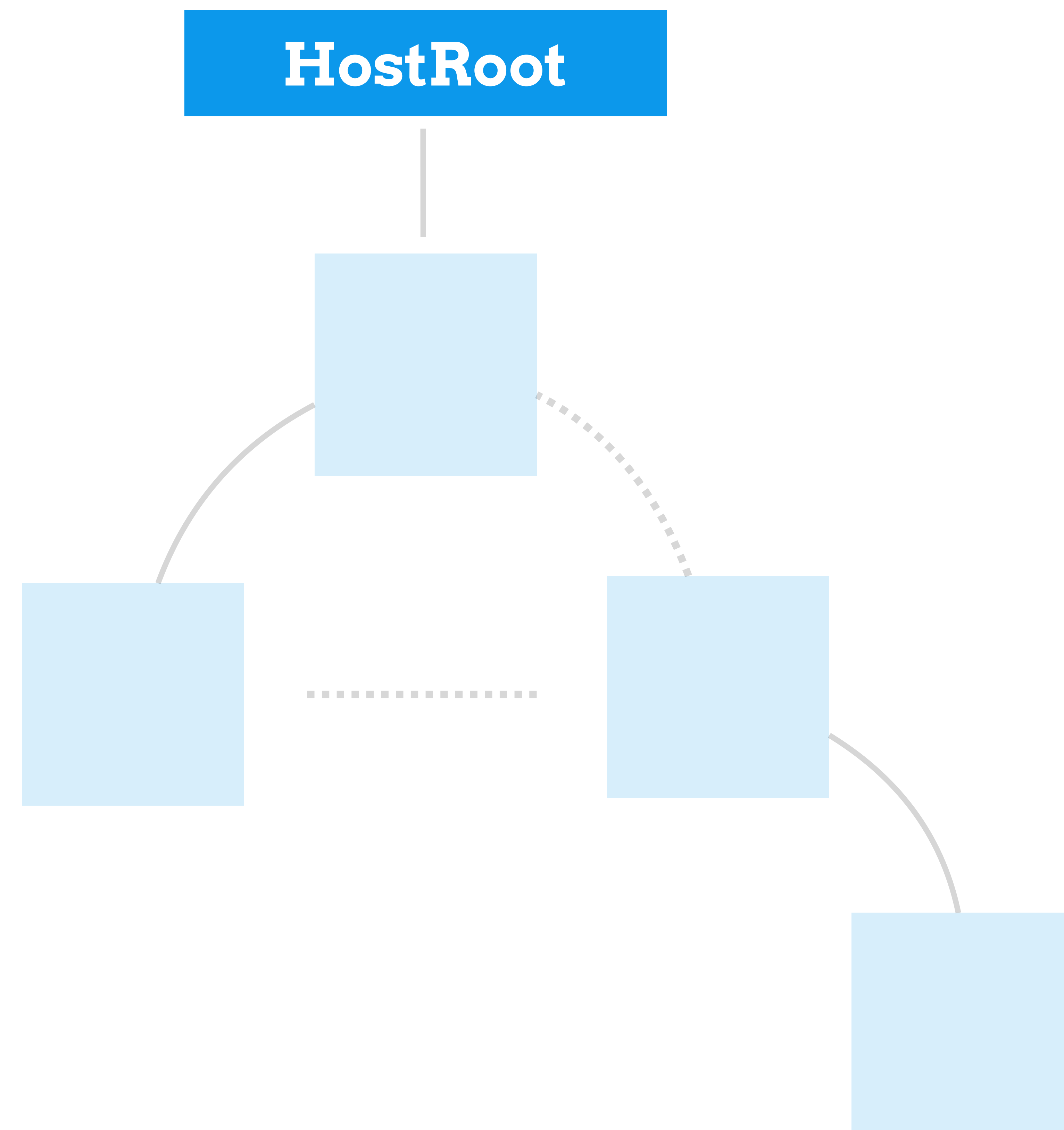


@GURLCODE

current.alternate = workInProgress

renderRoot

performWork

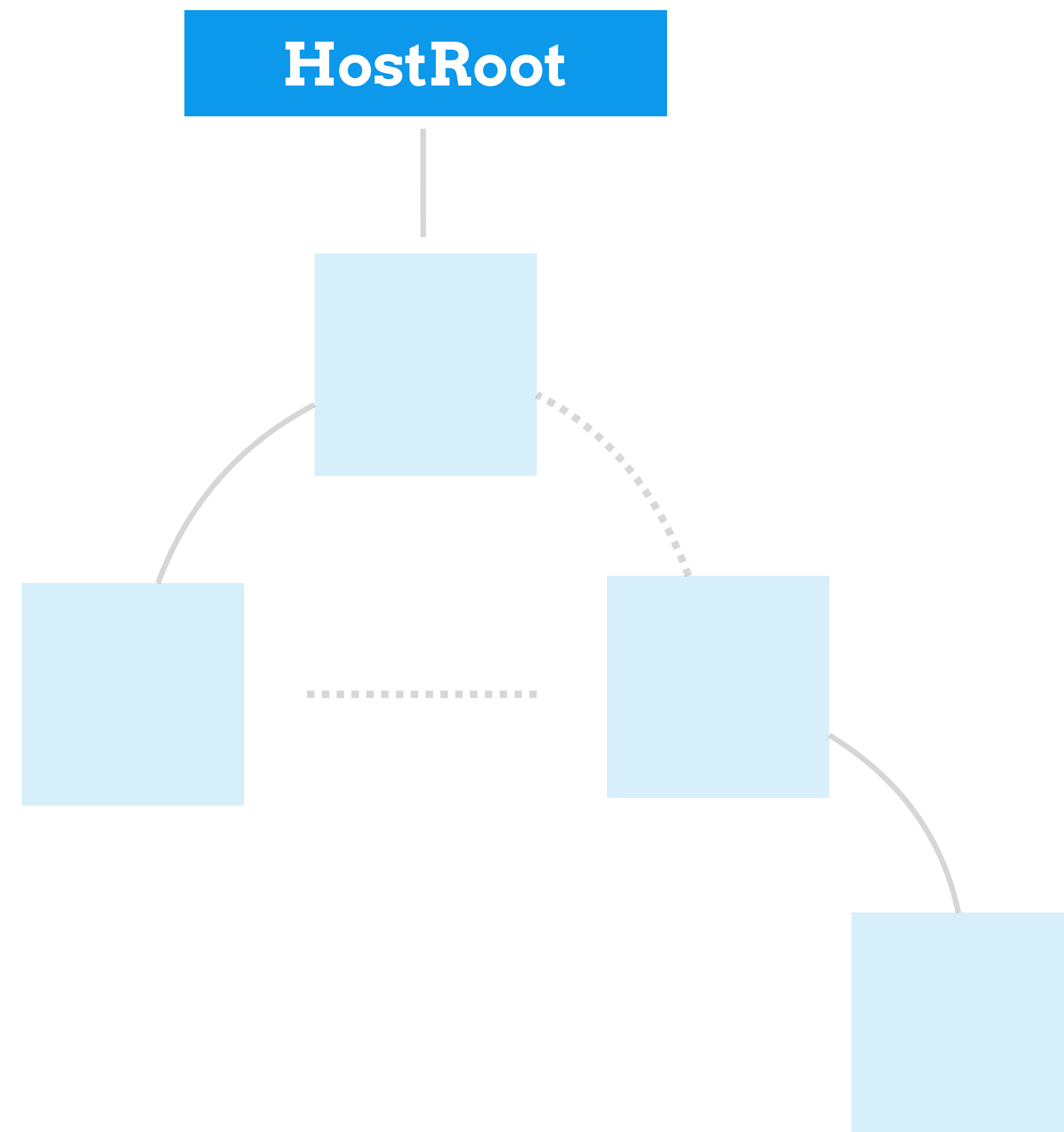


@GURLCODE

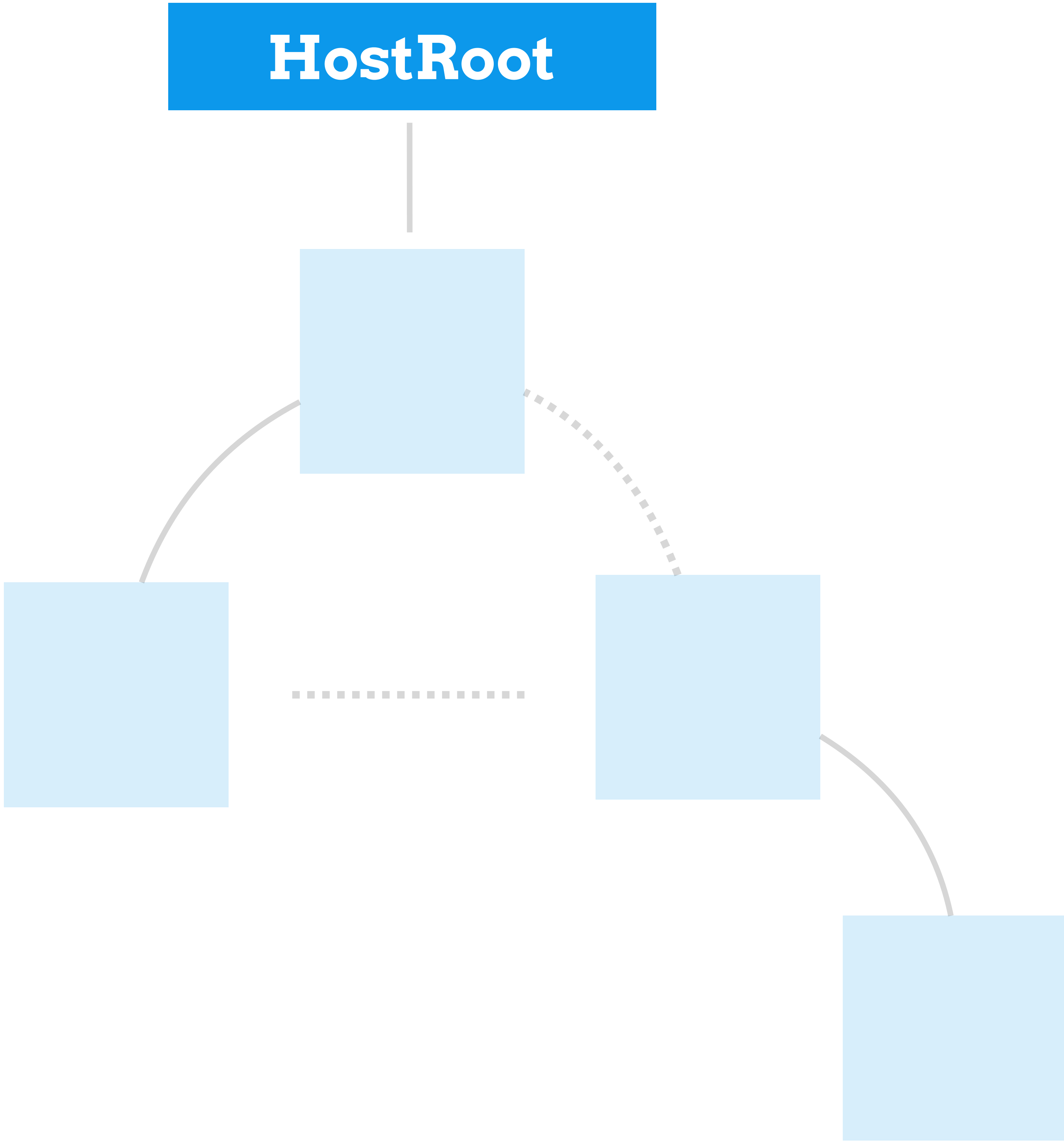
createWorkInProgress(current)

renderRoot

performWork



@GURLCODE



nextUnitOfWork = workInProgress

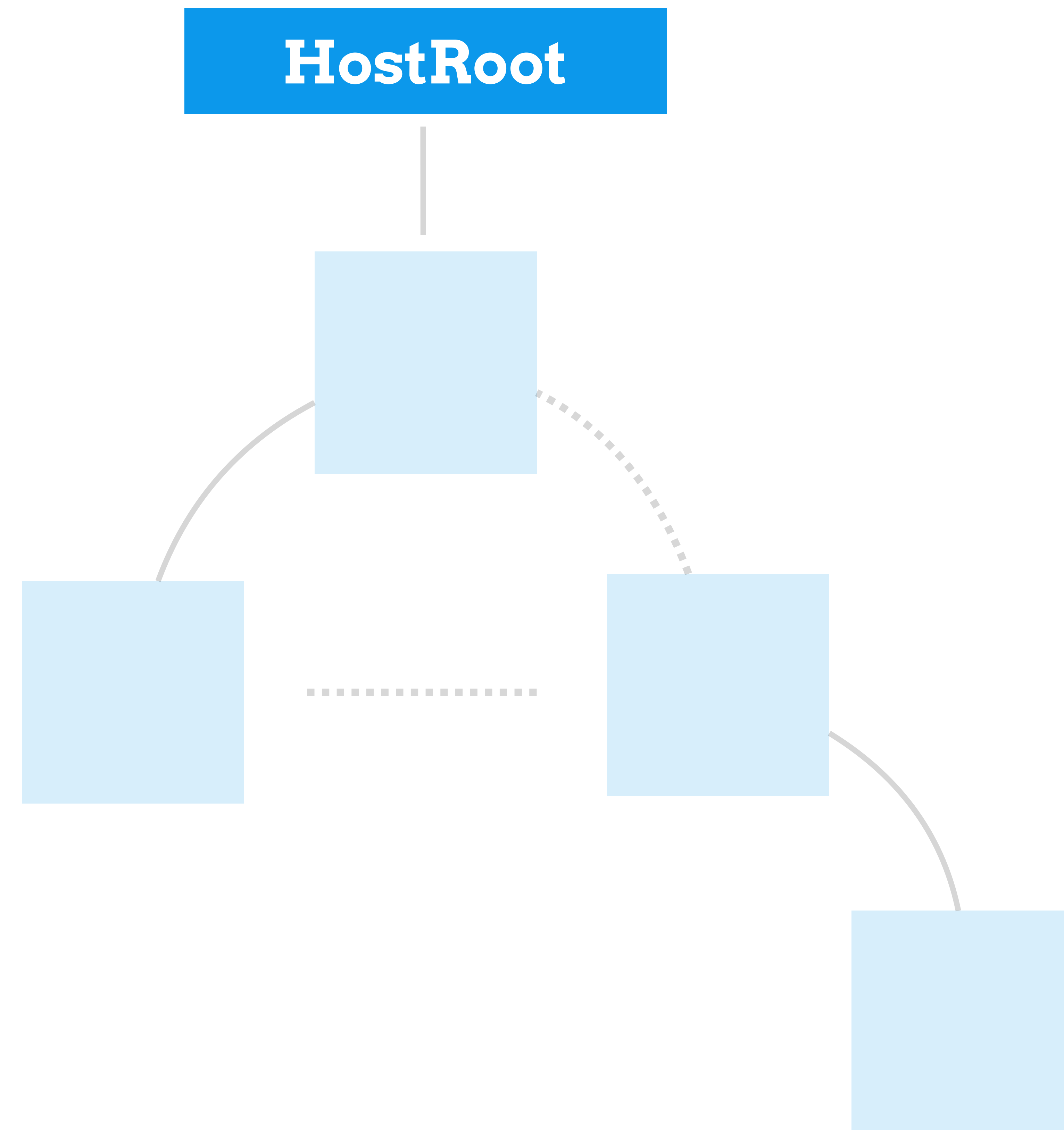
performWork

@GURLCODE

workLoop

nextUnitOfWork = workInProgress

performWork



@GURLCODE

```
function workLoop() {
    while (nextUnitOfWork !== null) {
      nextUnitOfWork = performUnitOfWork(nextUnitOfWork);
    }
}
```

@GURLCODE

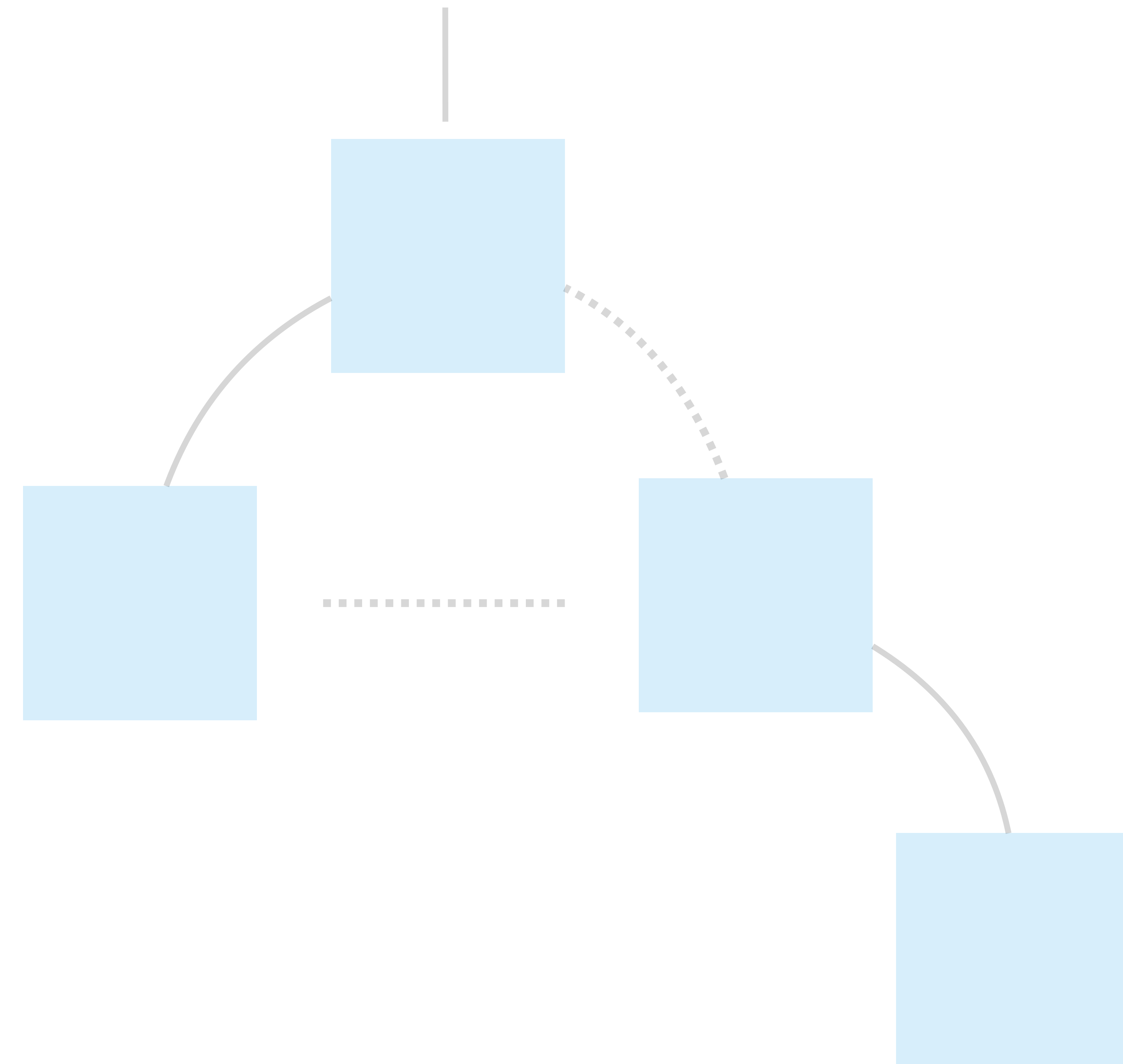
Dan Abramov
@dan_abramov

fiber

Unit of work = a node. For example a component, or a div.

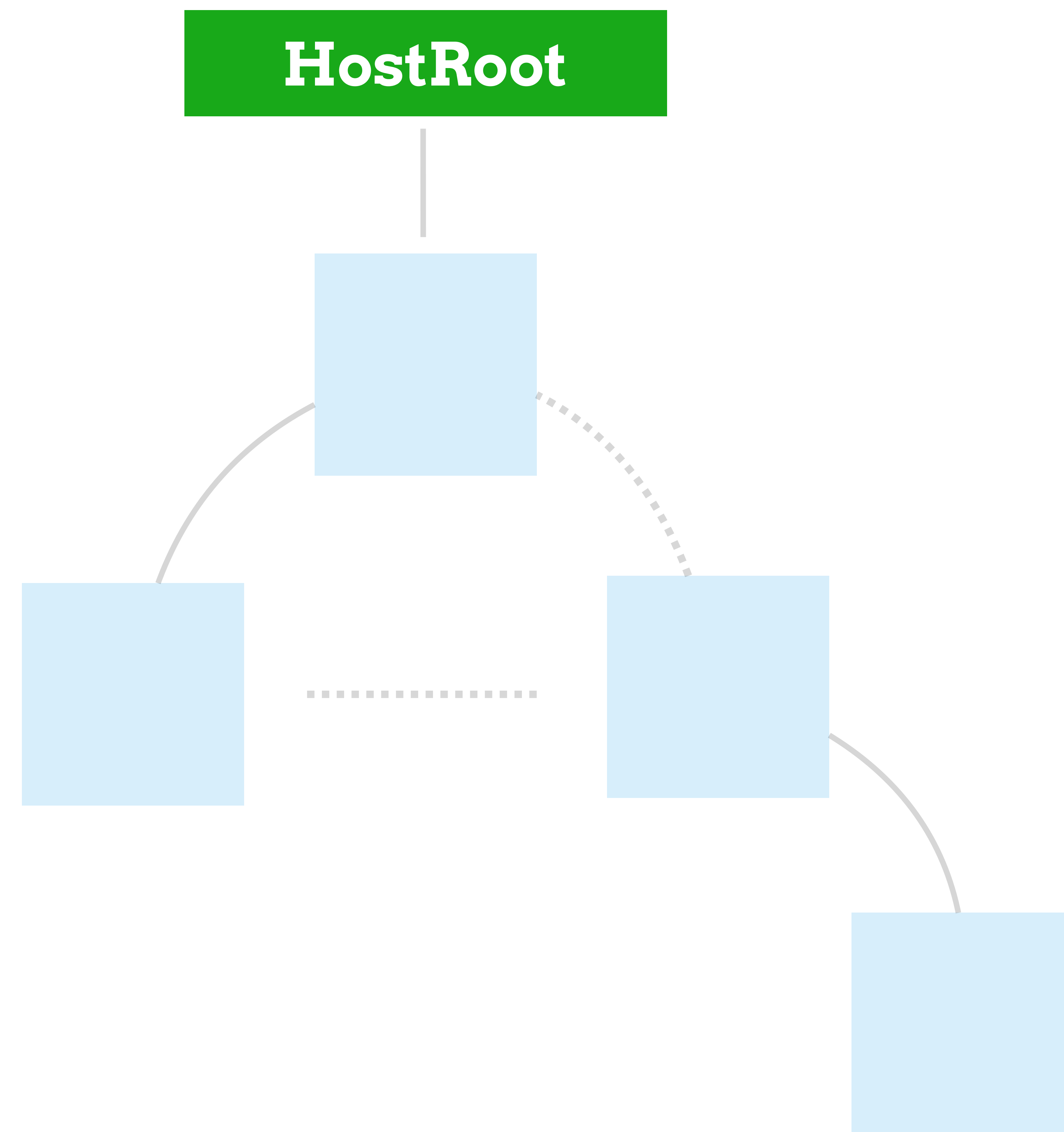
8:36 PM - 24 Apr 2018

@GURLCODE



workLoop

@GURLCODE



performUnitOfWork

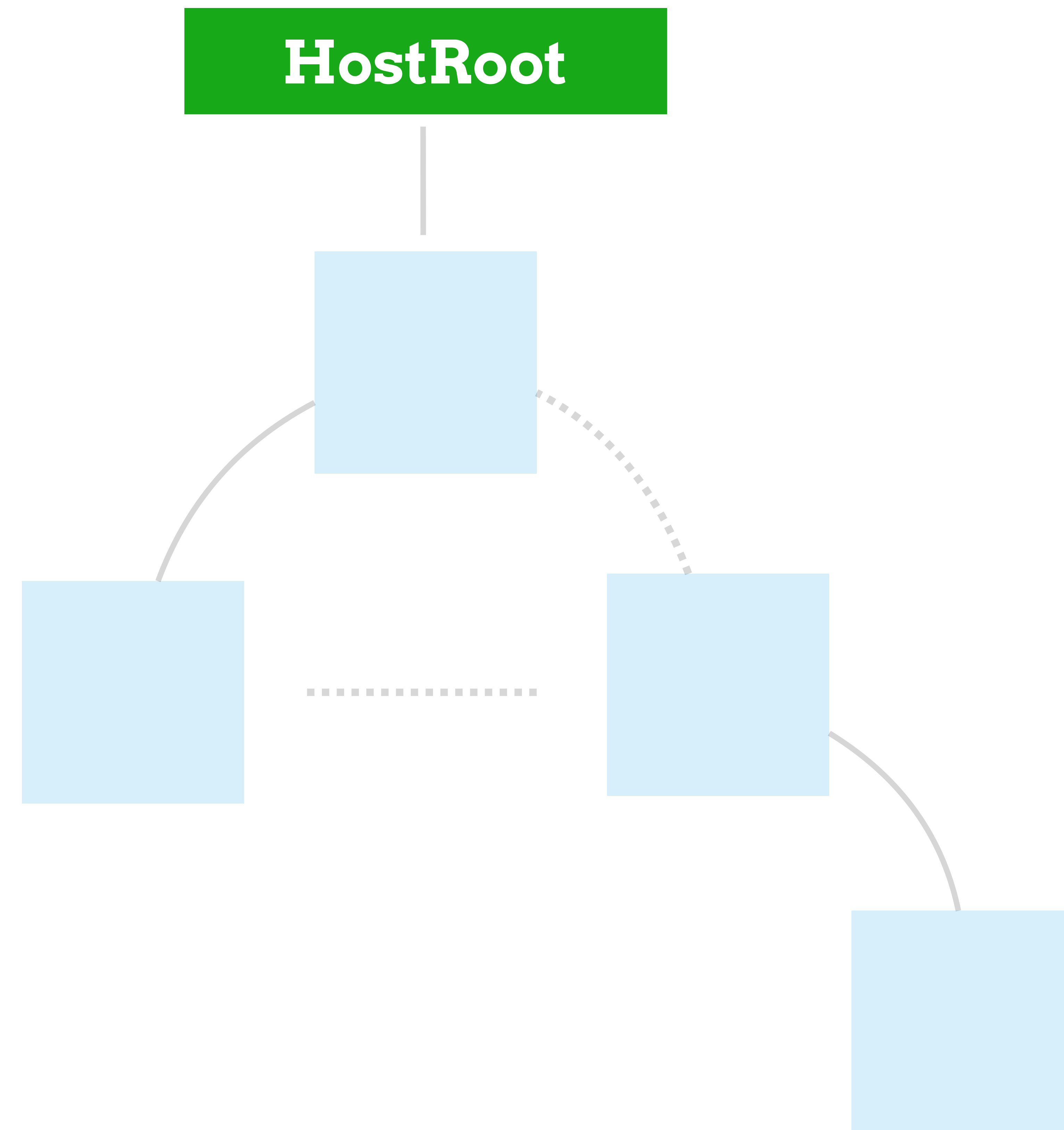
workLoop

@GURLCODE

beginWork

performUnitOfWork

workLoop



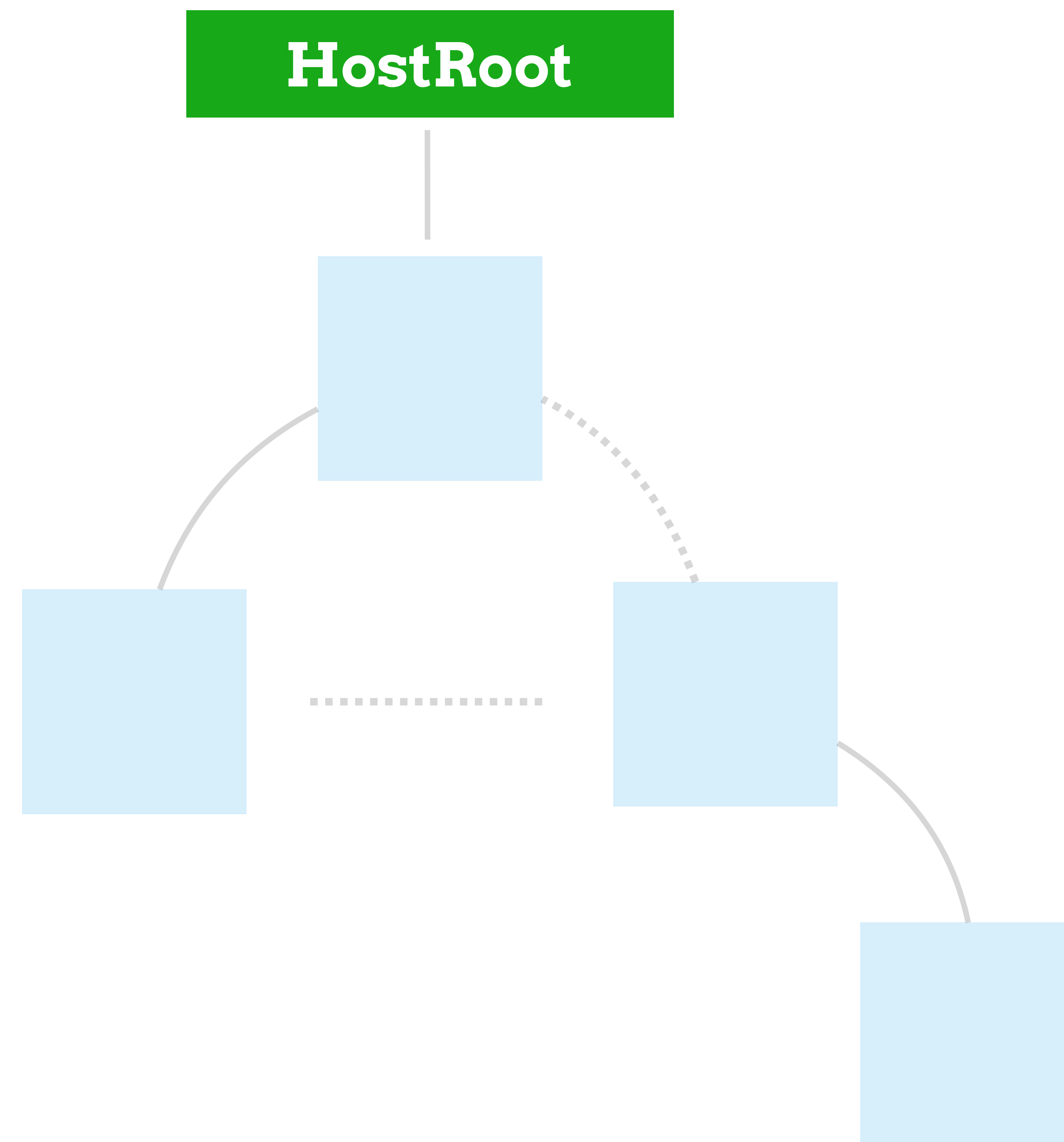
@GURLCODE

updateHostRoot

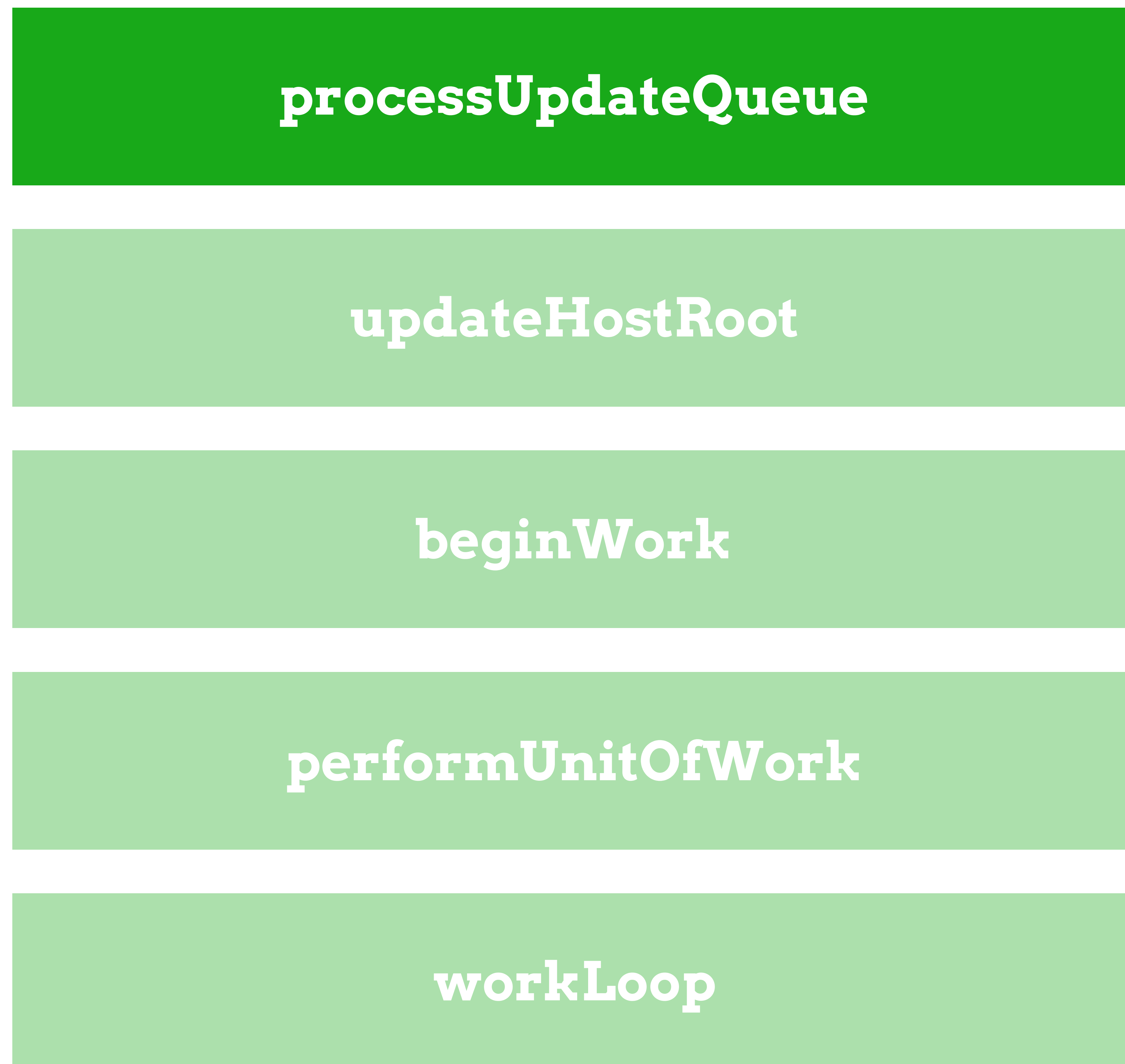
beginWork

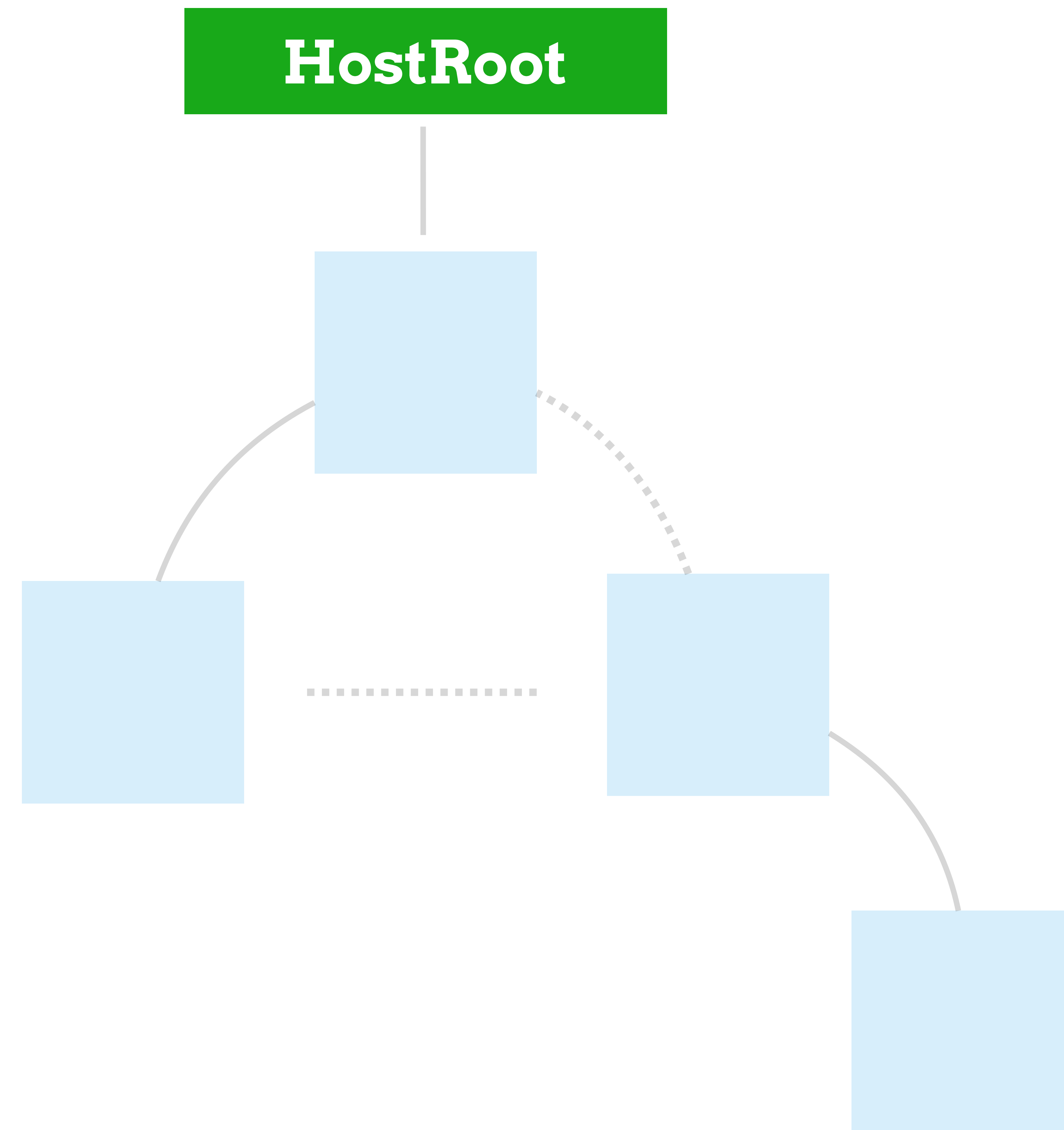
performUnitOfWork

workLoop



@GURLCODE





@GURLCODE

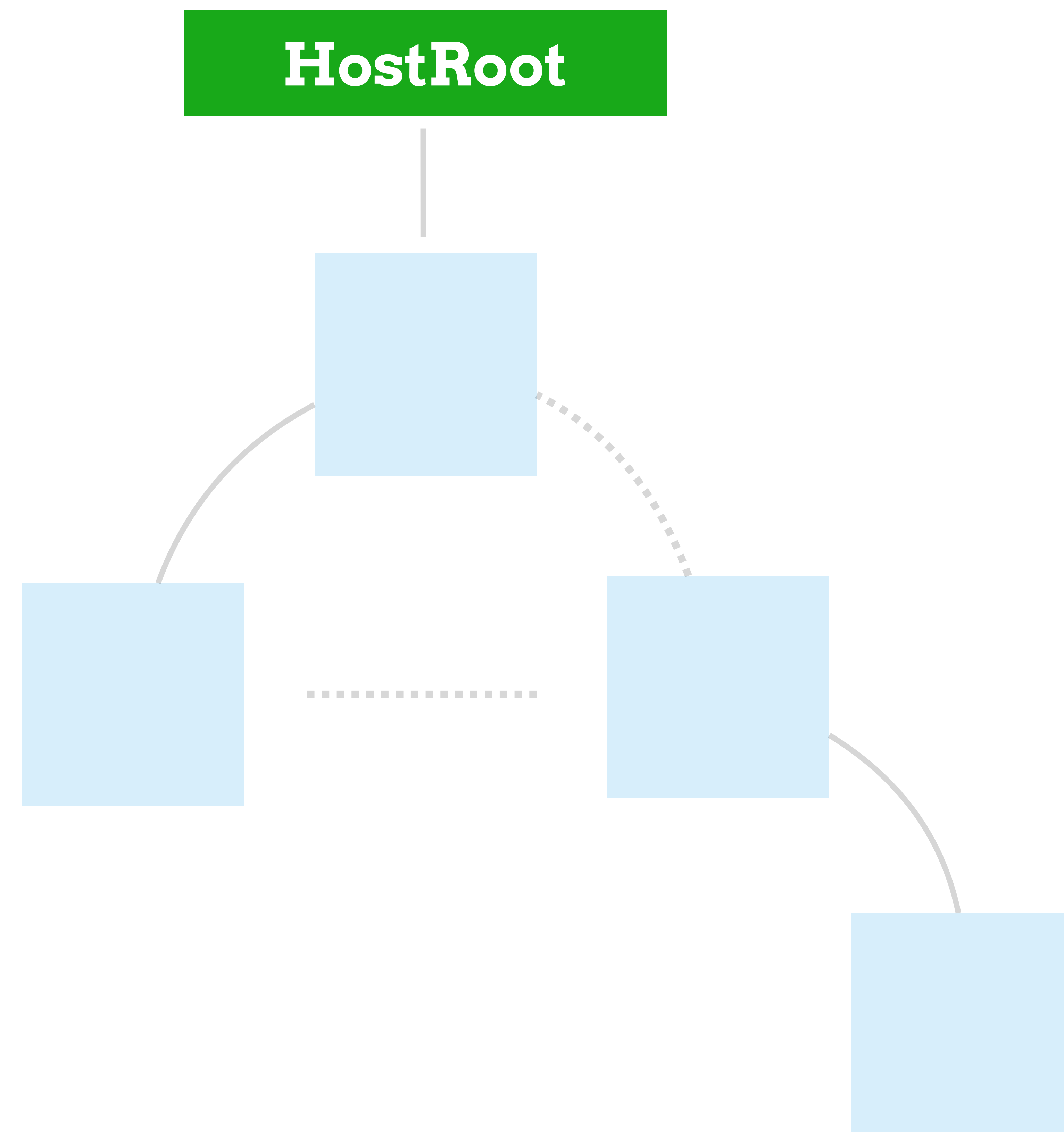
newState = { element: element }

updateHostRoot

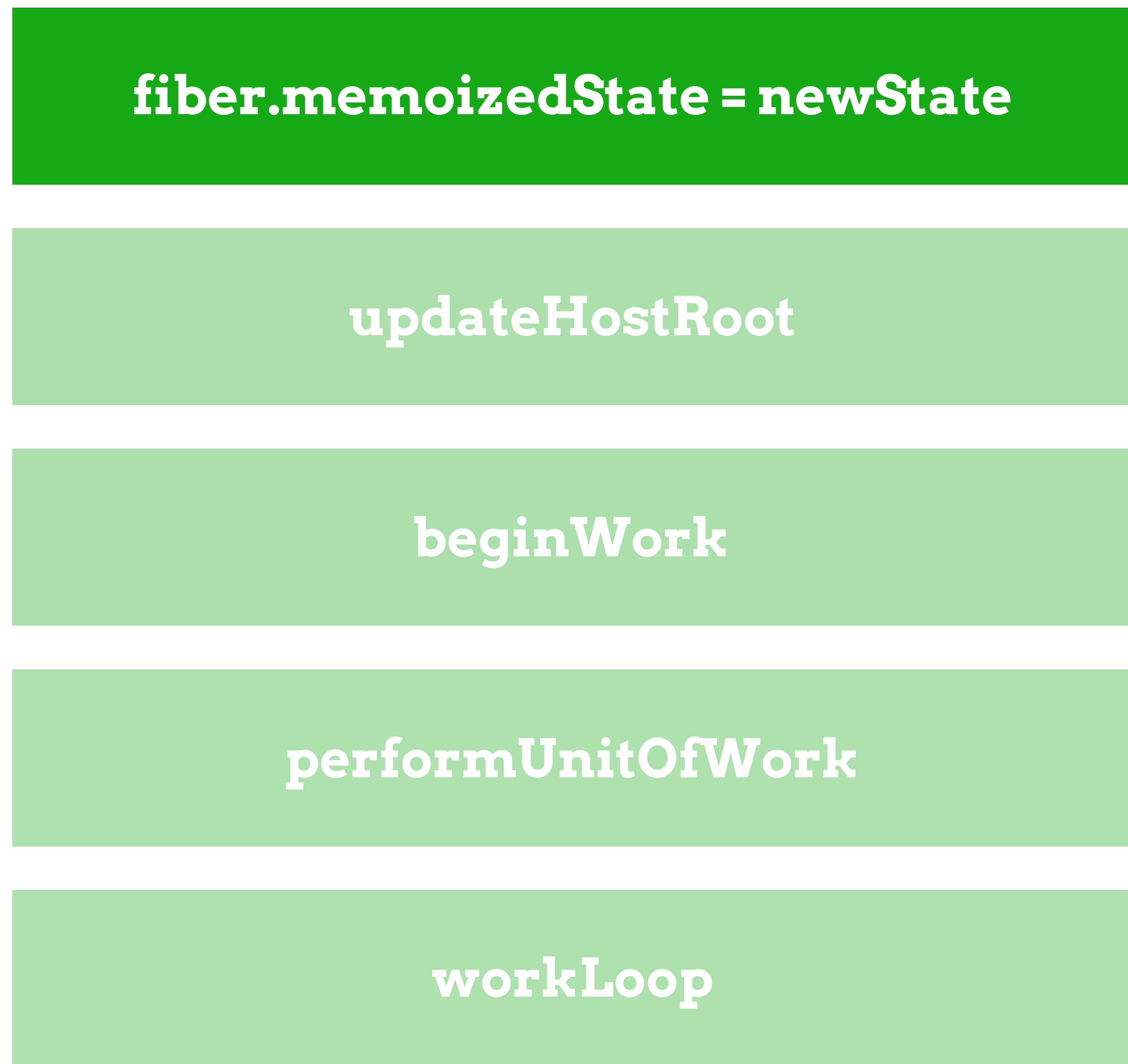
beginWork

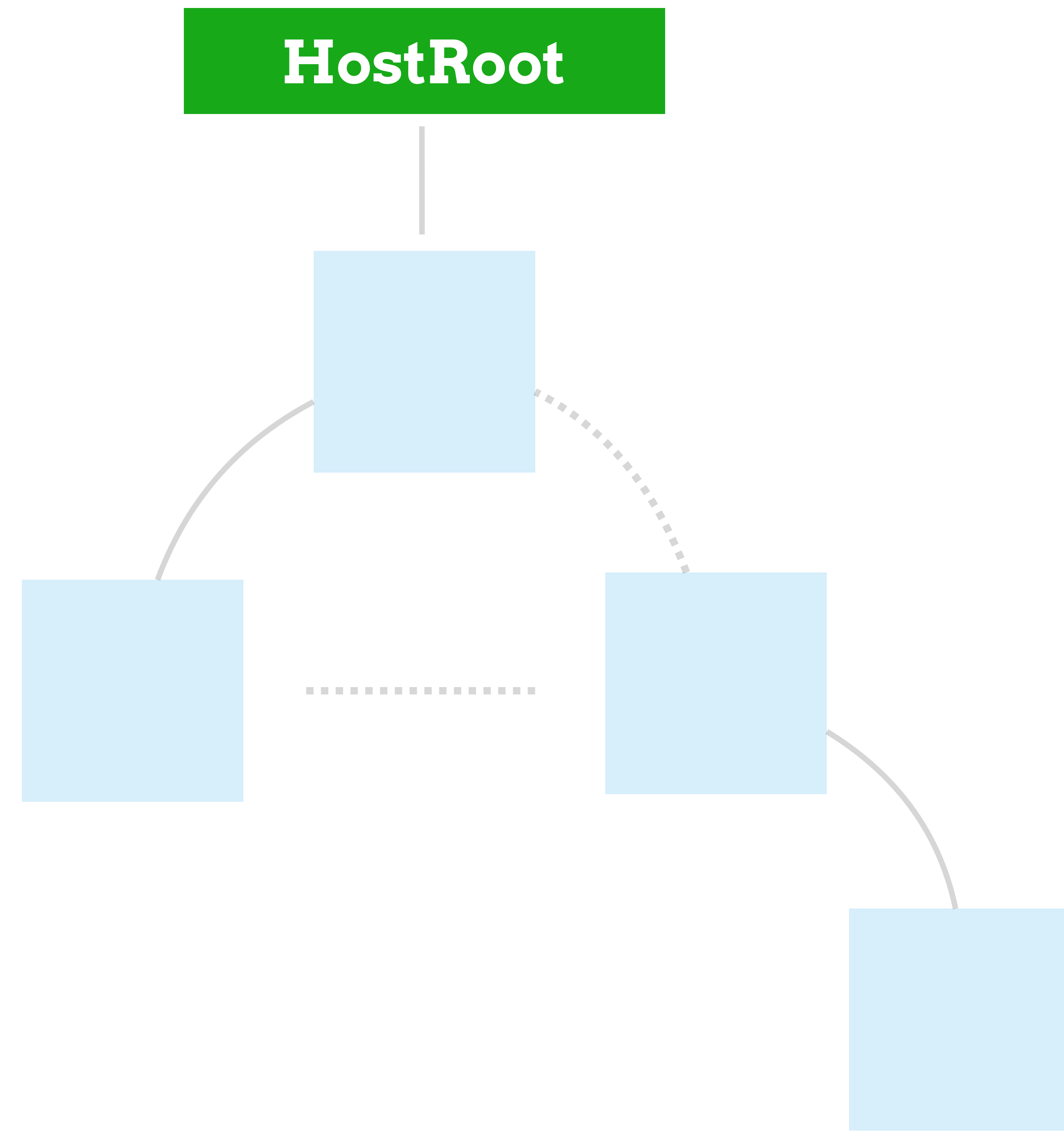
performUnitOfWork

workLoop



@GURLCODE

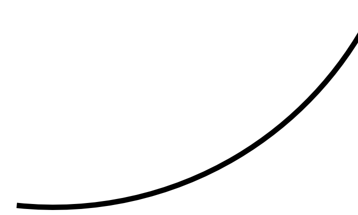




@GURLCODE

FIBER

```
{
    alternate
    tag
    child
    sibling
    return
    memoizedState
    ...
}
```



current state

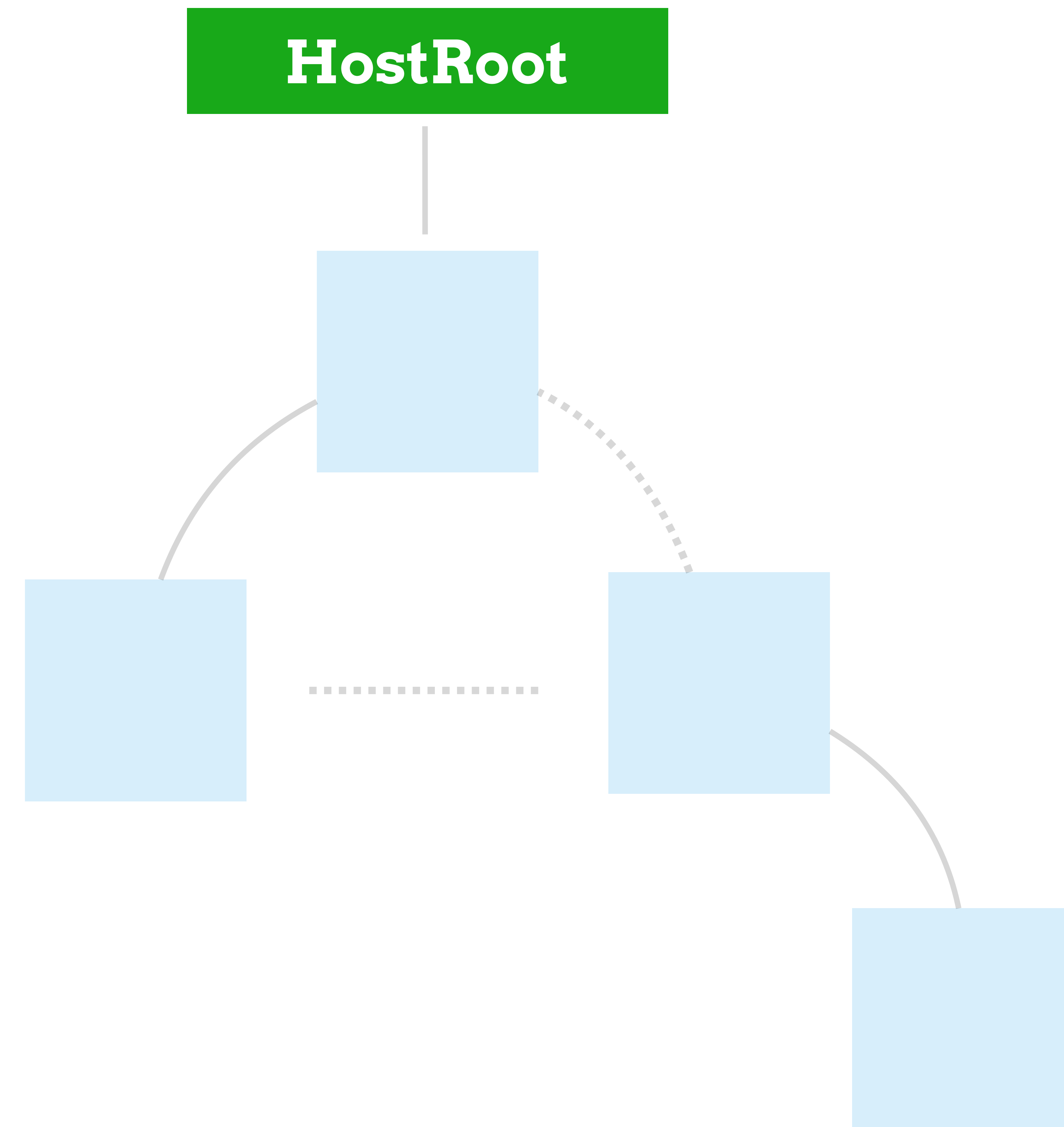
@GURLCODE

nextChild = fiber.memoizedState.element

beginWork

performUnitOfWork

workLoop



@GURLCODE

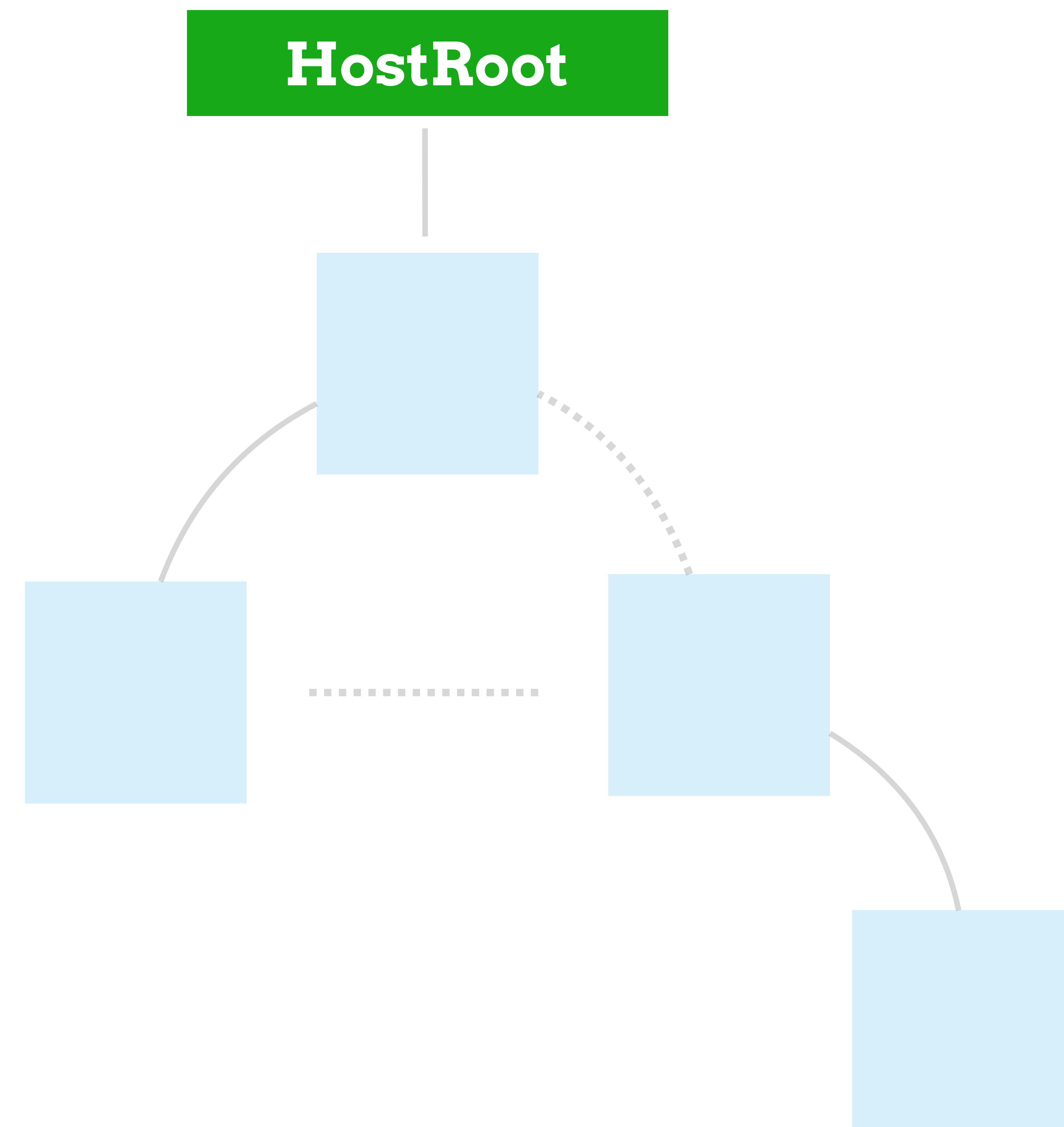
reconcileChildFibers(child, nextChild)

nextChild = fiber.memoizedState.element

beginWork

performUnitOfWork

workLoop



@GURLCODE

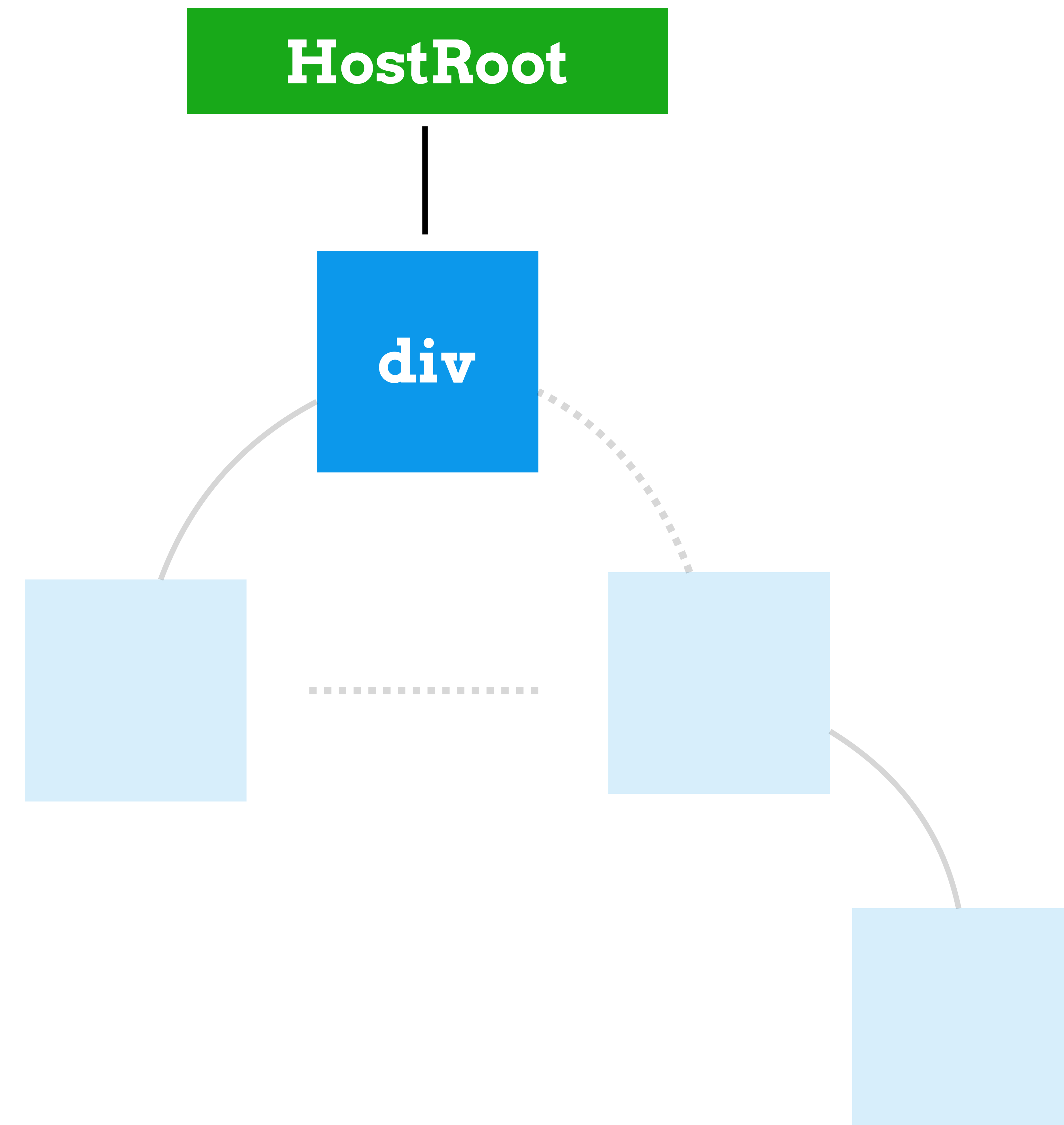
workInProgress.child = createFiber( )

nextChild = fiber.memoizedState.element

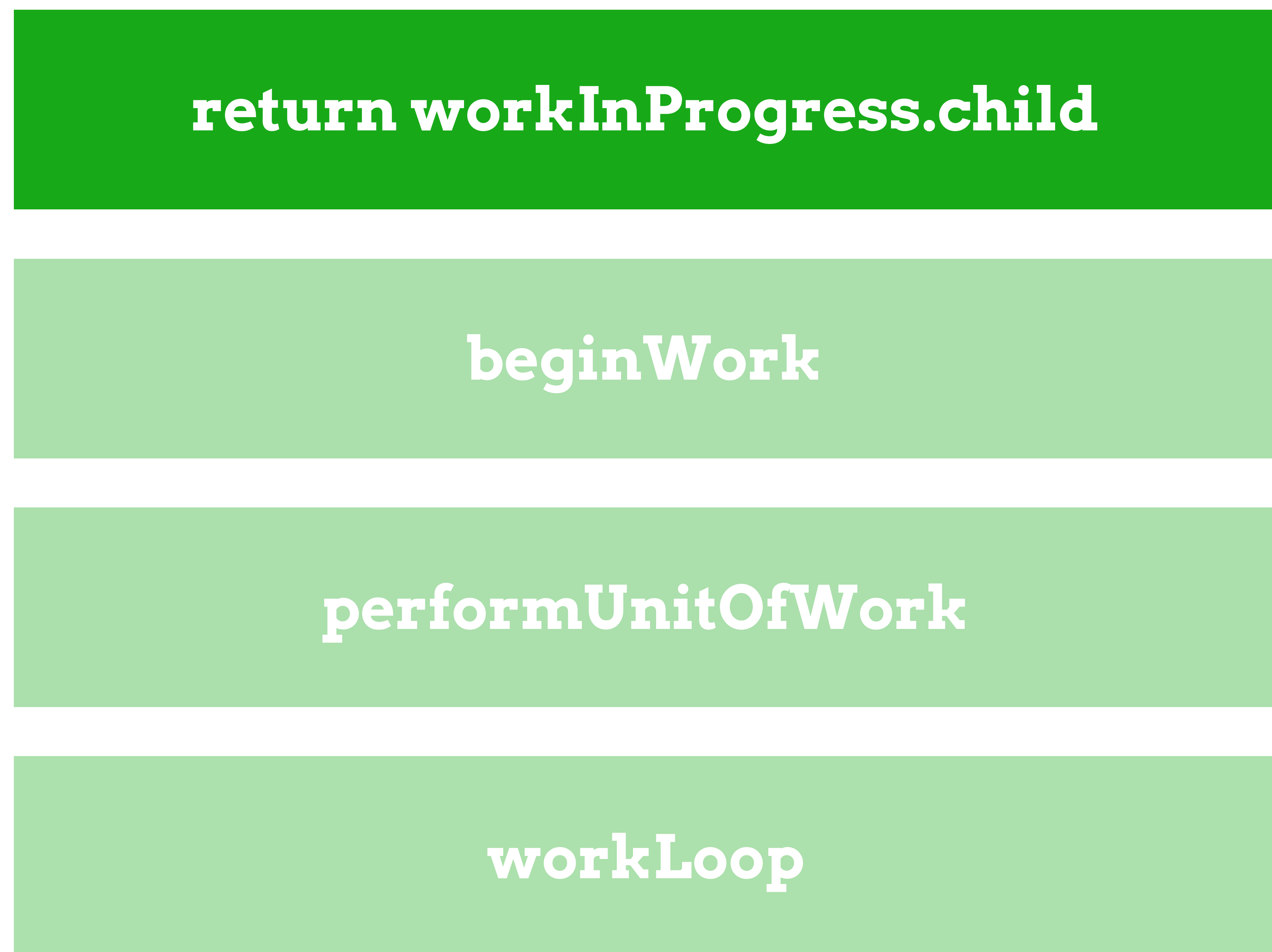
beginWork

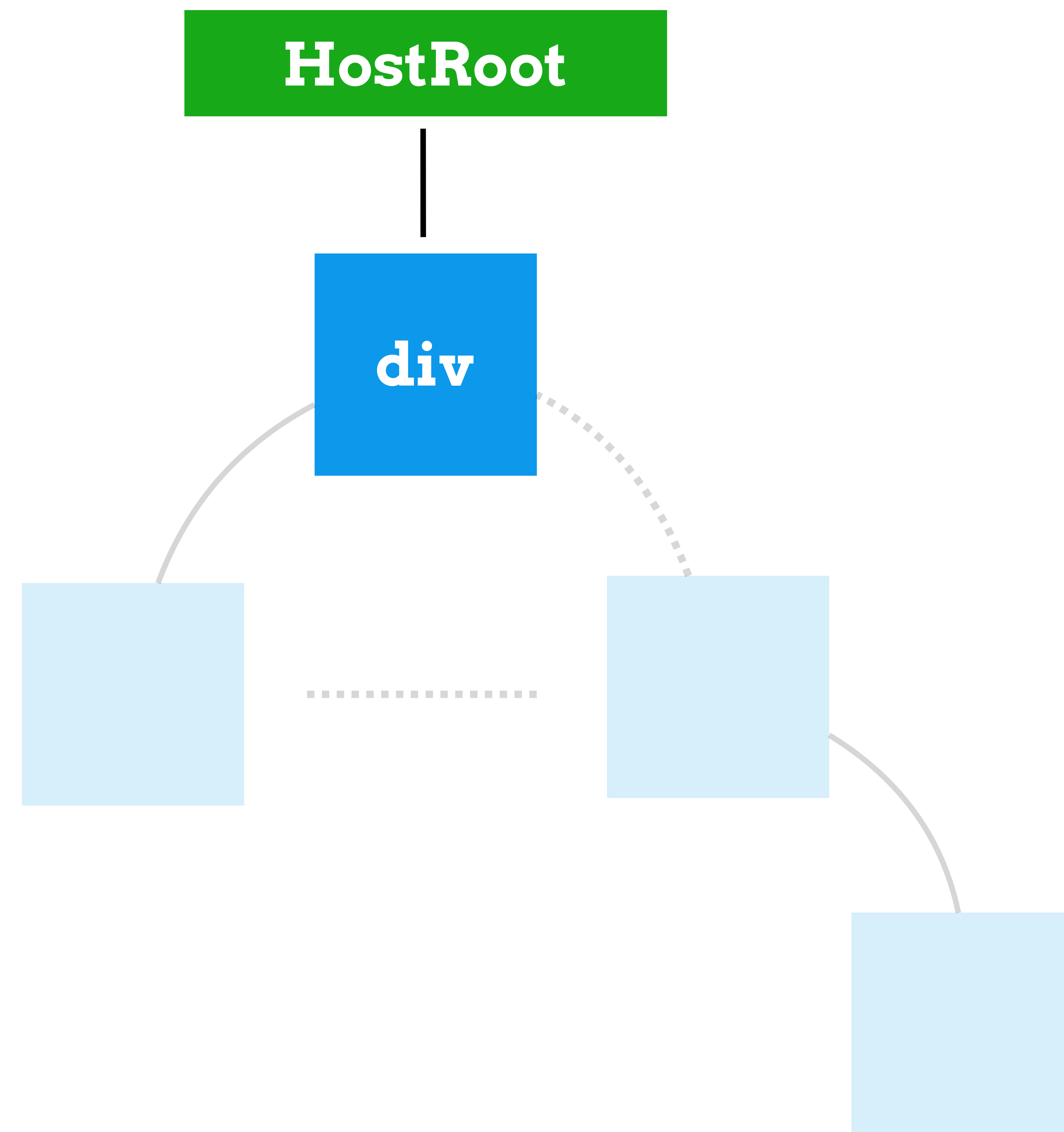
performUnitOfWork

workLoop



@GURLCODE



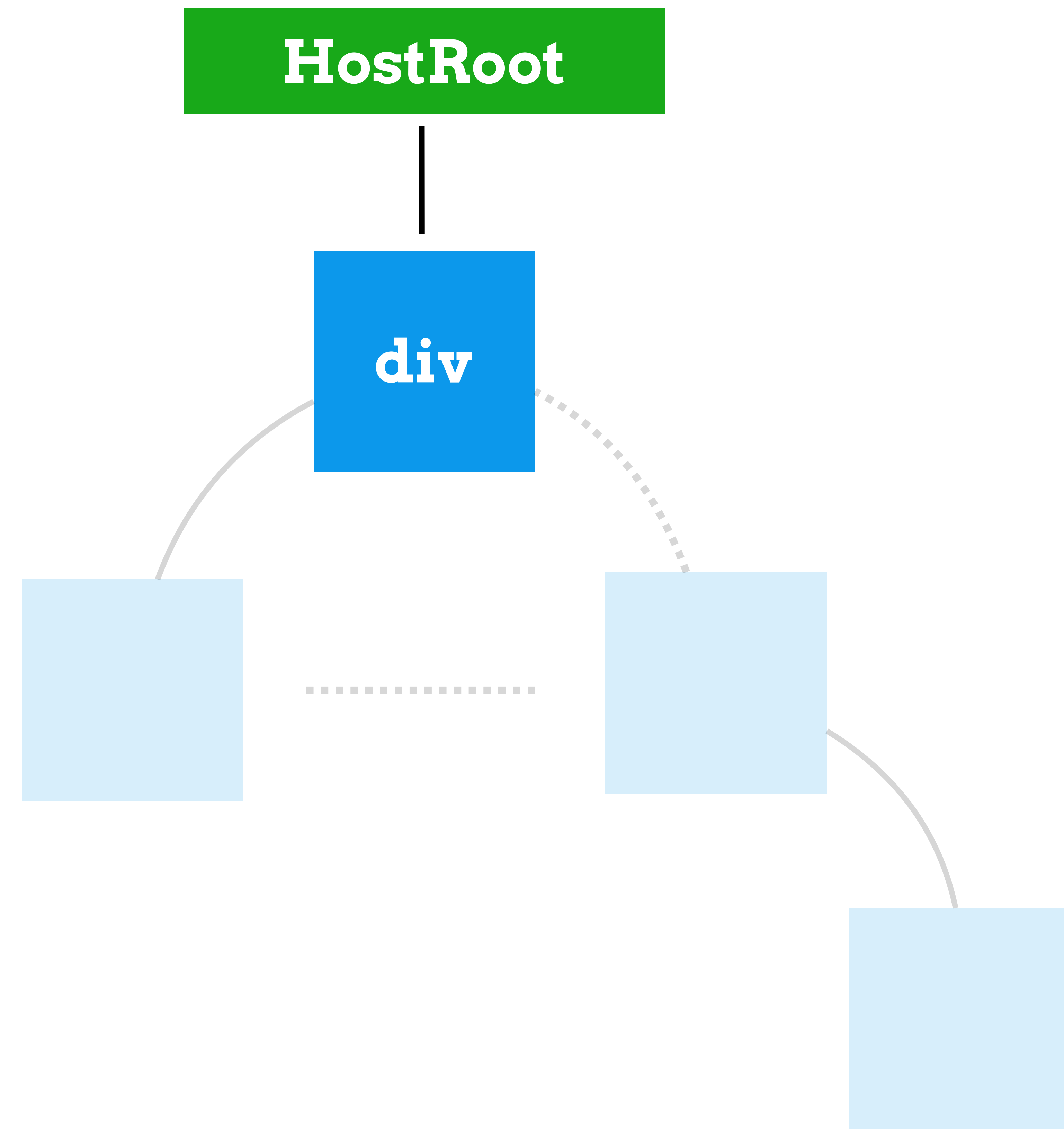


@GURLCODE

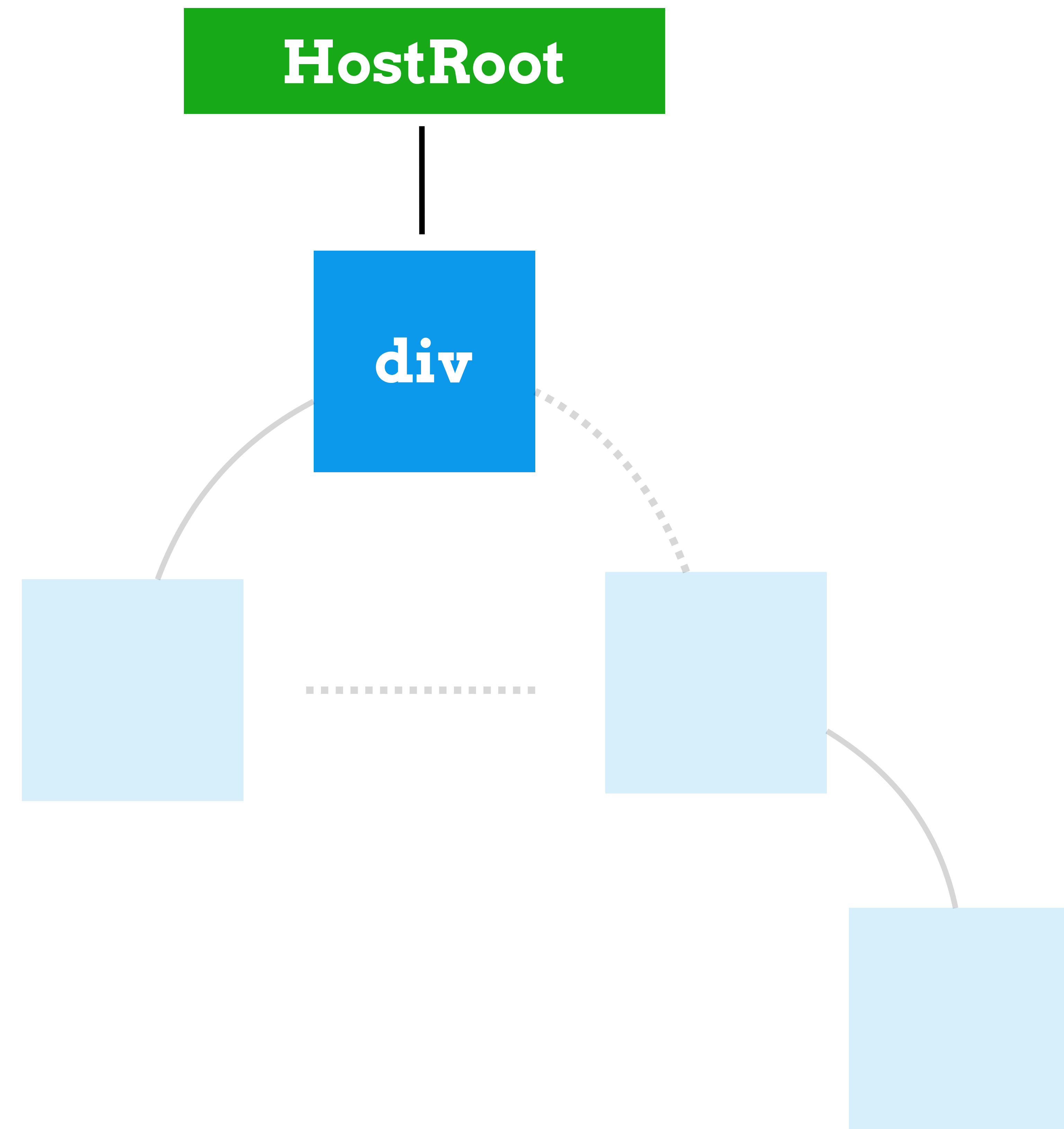
return workInProgress.child

performUnitOfWork

workLoop



@GURLCODE



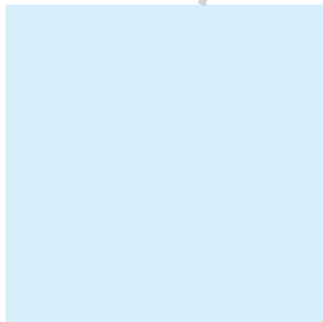
return workInProgress.child

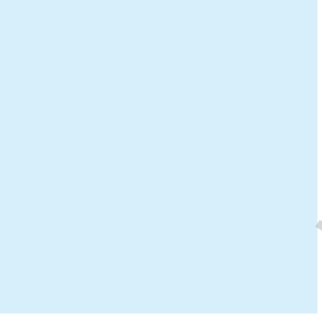
workLoop

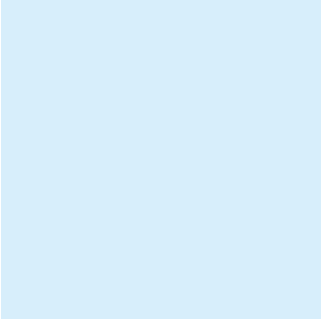
@GURLCODE

HostRoot

div

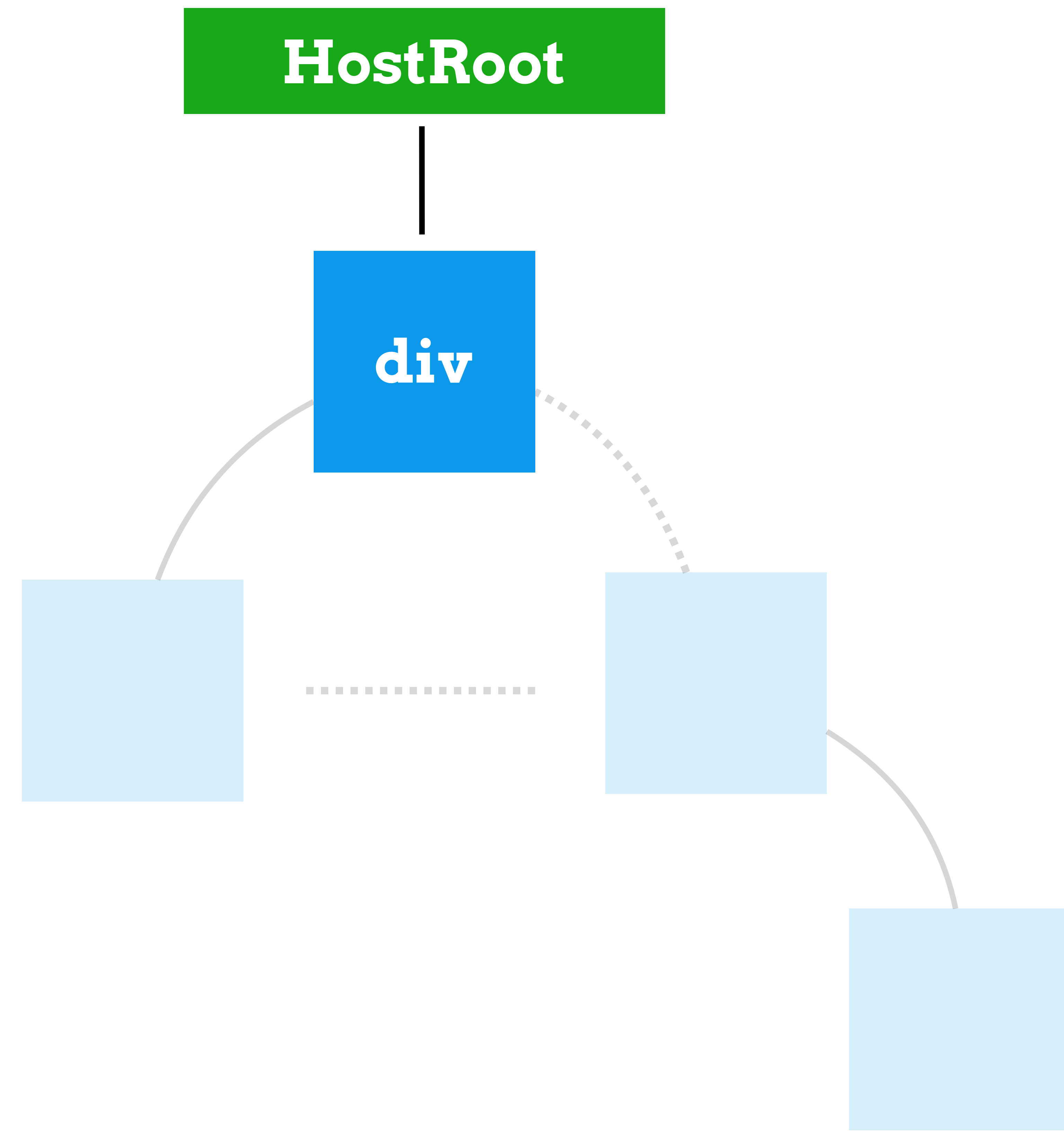






nextUnitOfWork = performUnitWork

@GURLCODE

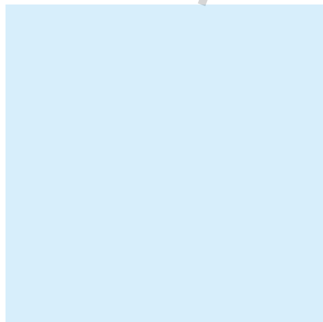


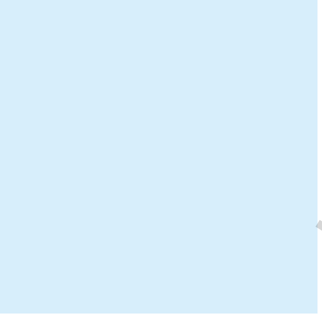
nextUnitOfWork !== null

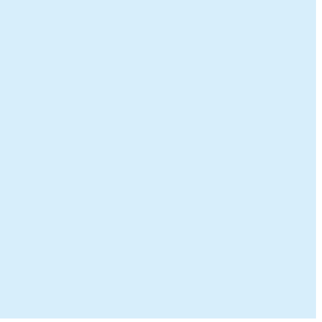
@GURLCODE

HostRoot

div

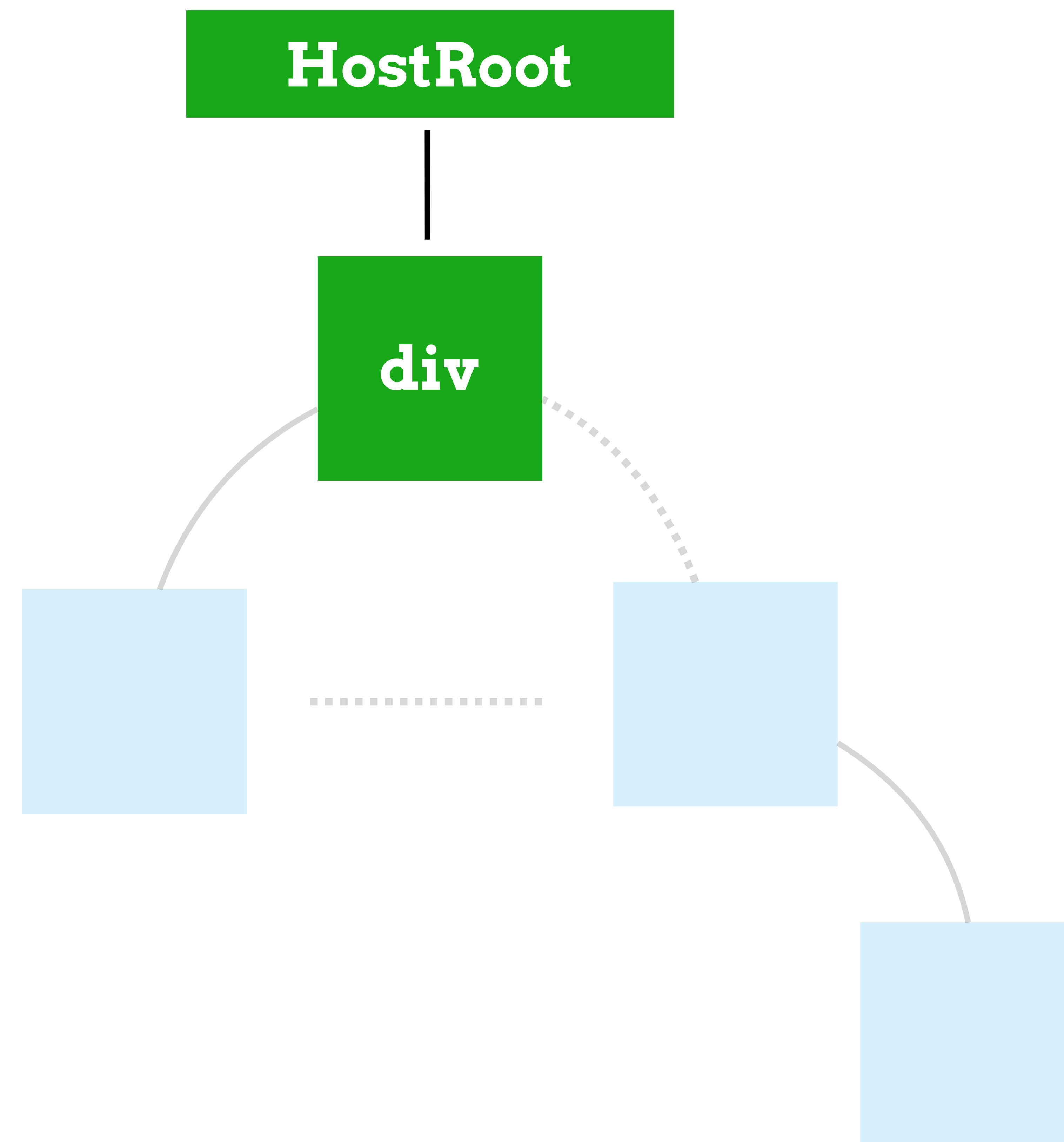






workLoop

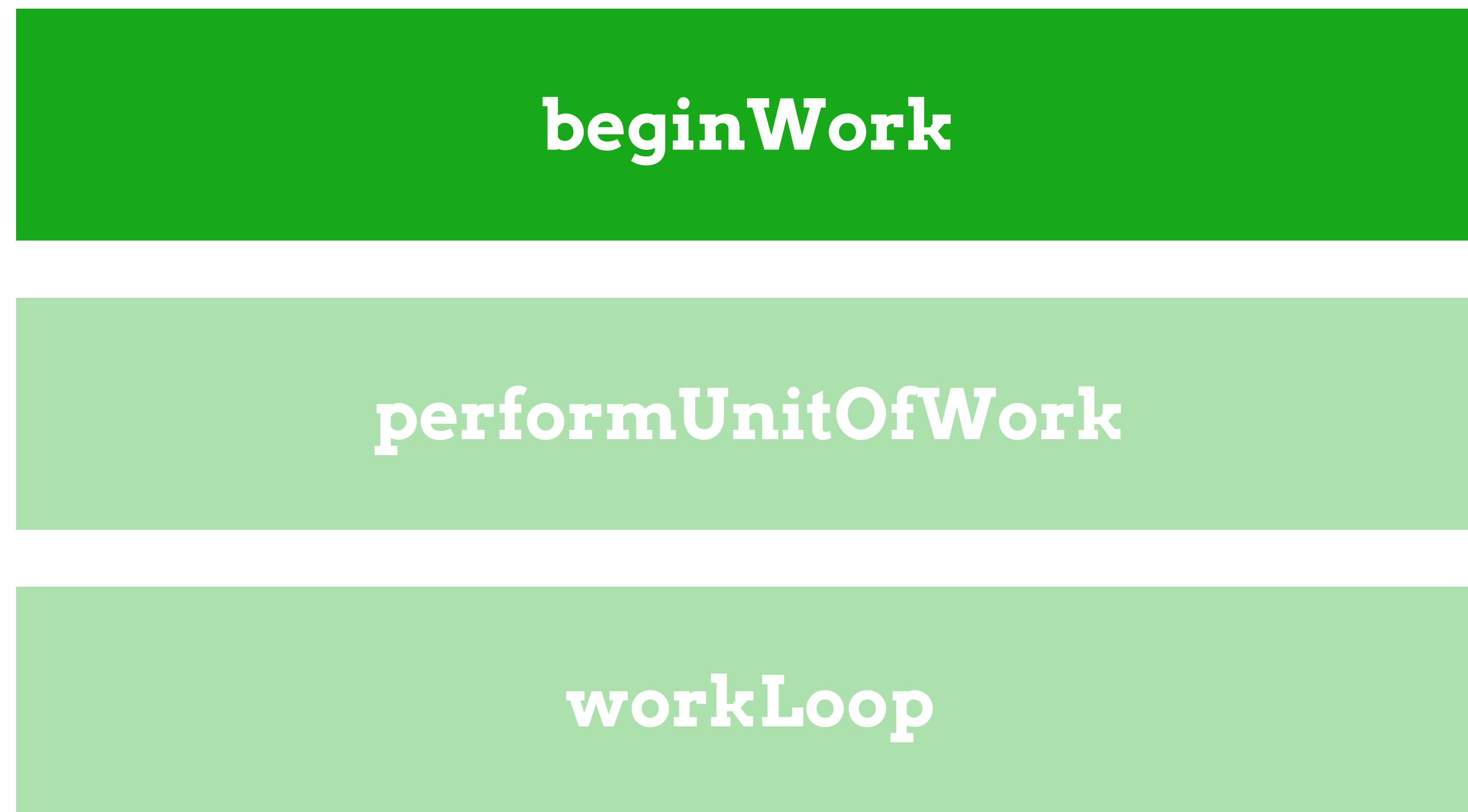
@GURLCODE

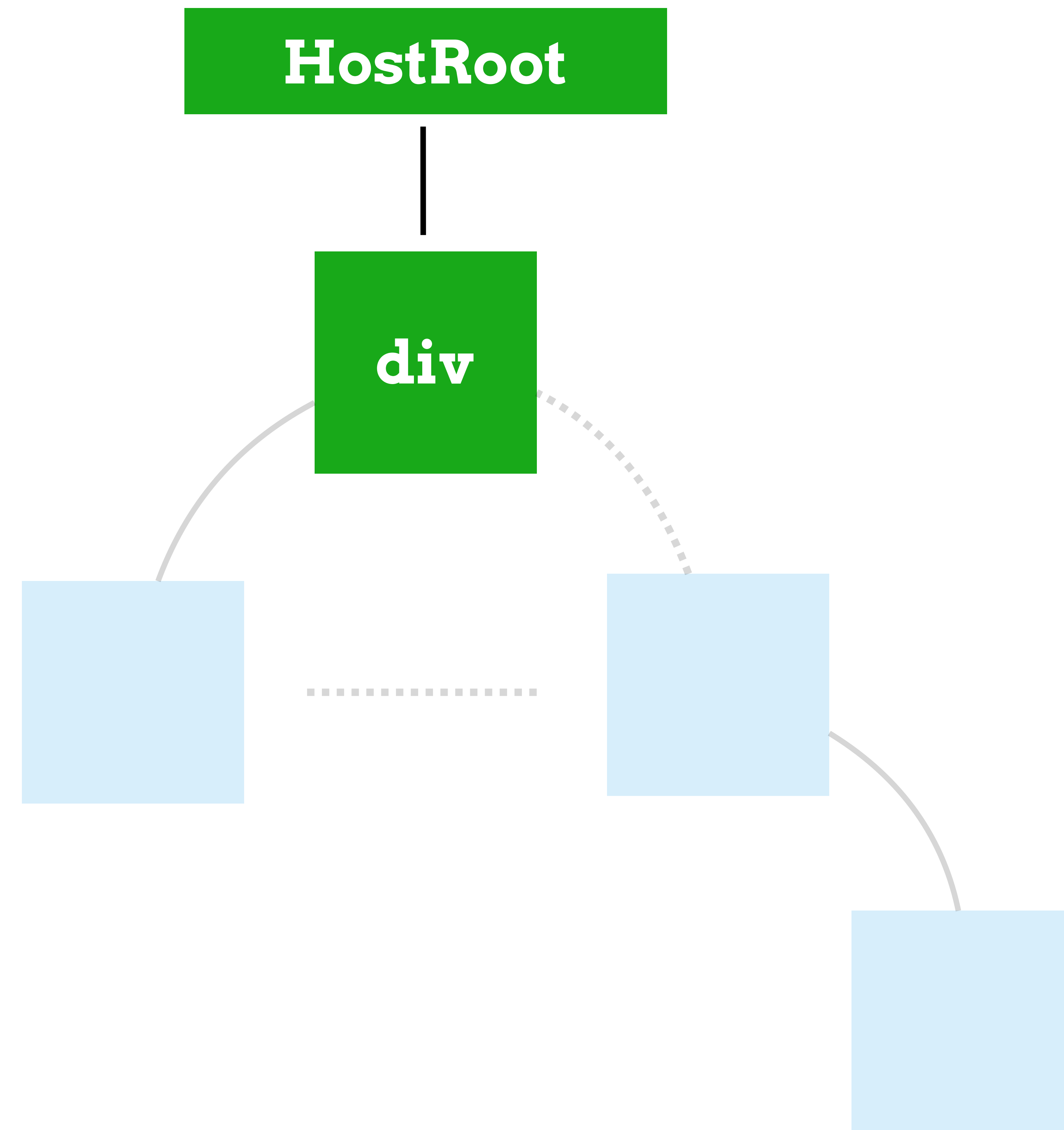


performUnitOfWork

workLoop

@GURLCODE





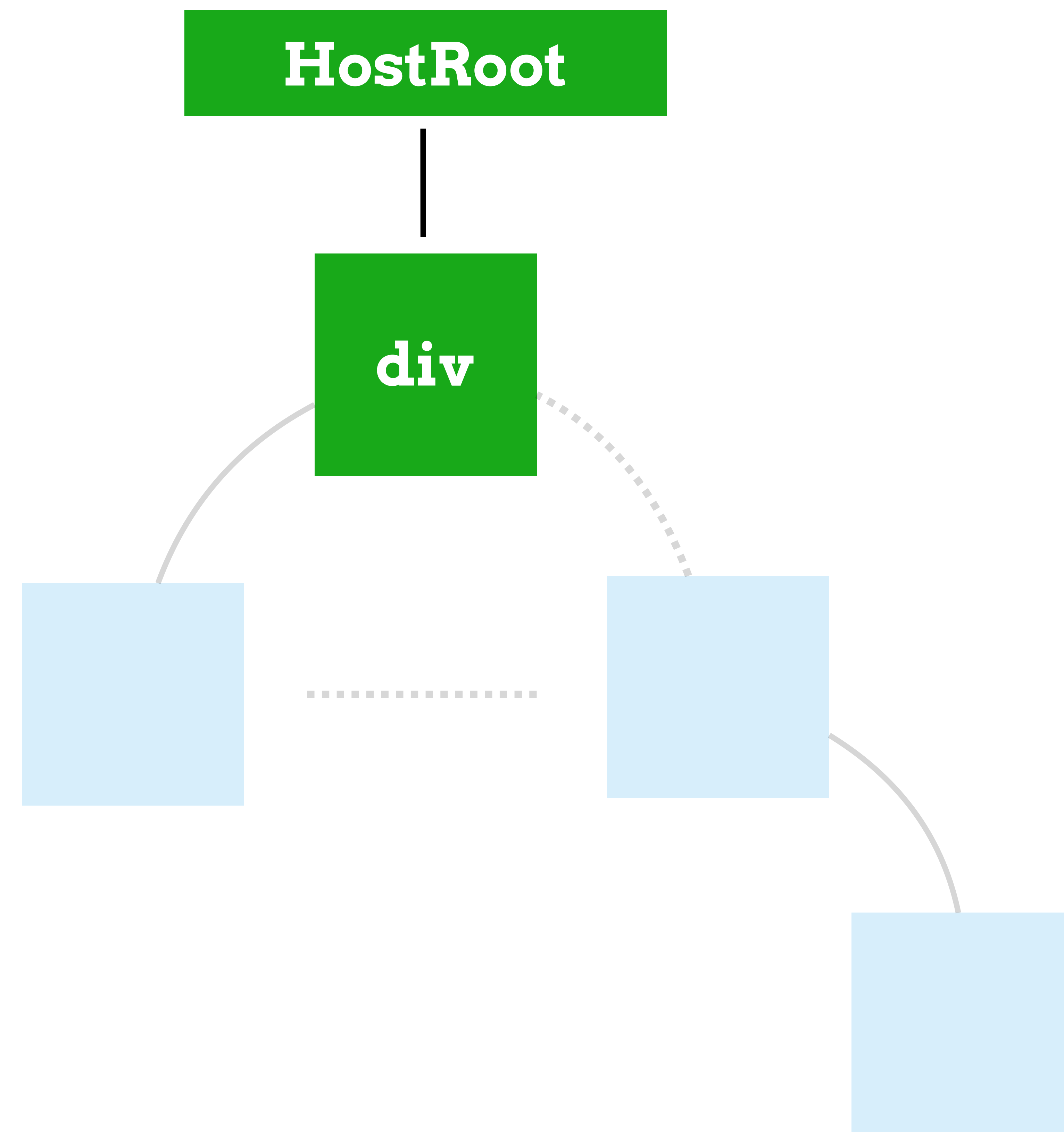
@GURLCODE

updateHostComponent

beginWork

performUnitOfWork

workLoop



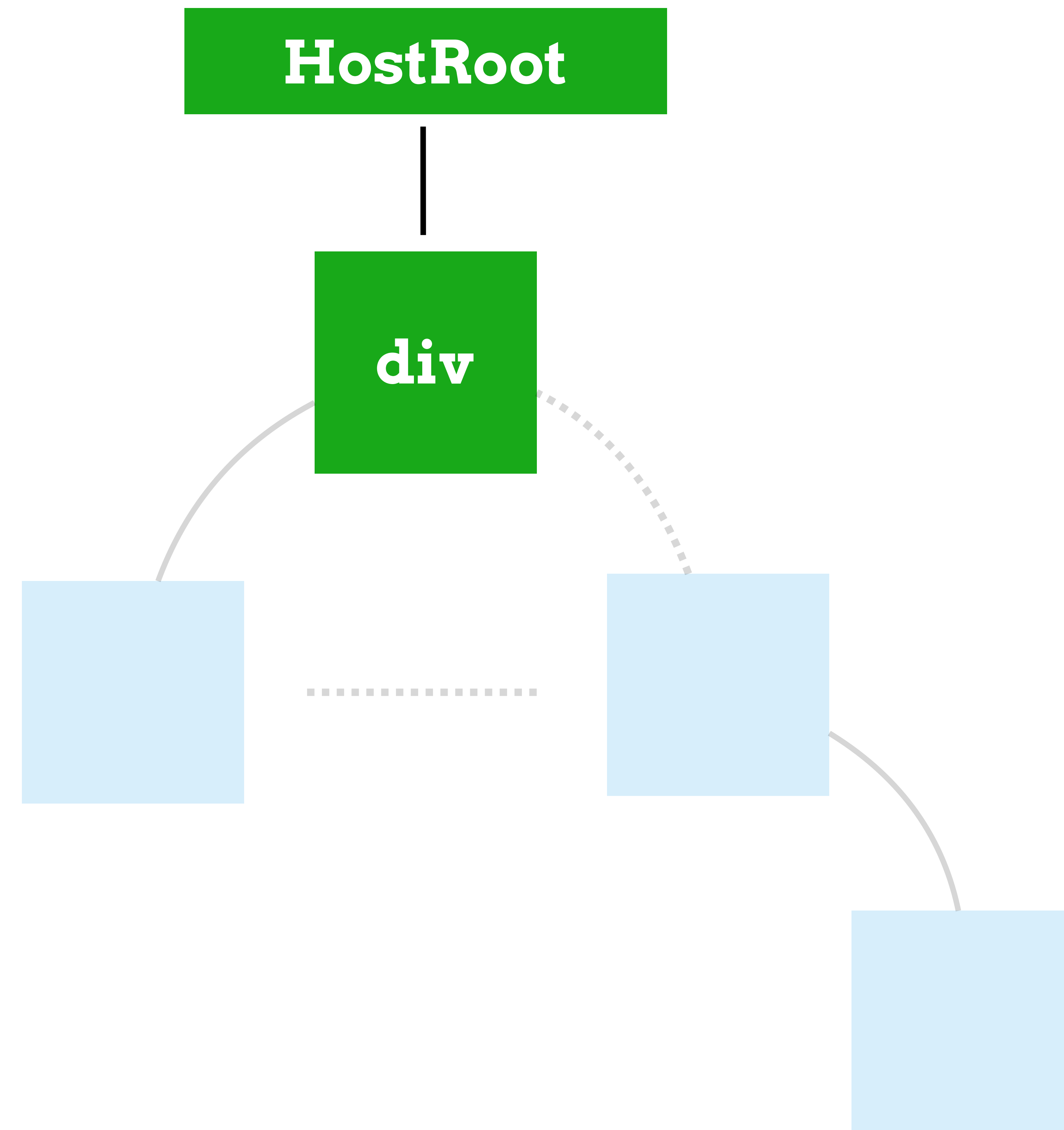
@GURLCODE

nextChild = fiber.pendingProps.children

beginWork

performUnitOfWork

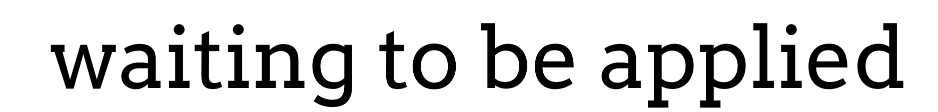
workLoop

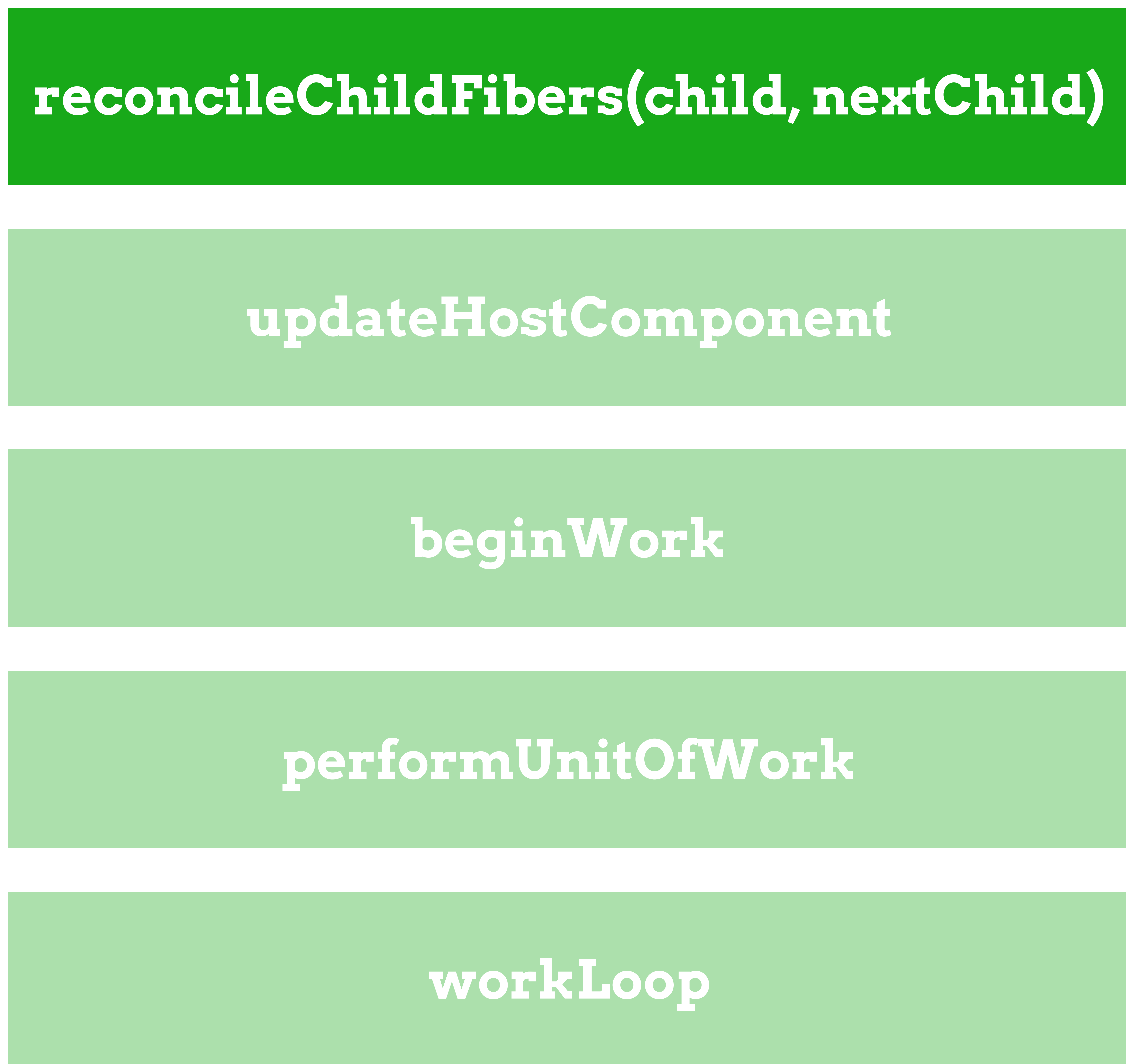


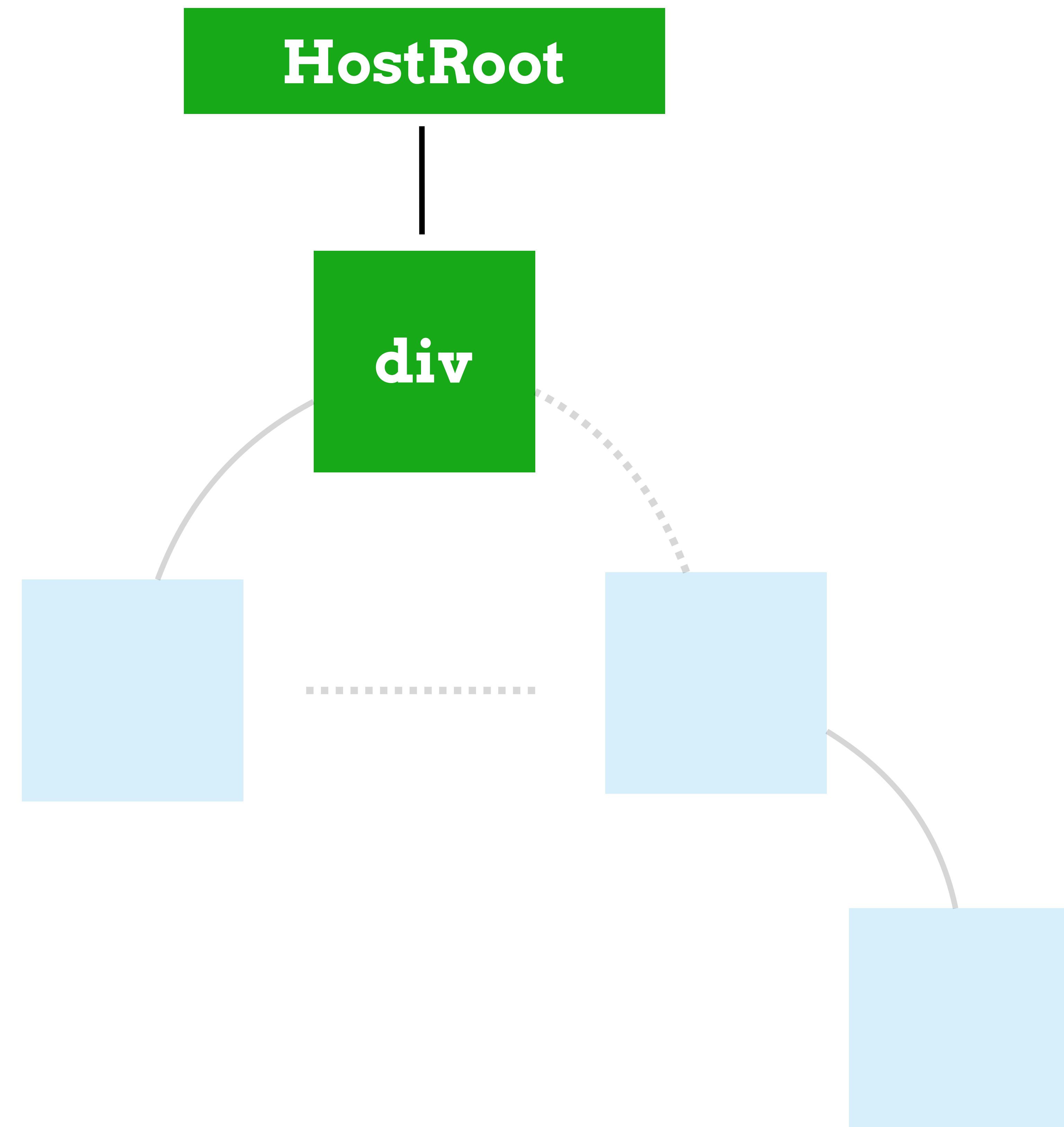
@GURLCODE

FIBER

```
{
    alternate
    tag
    child
    sibling
    return
    memoizedState
    pendingProps
    ...
}
```







@GURLCODE

reconcileChildrenArray

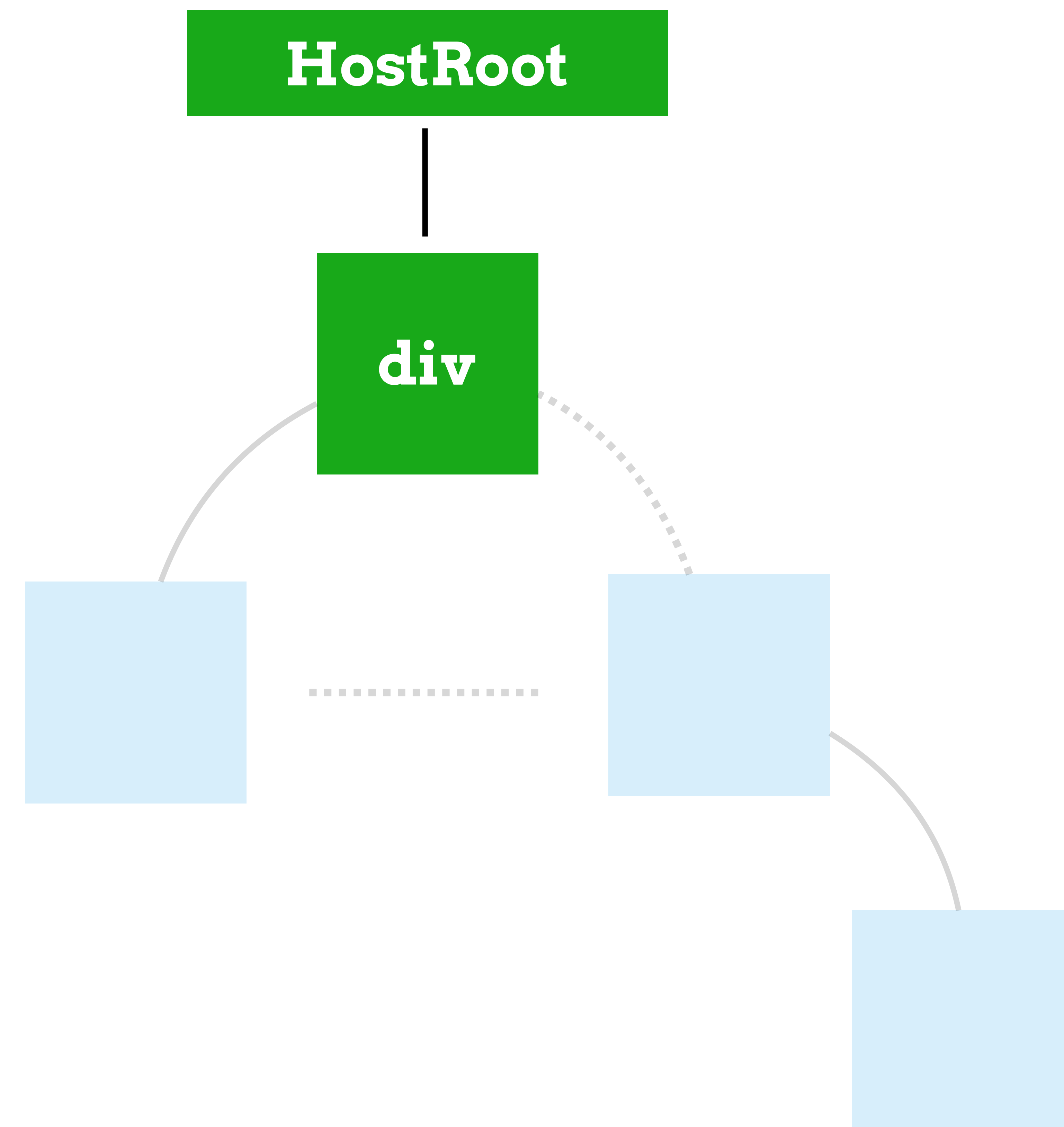
reconcileChildFibers

updateHostComponent

beginWork

performUnitOfWork

workLoop



@GURLCODE

child = createFiber()

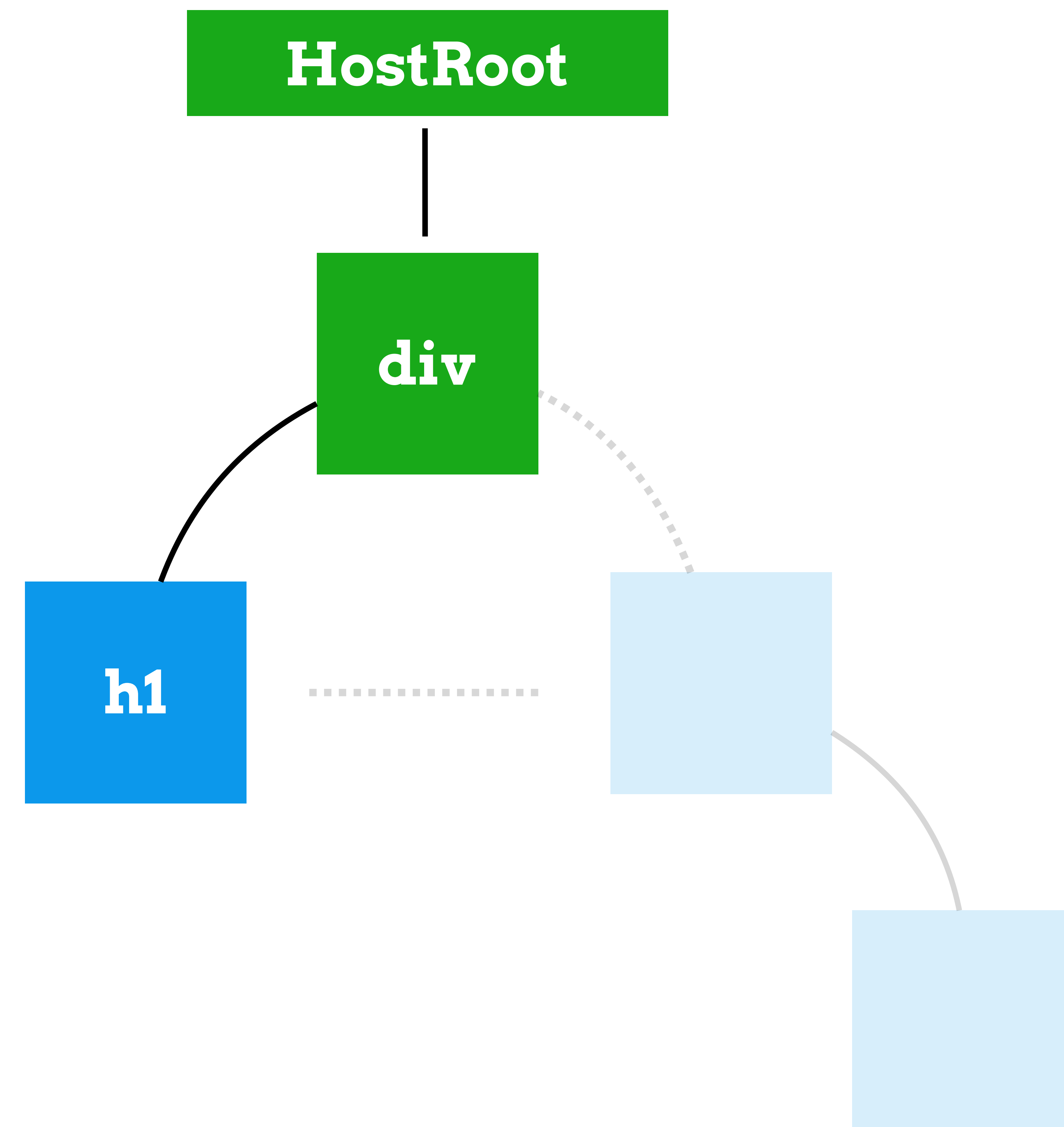
reconcileChildFibers

updateHostComponent

beginWork

performUnitOfWork

workLoop



@GURLCODE

child.sibling = createFiber( )

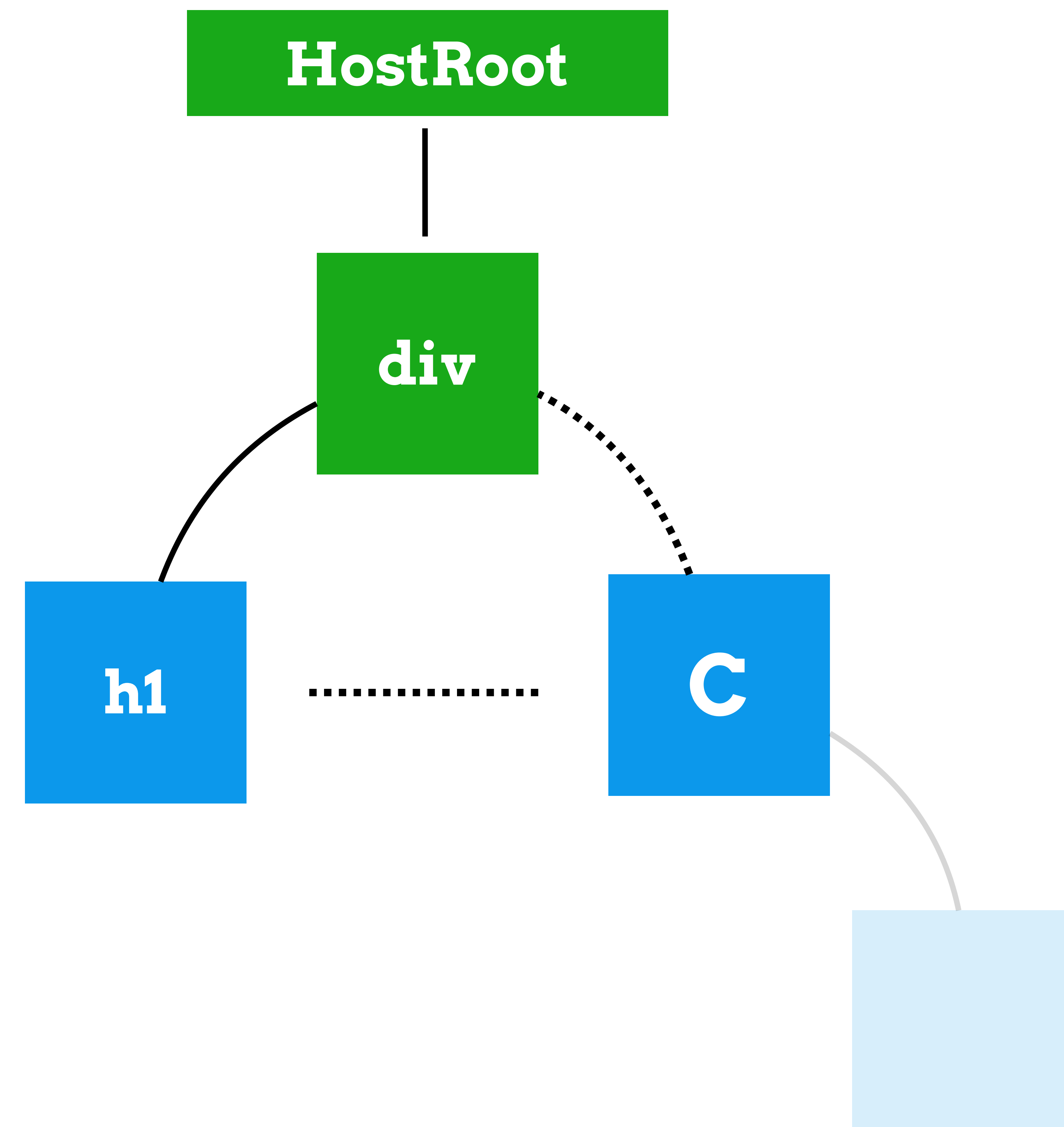
reconcileChildFibers

updateHostComponent

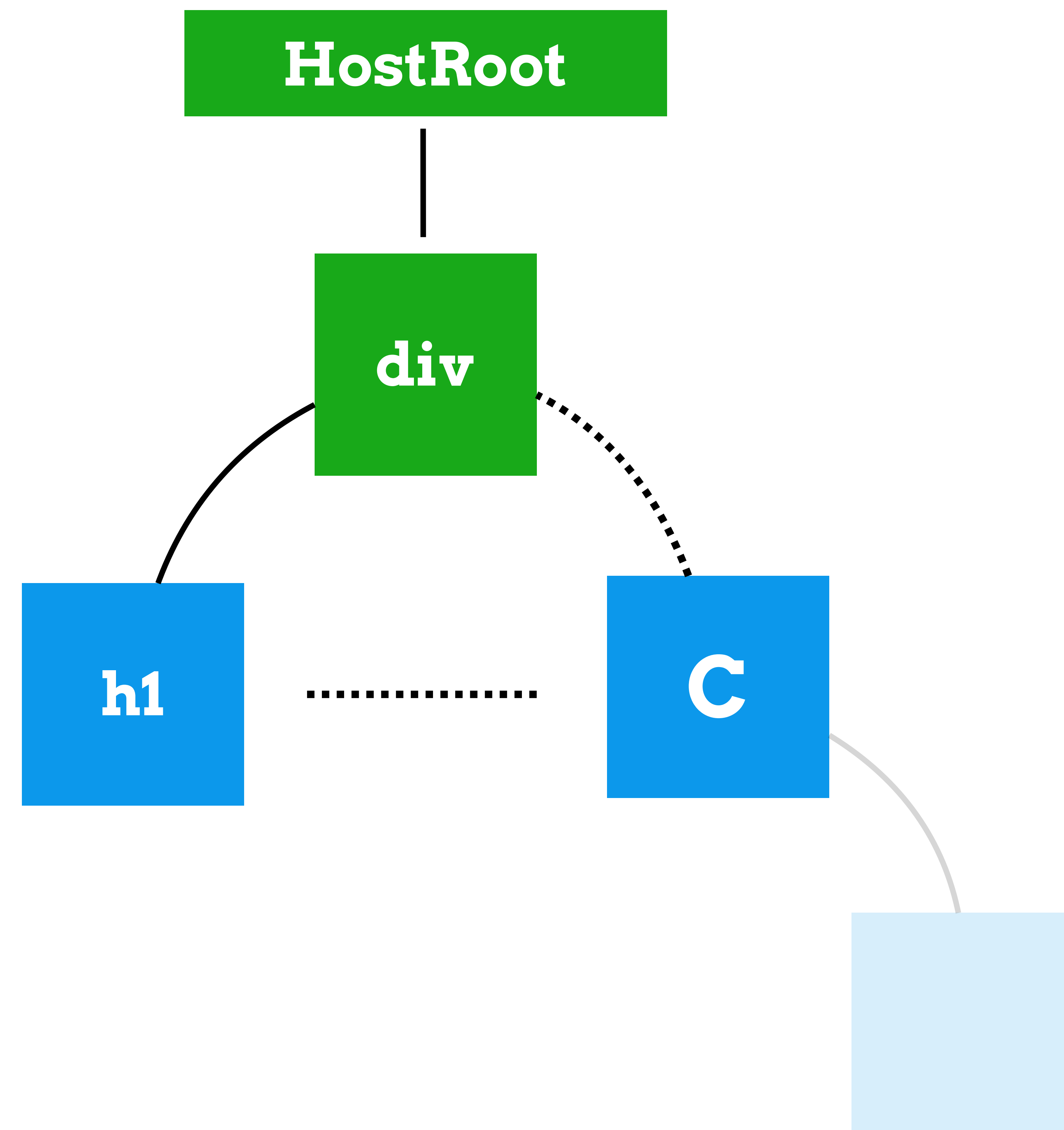
beginWork

performUnitOfWork

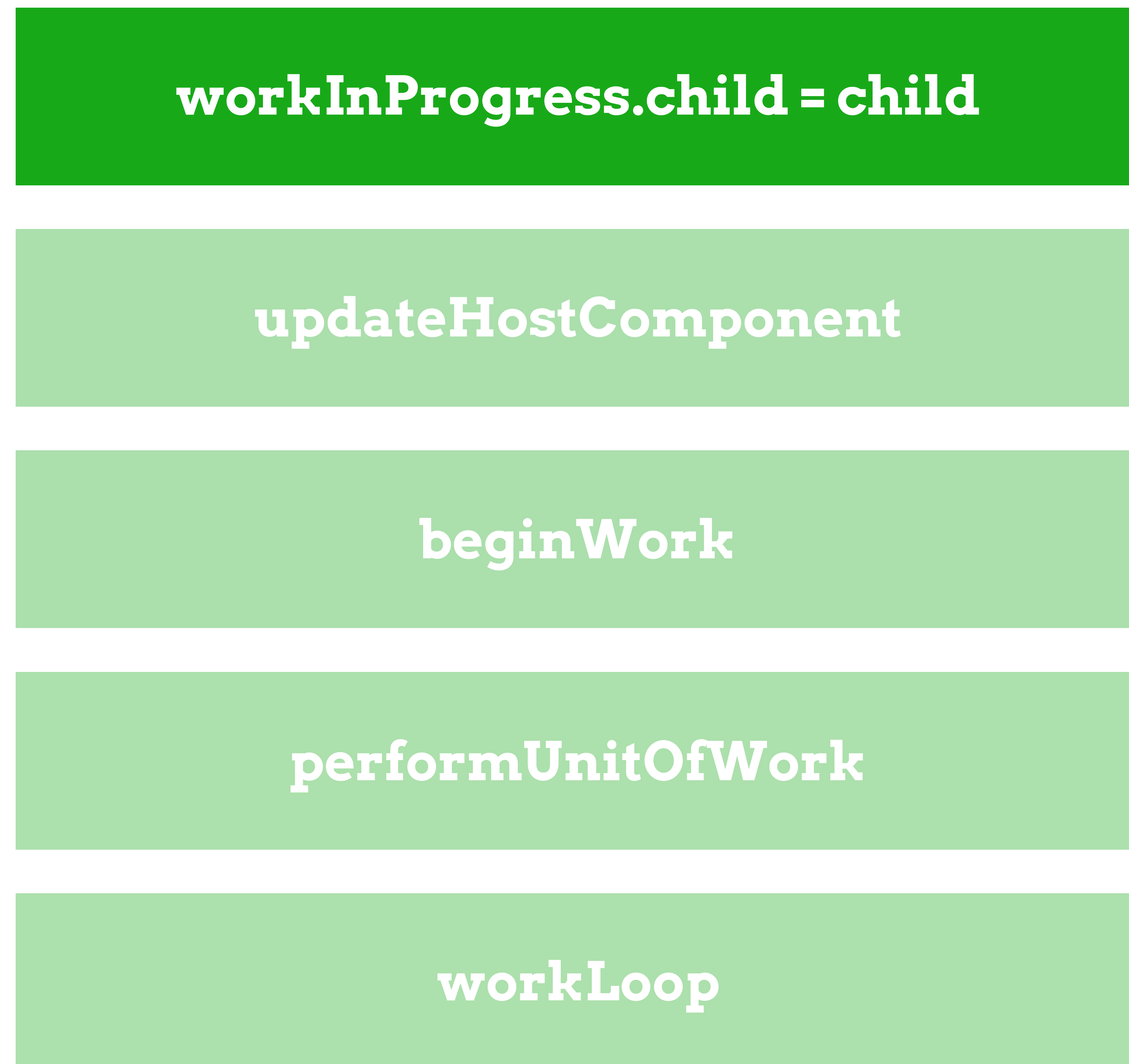
workLoop

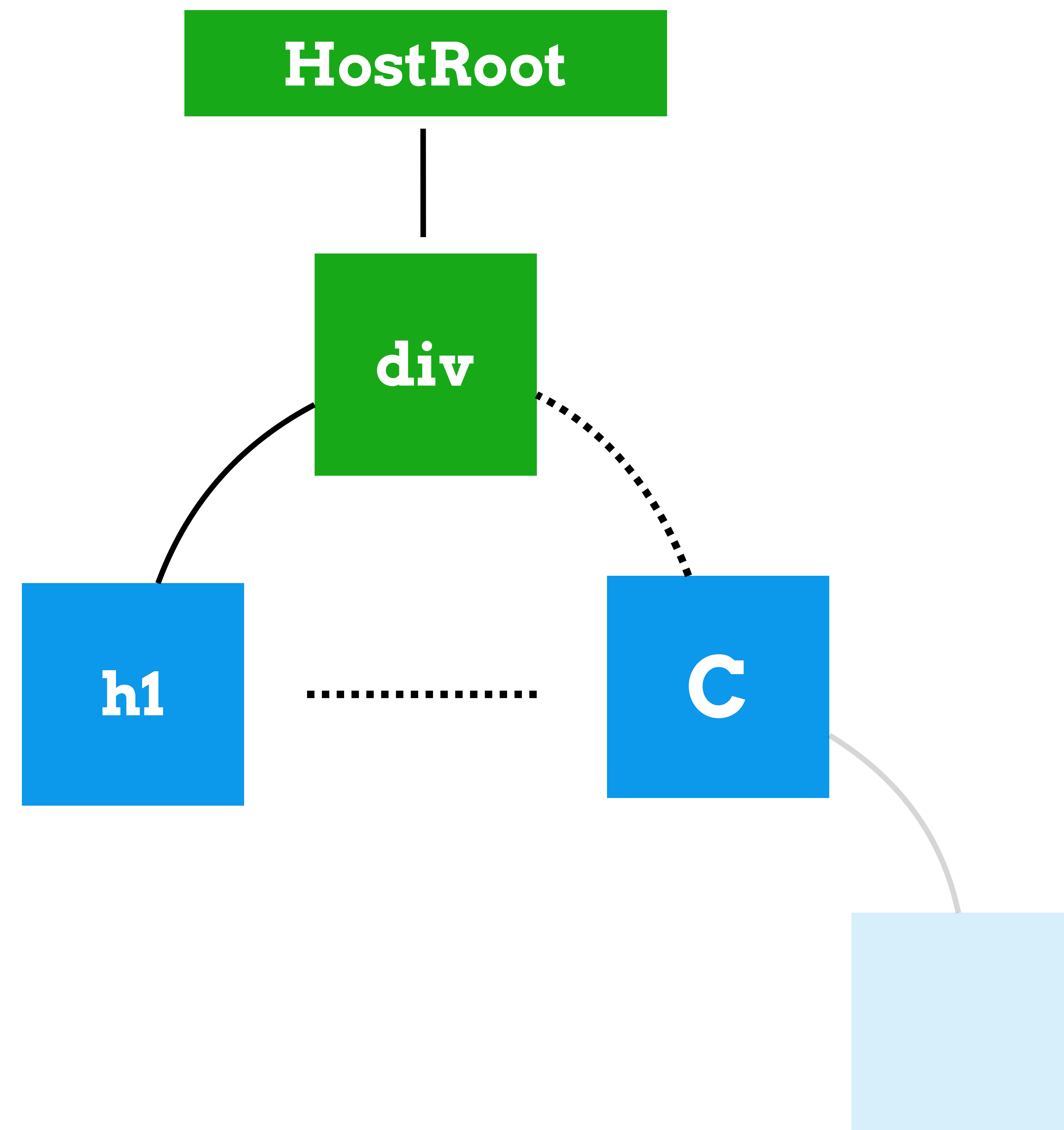


@GURLCODE

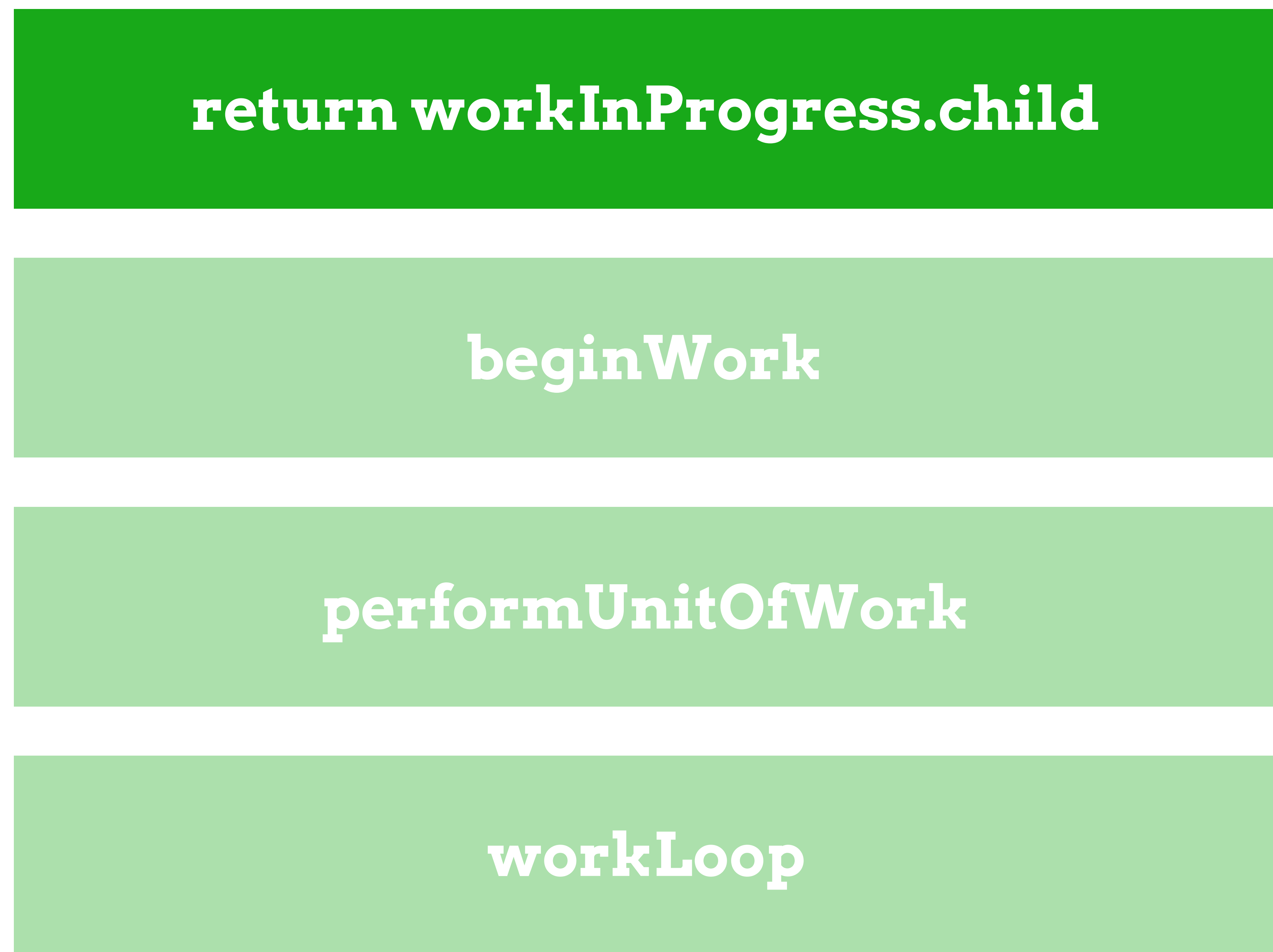


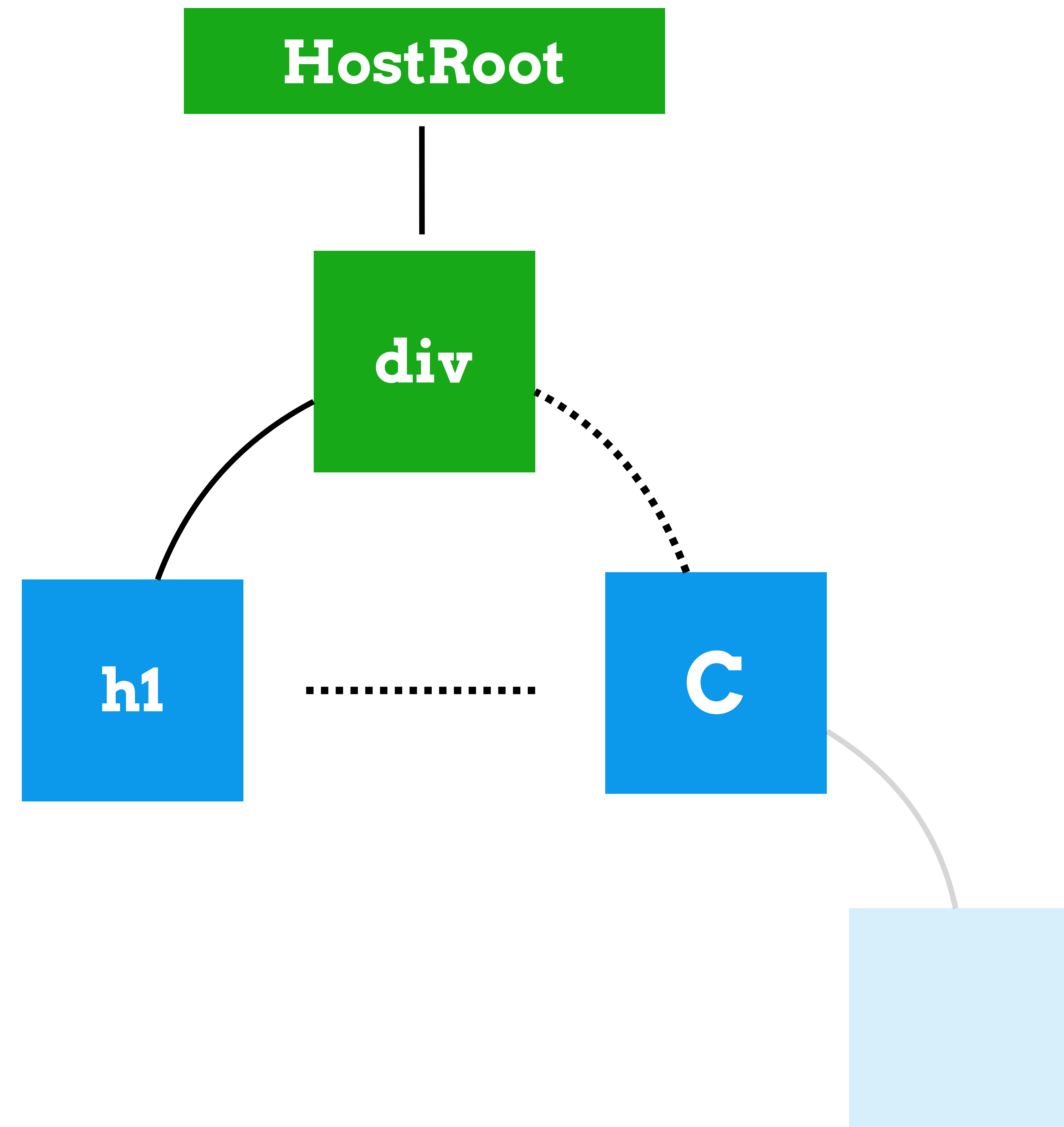
@GURLCODE





@GURLCODE



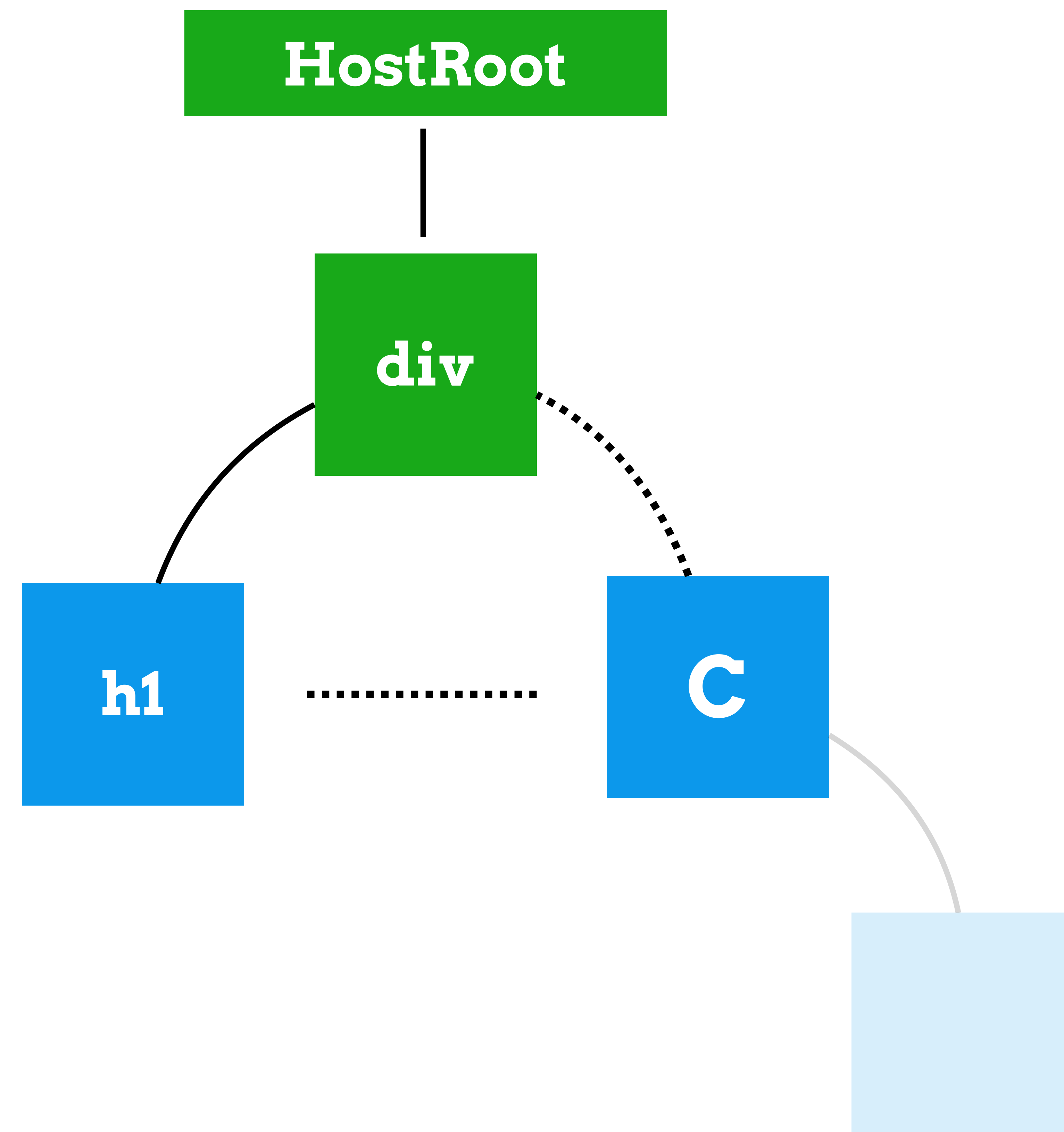


@GURLCODE

return workInProgress.child

performUnitOfWork

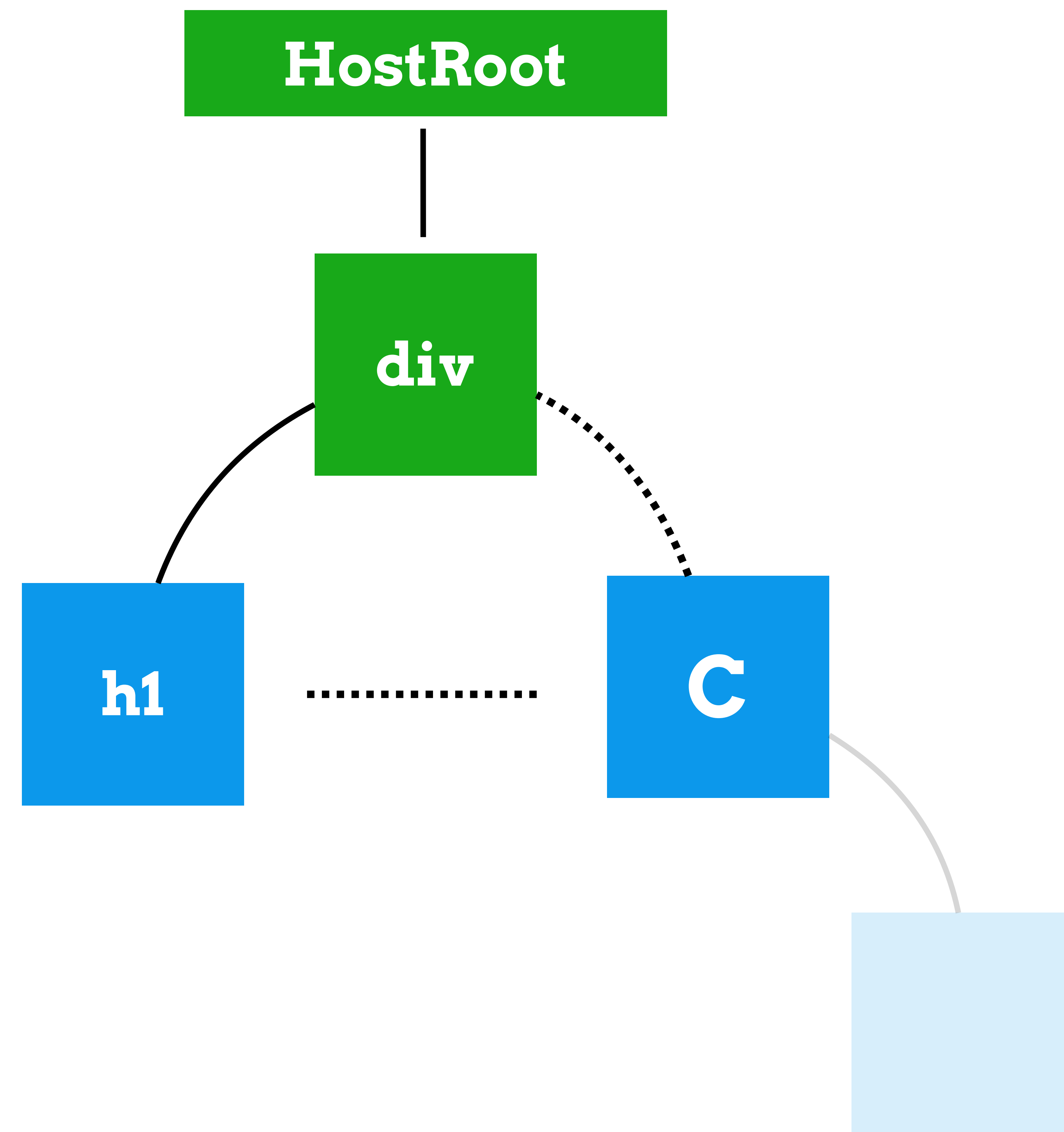
workLoop



@GURLCODE

return workInProgress.child

workLoop



@GURLCODE

HostRoot

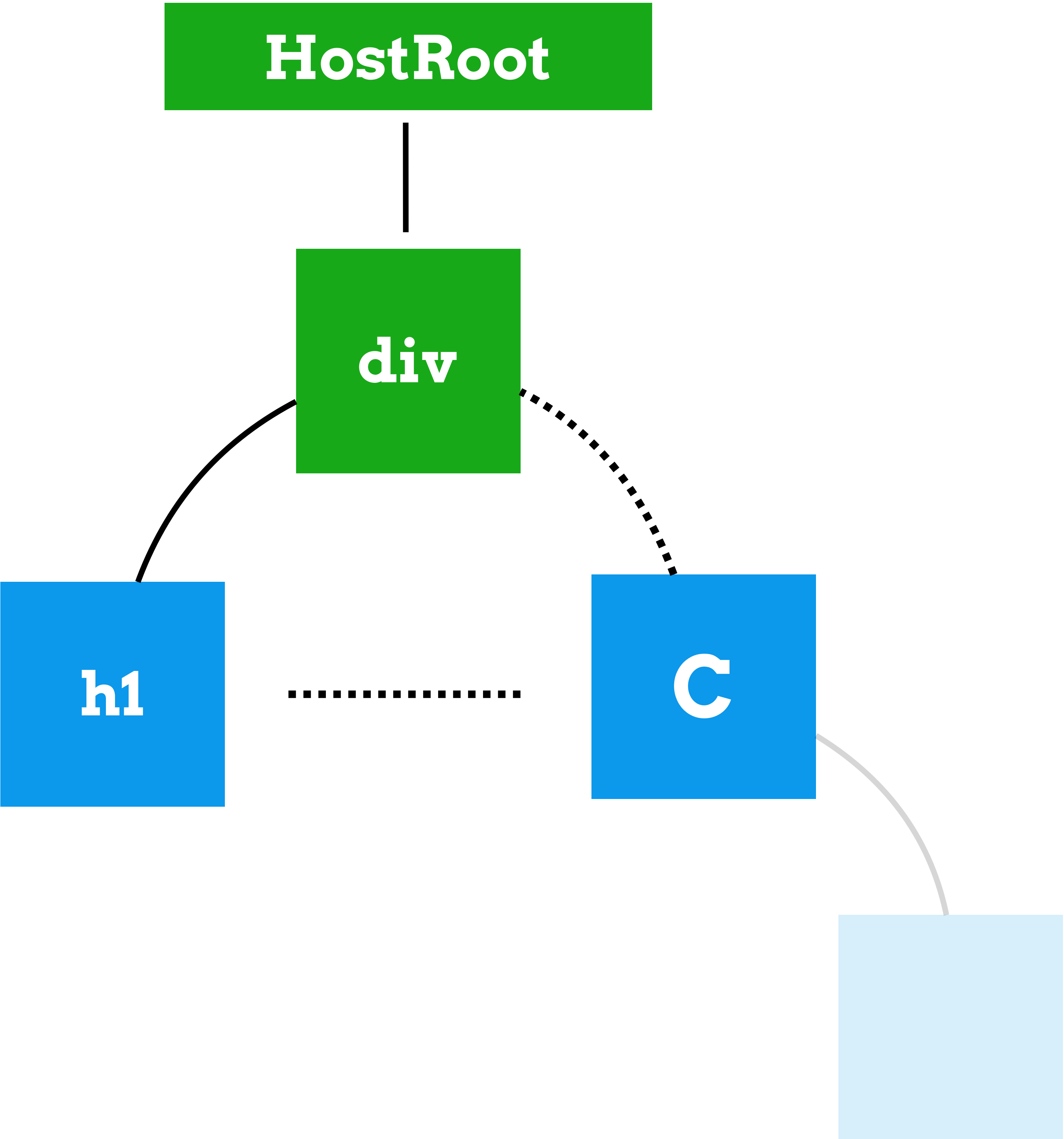
div

h1

C

nextUnitOfWork = performUnitWork

@GURLCODE



nextUnitOfWork !== null

@GURLCODE

HostRoot

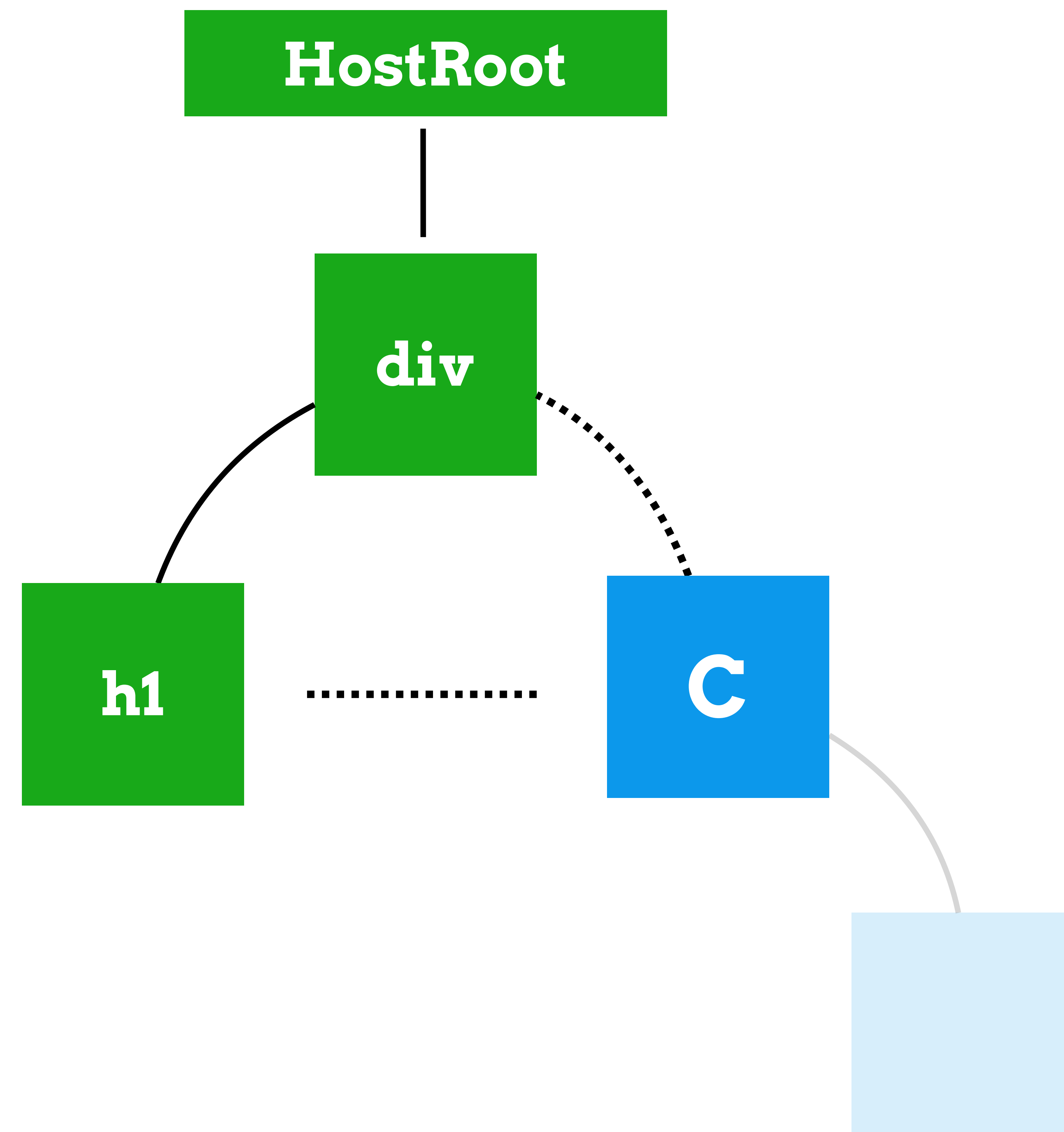
div

h1

C

workLoop

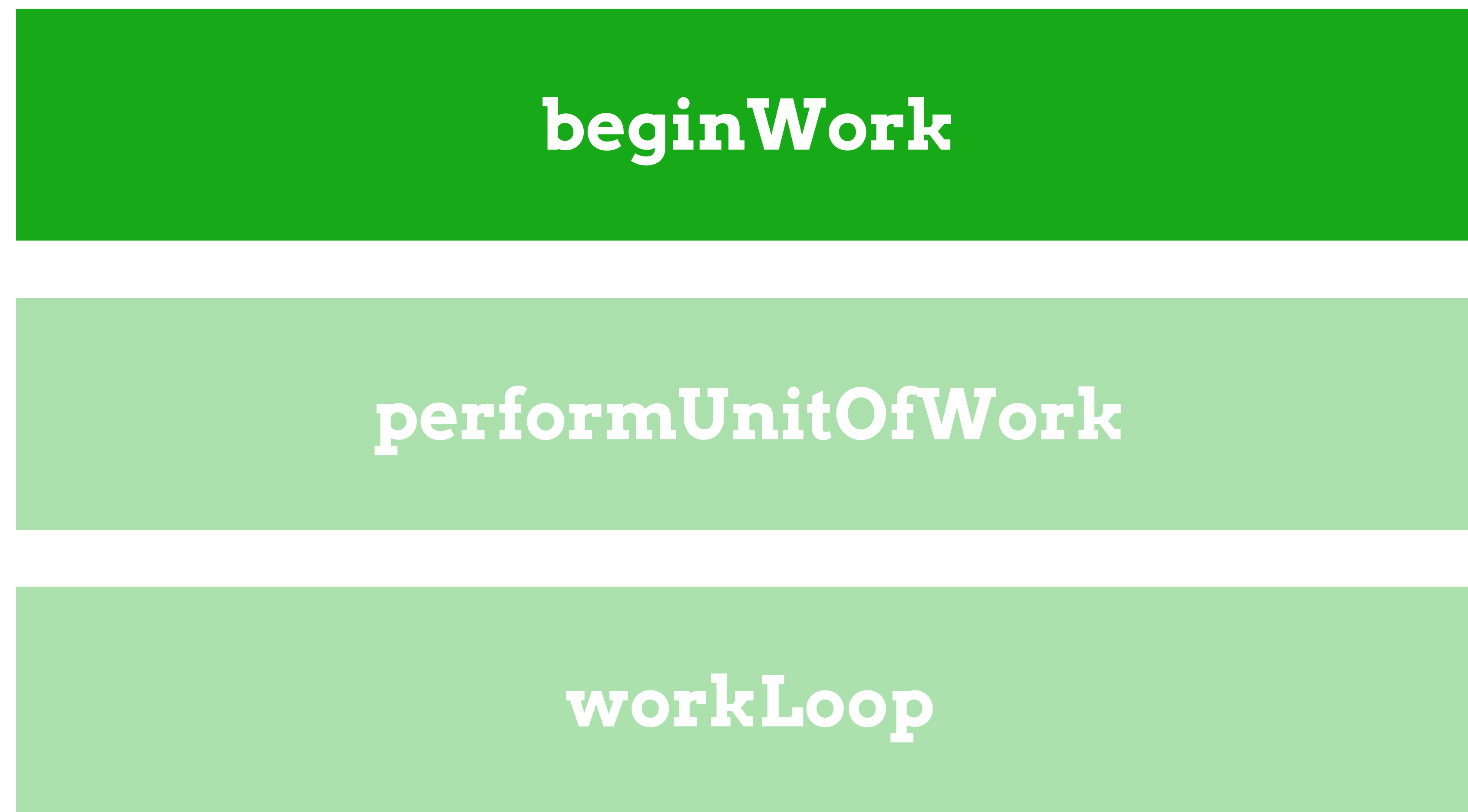
@GURLCODE

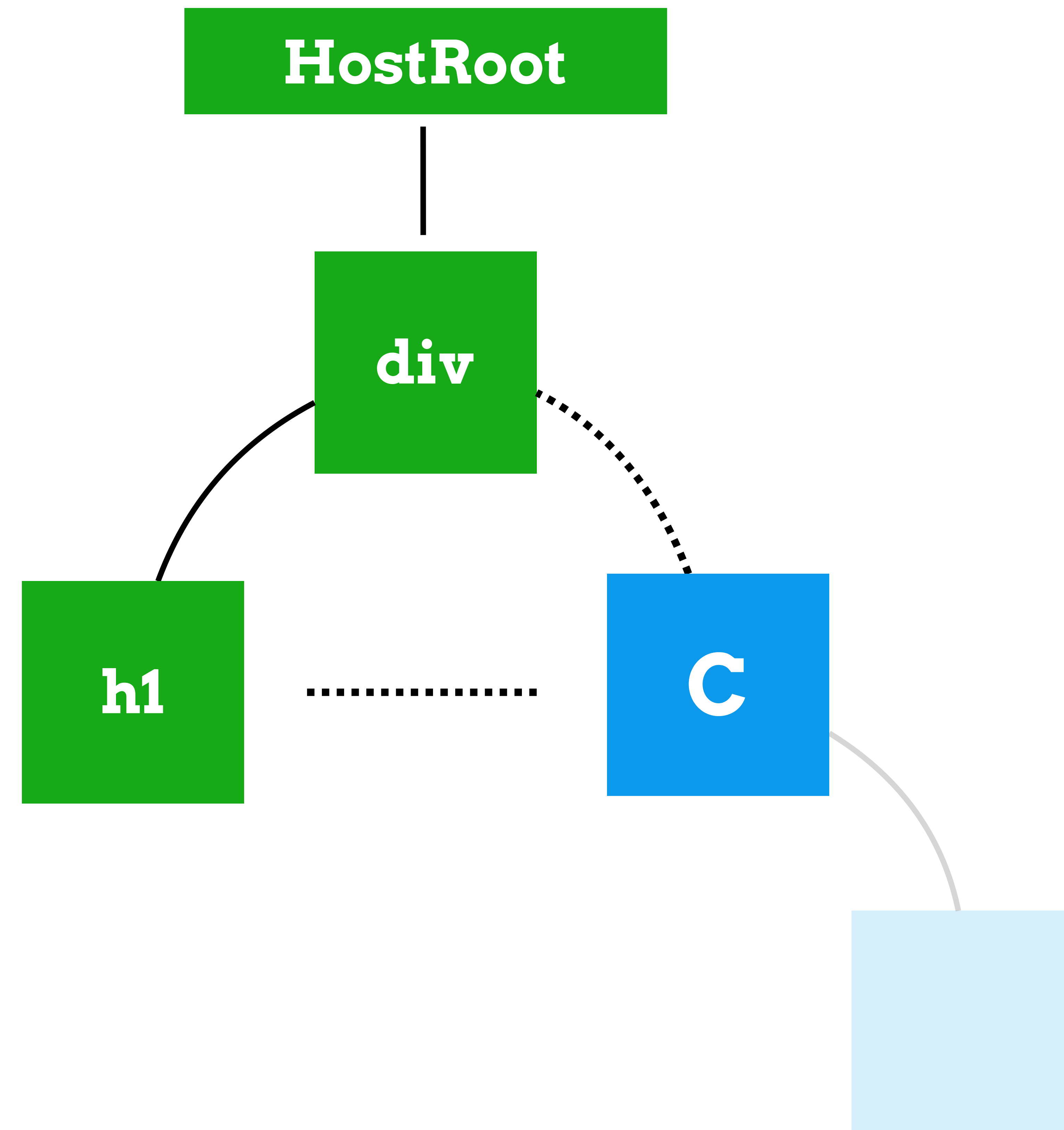


performUnitOfWork

workLoop

@GURLCODE





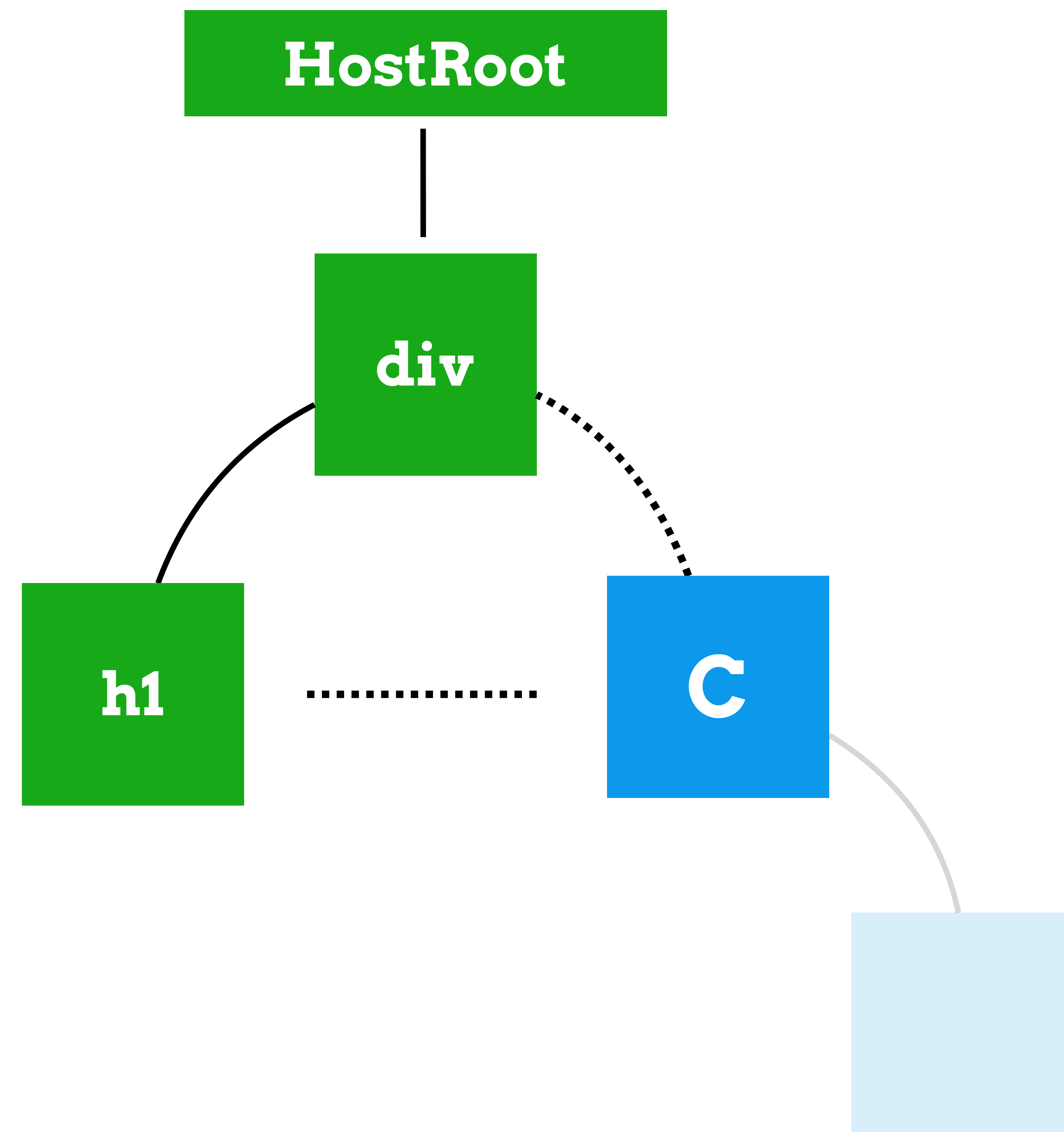
@GURLCODE

updateHostComponent

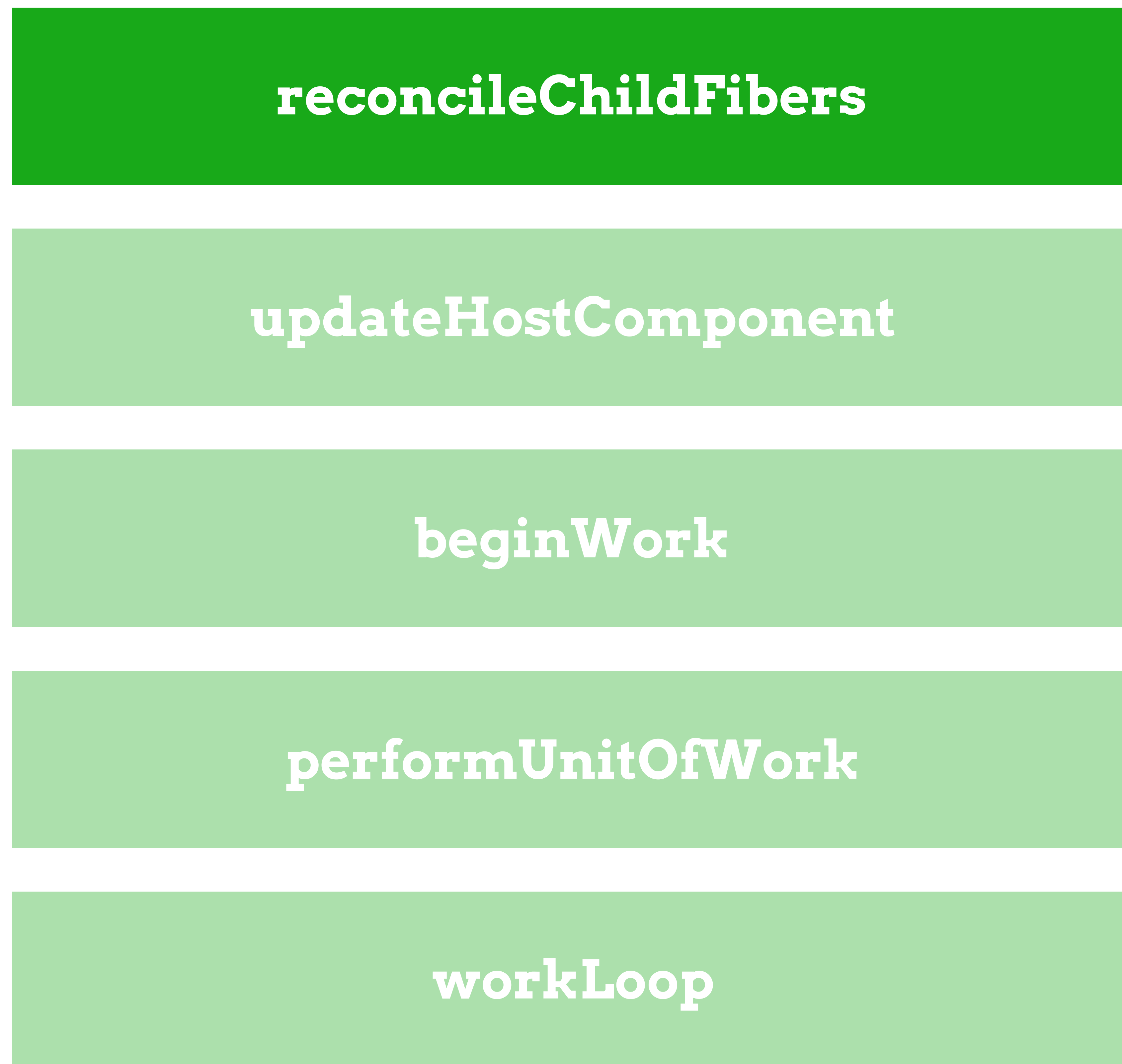
beginWork

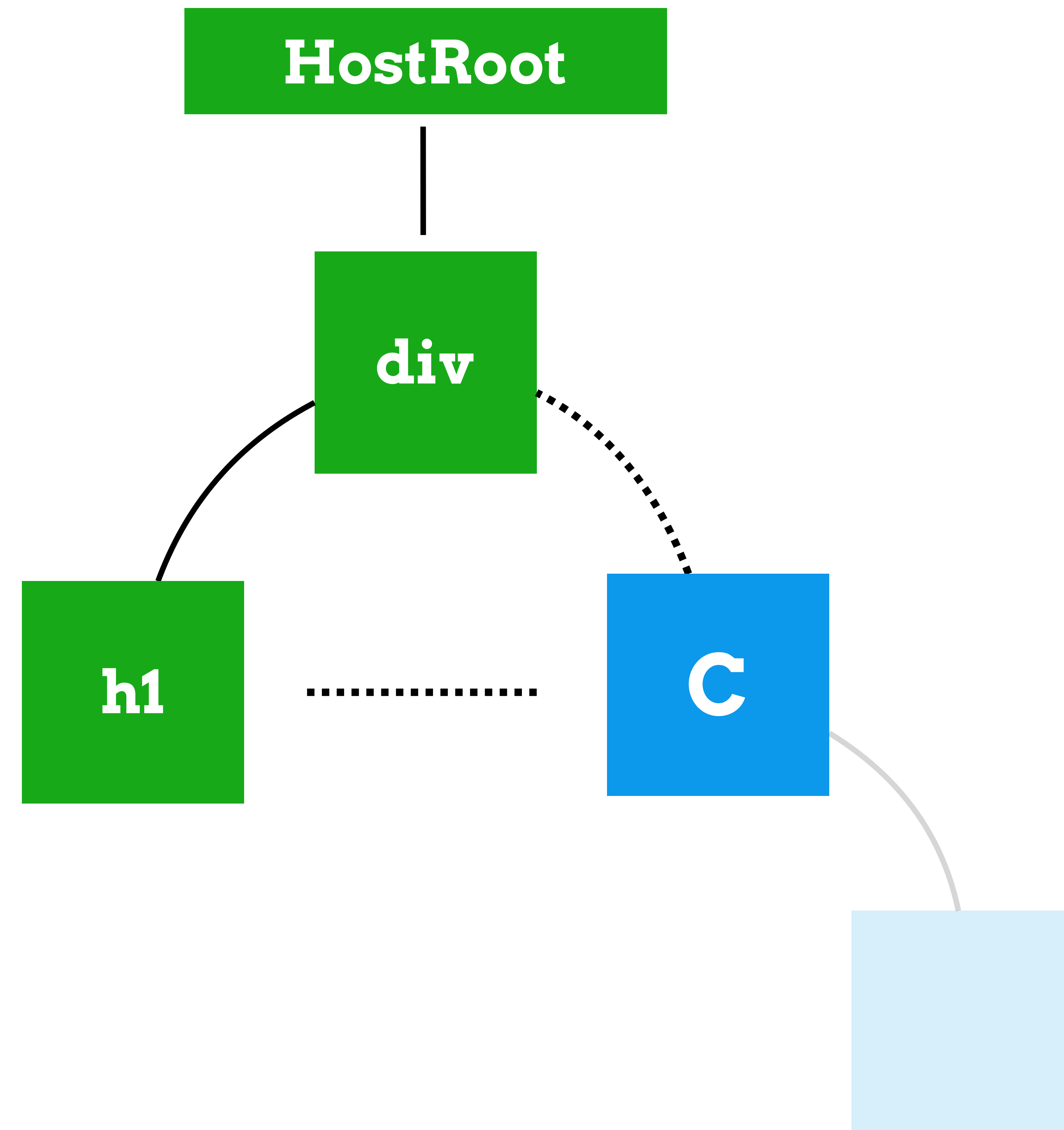
performUnitOfWork

workLoop

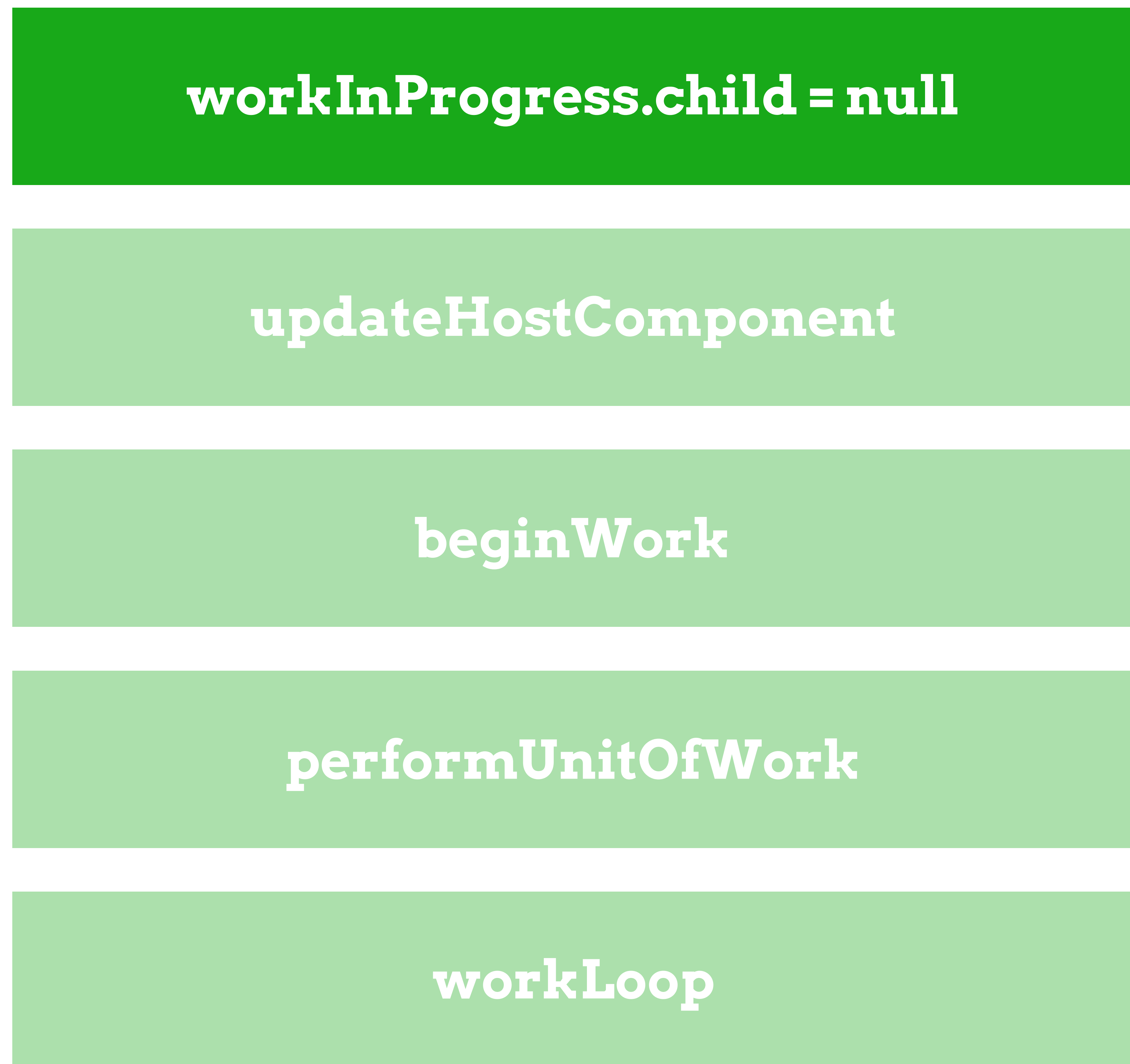


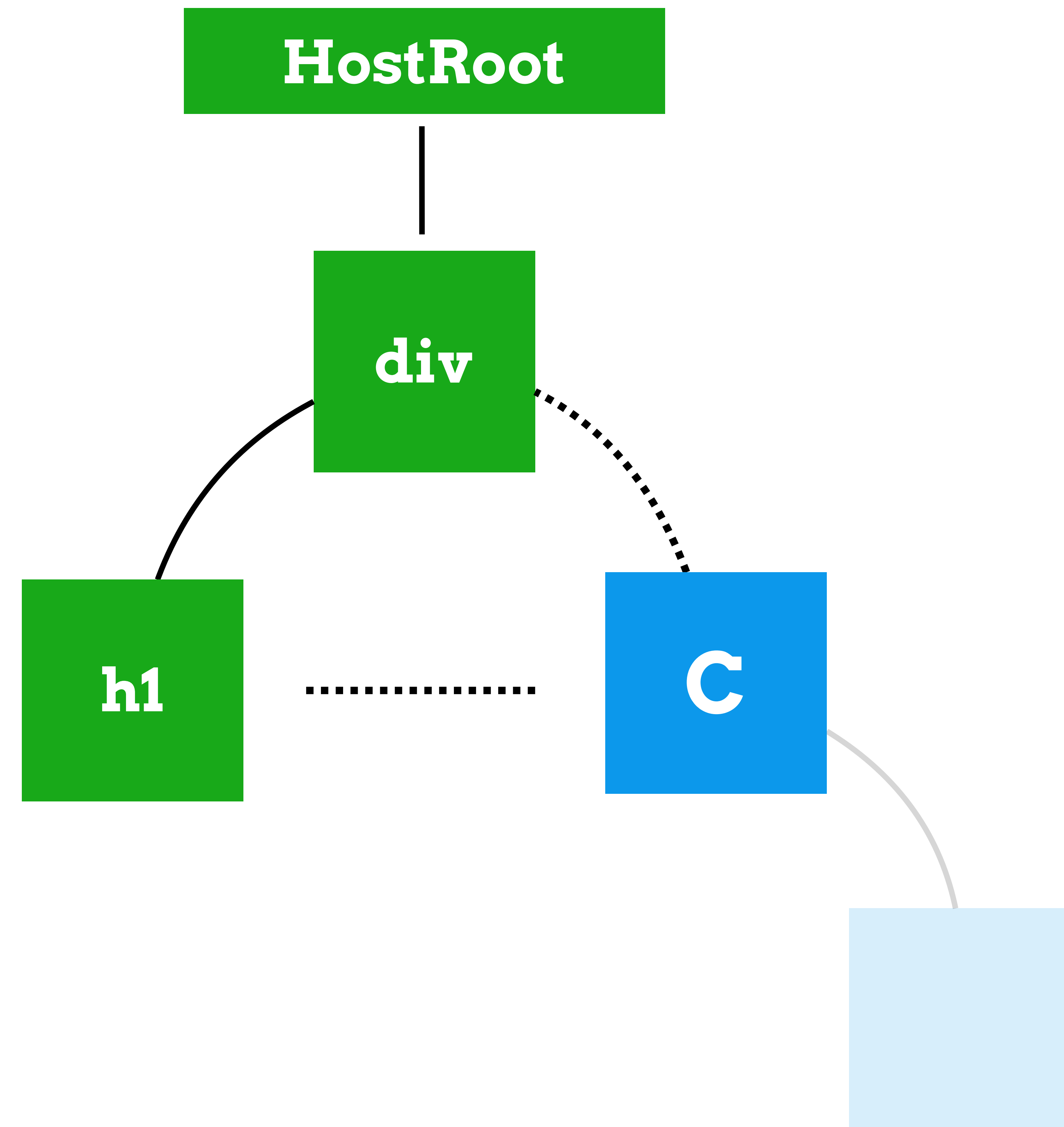
@GURLCODE



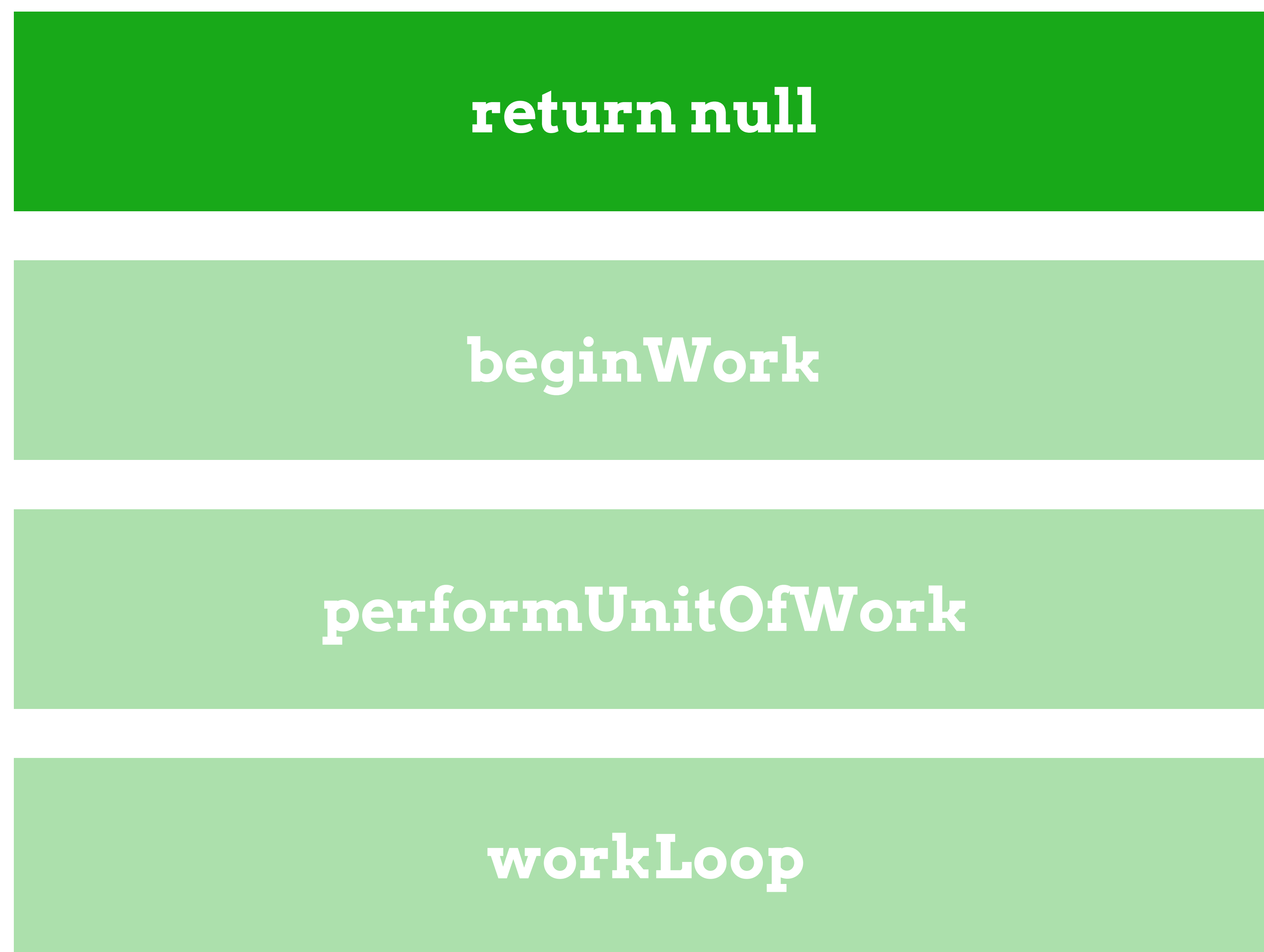


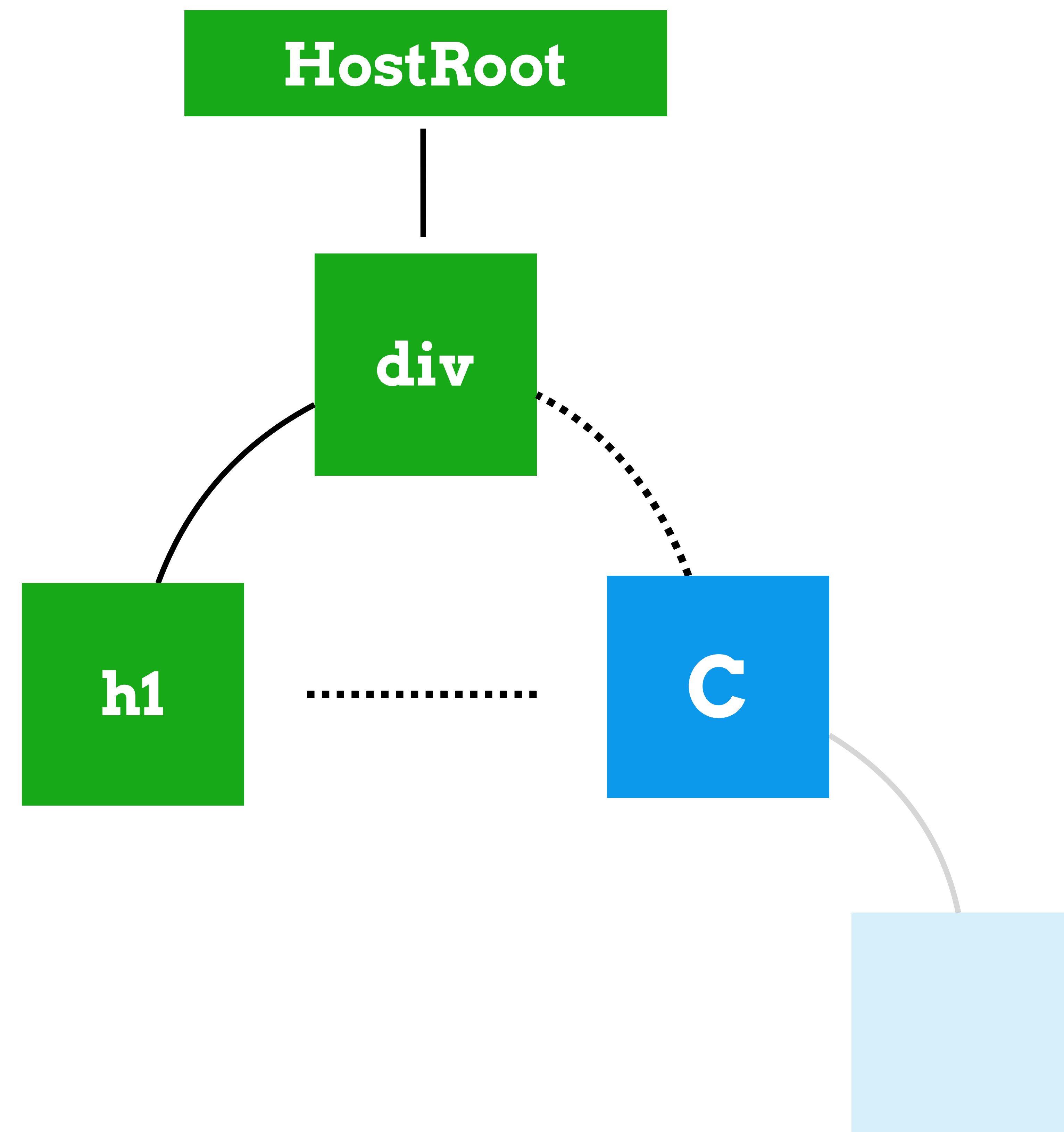
@GURLCODE



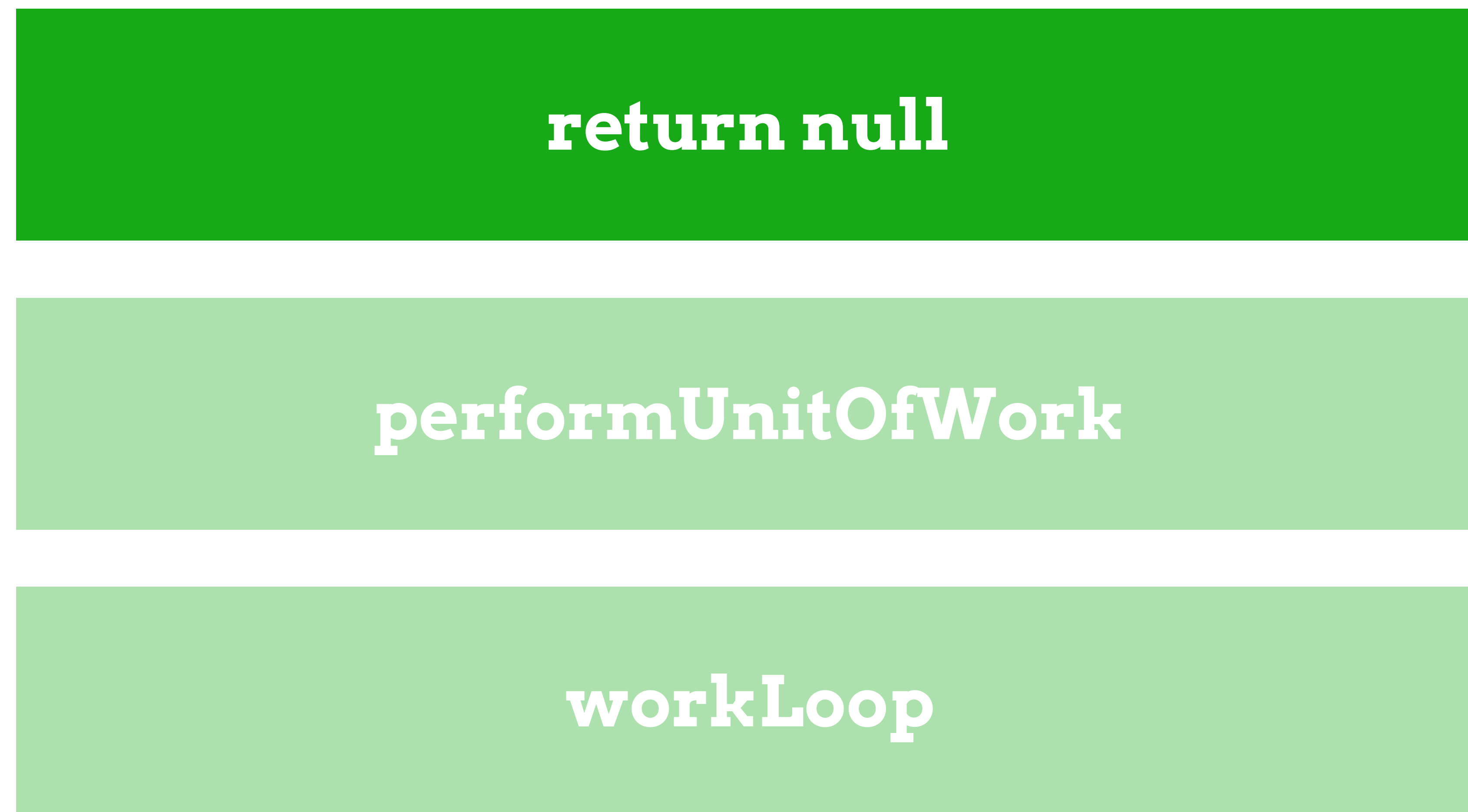


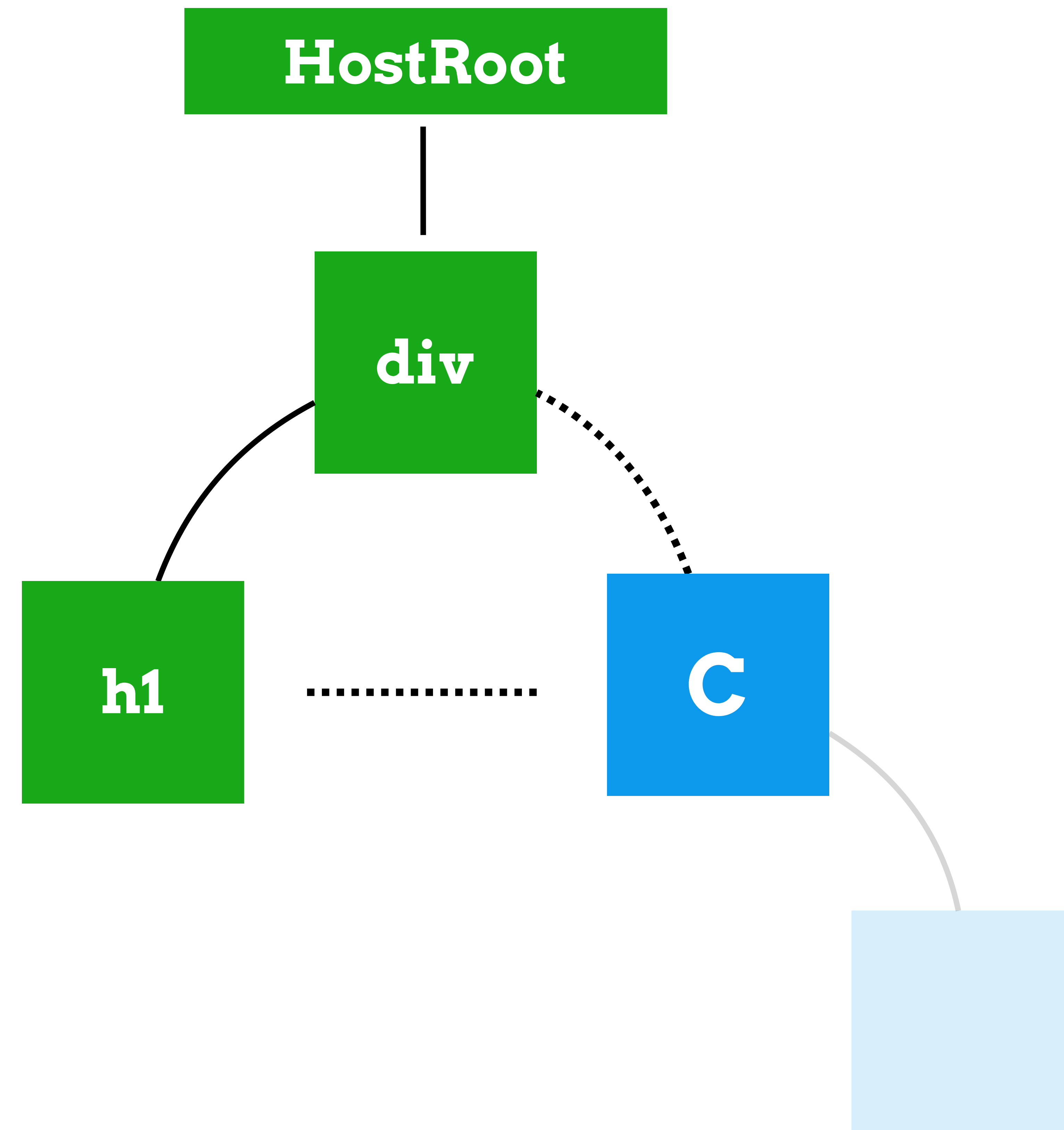
@GURLCODE





@GURLCODE

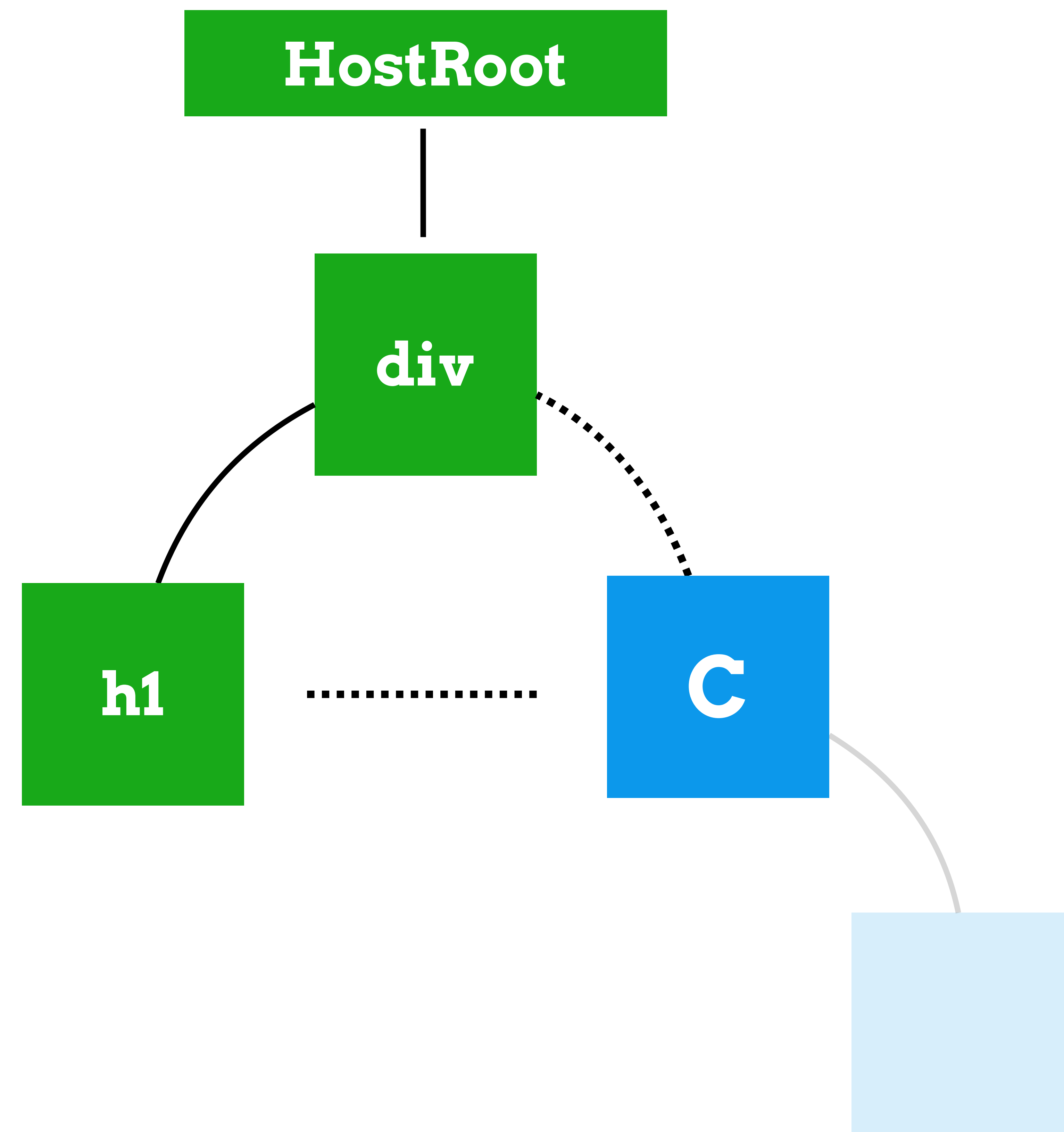




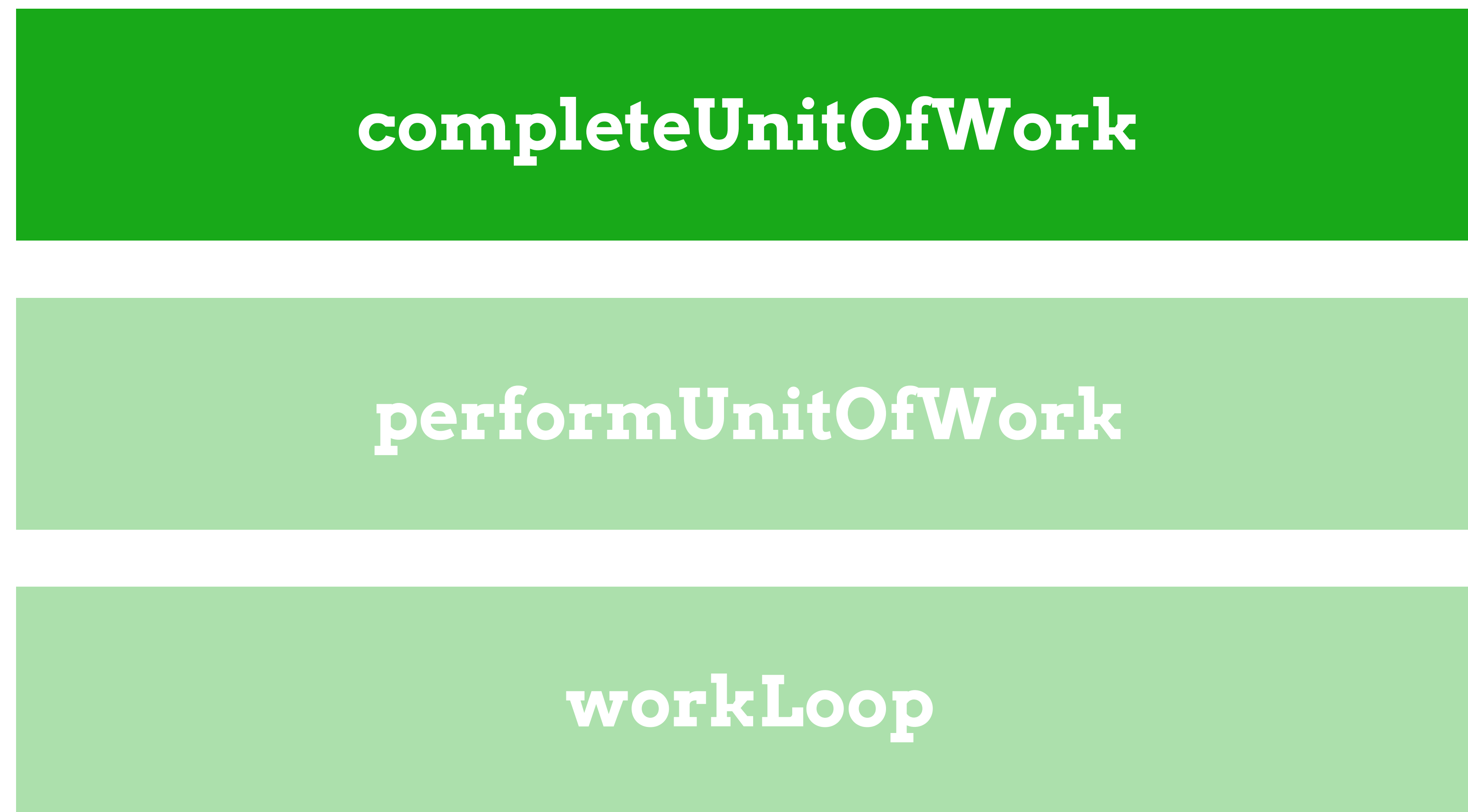
@GURLCODE

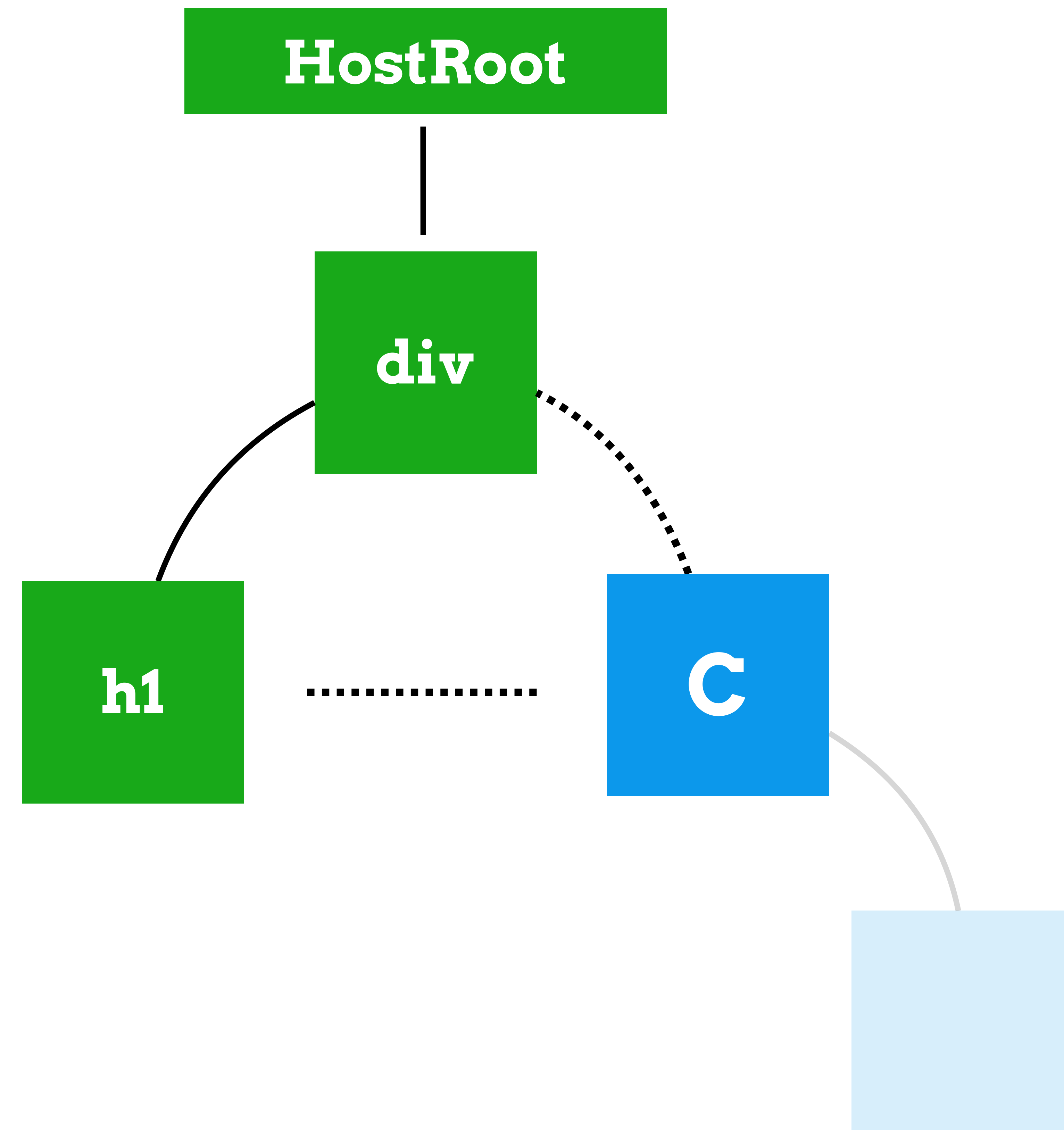
nextUnitOfWork === null

workLoop

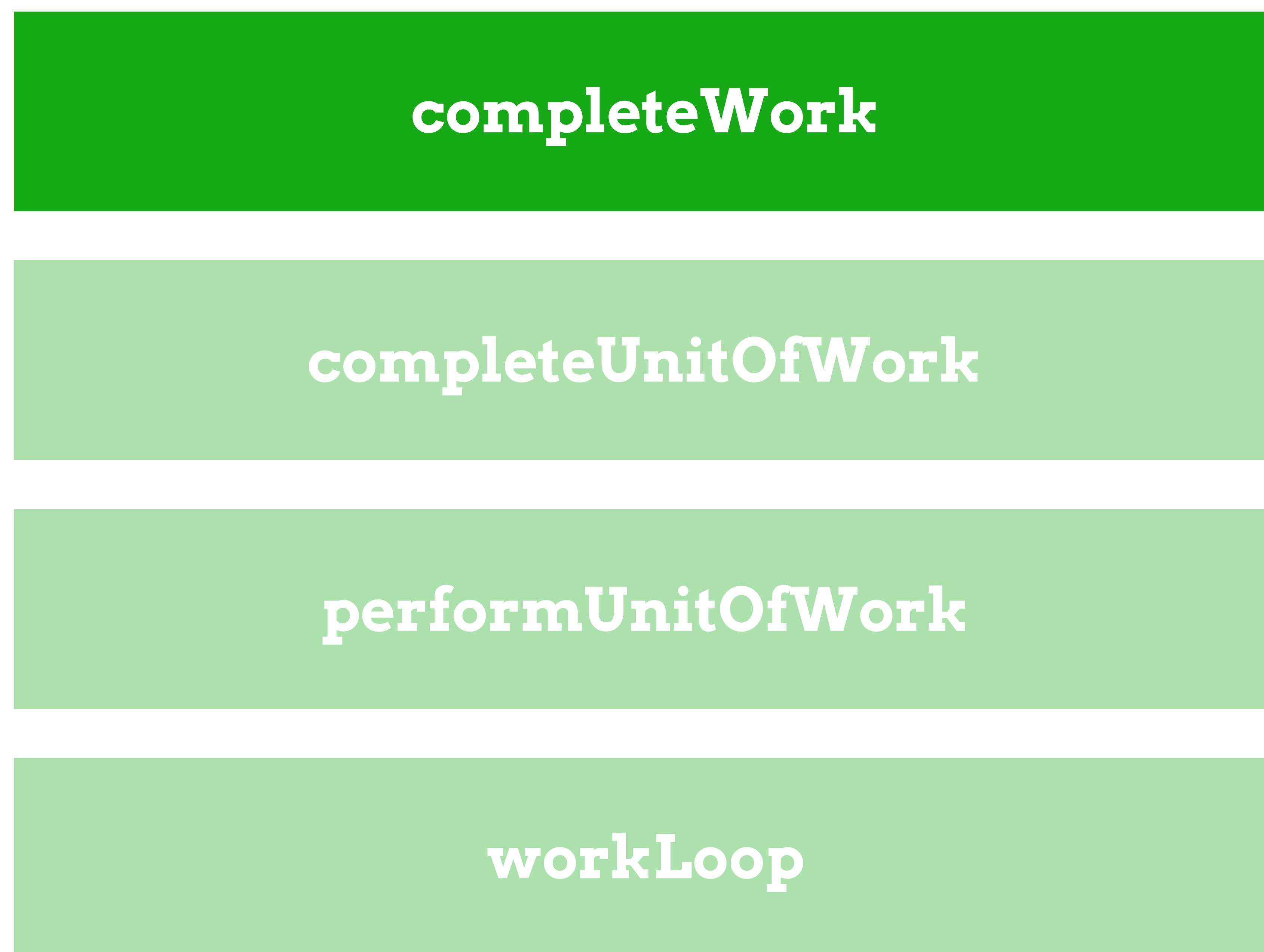


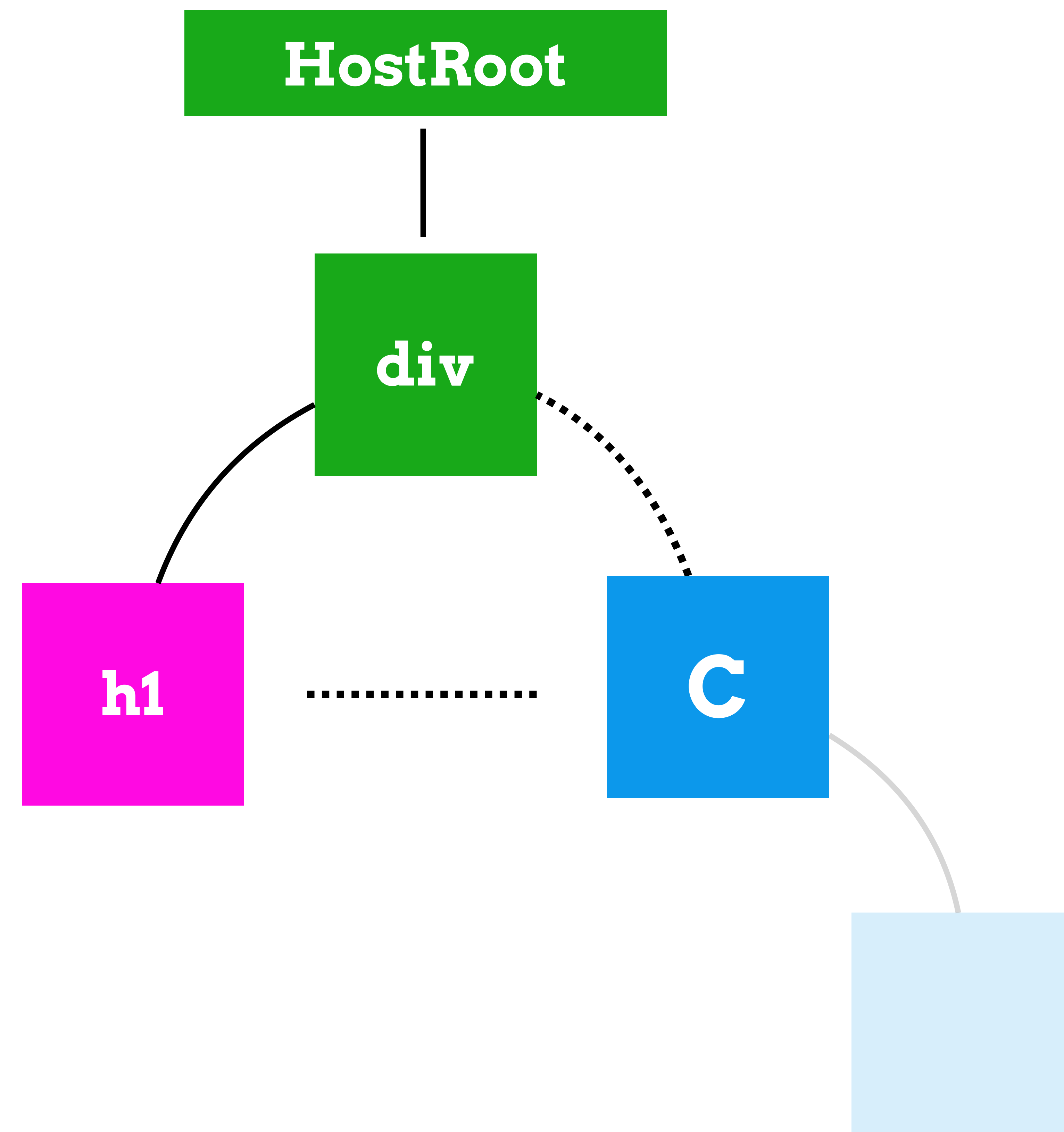
@GURLCODE





@GURLCODE





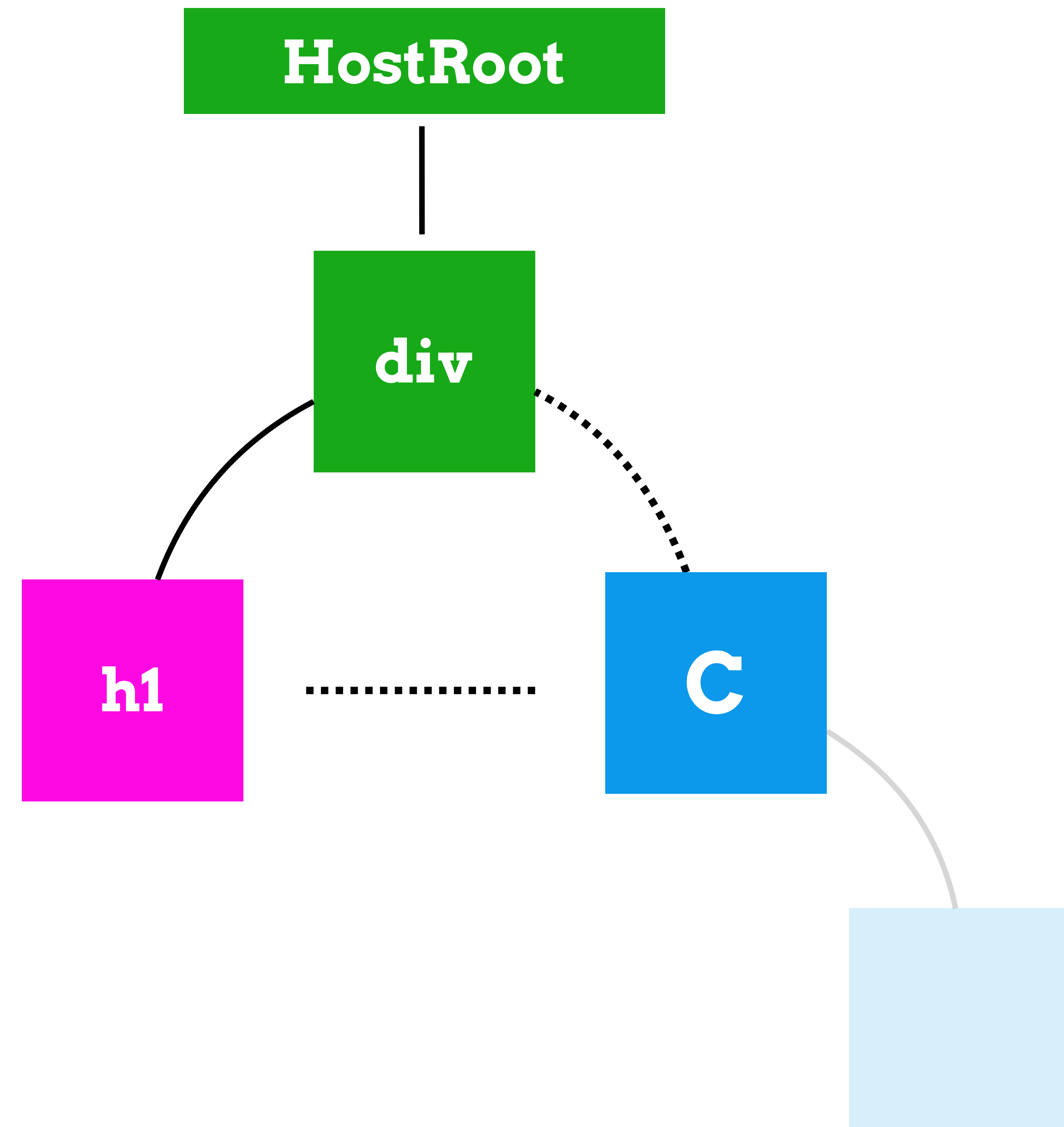
@GURLCODE

return fiber.sibling || fiber.return

completeUnitOfWork

performUnitOfWork

workLoop

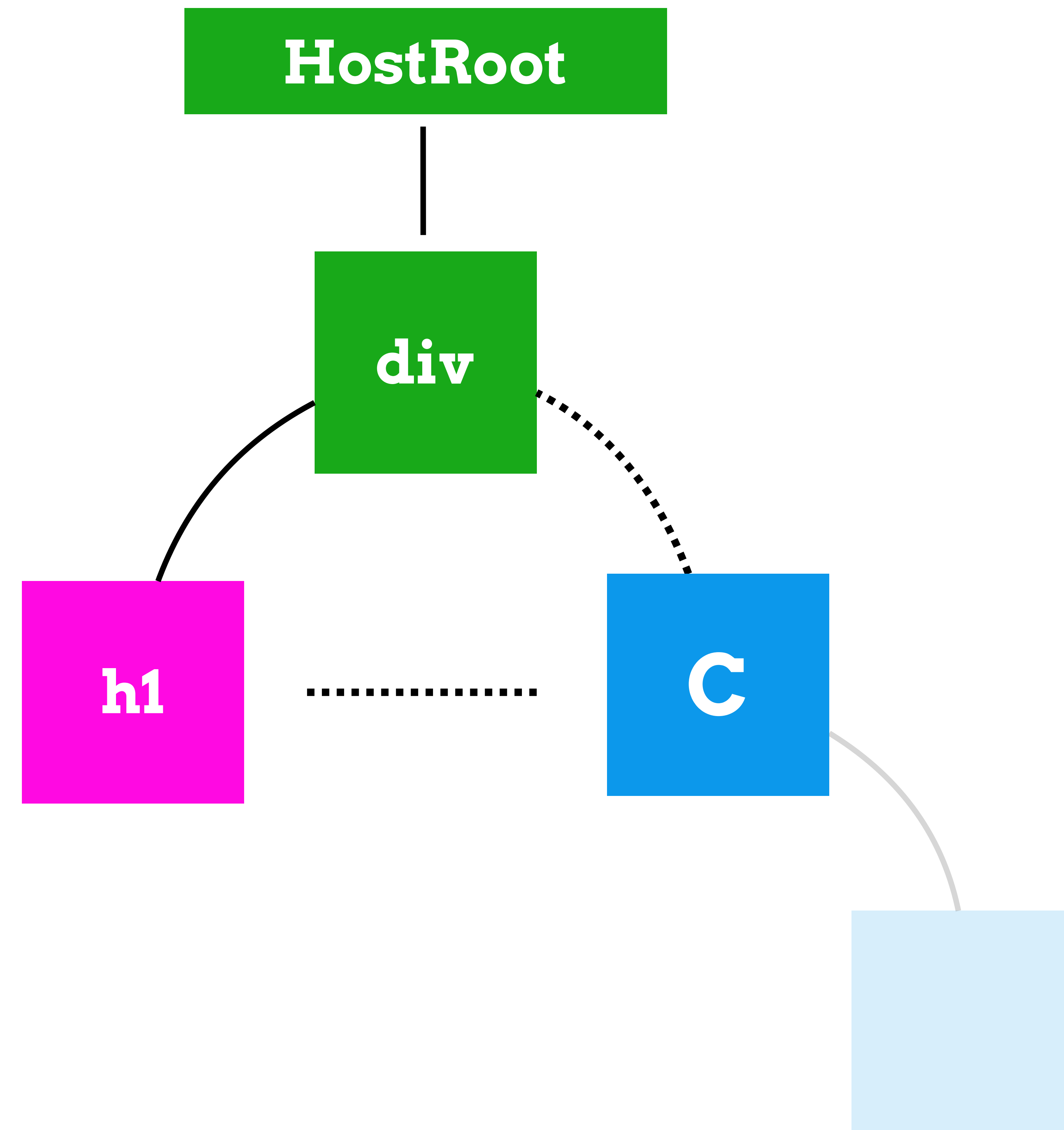


@GURLCODE

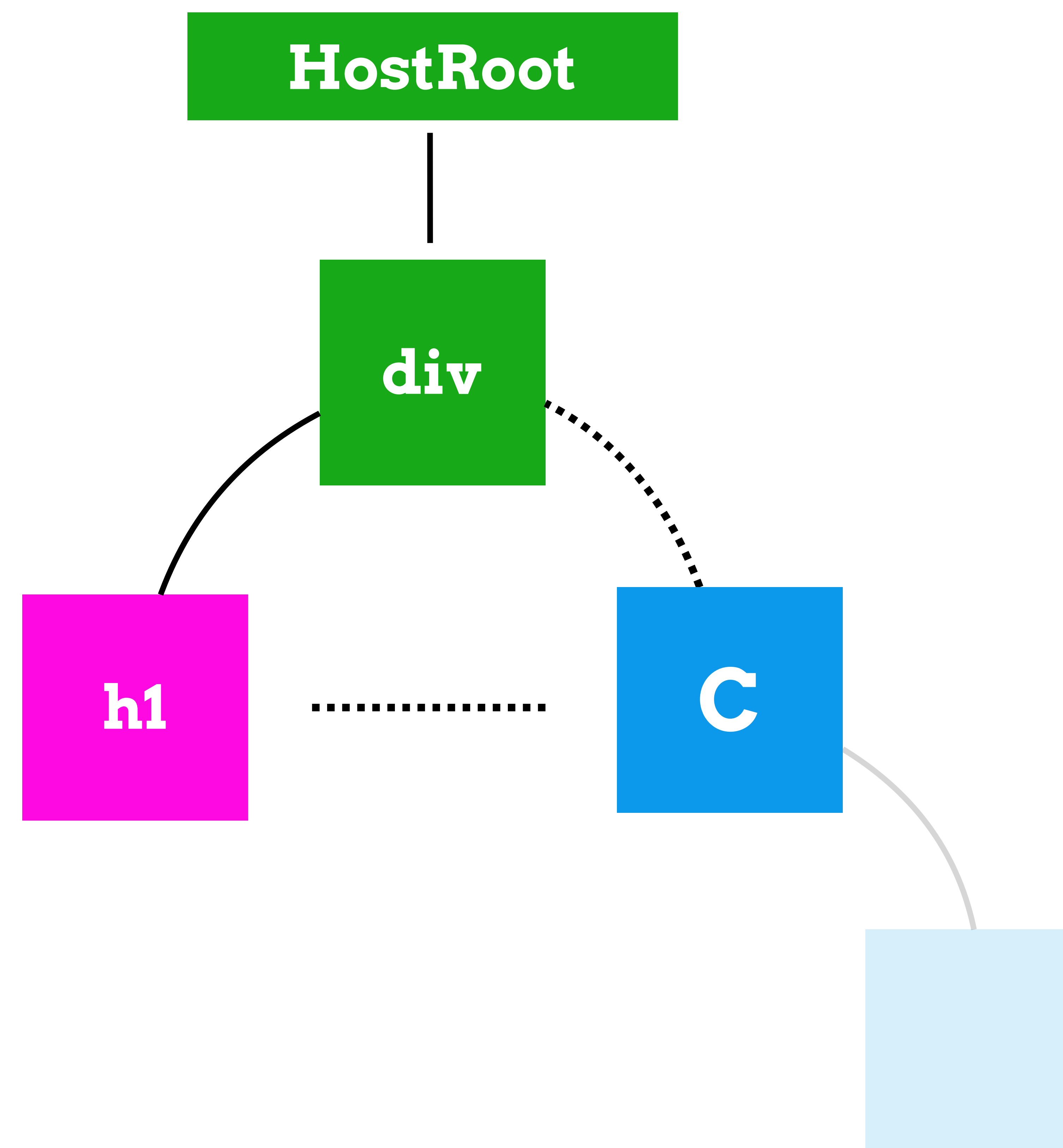
return fiber.sibling

performUnitOfWork

workLoop



@GURLCODE



return fiber.sibling

workLoop

@GURLCODE

HostRoot

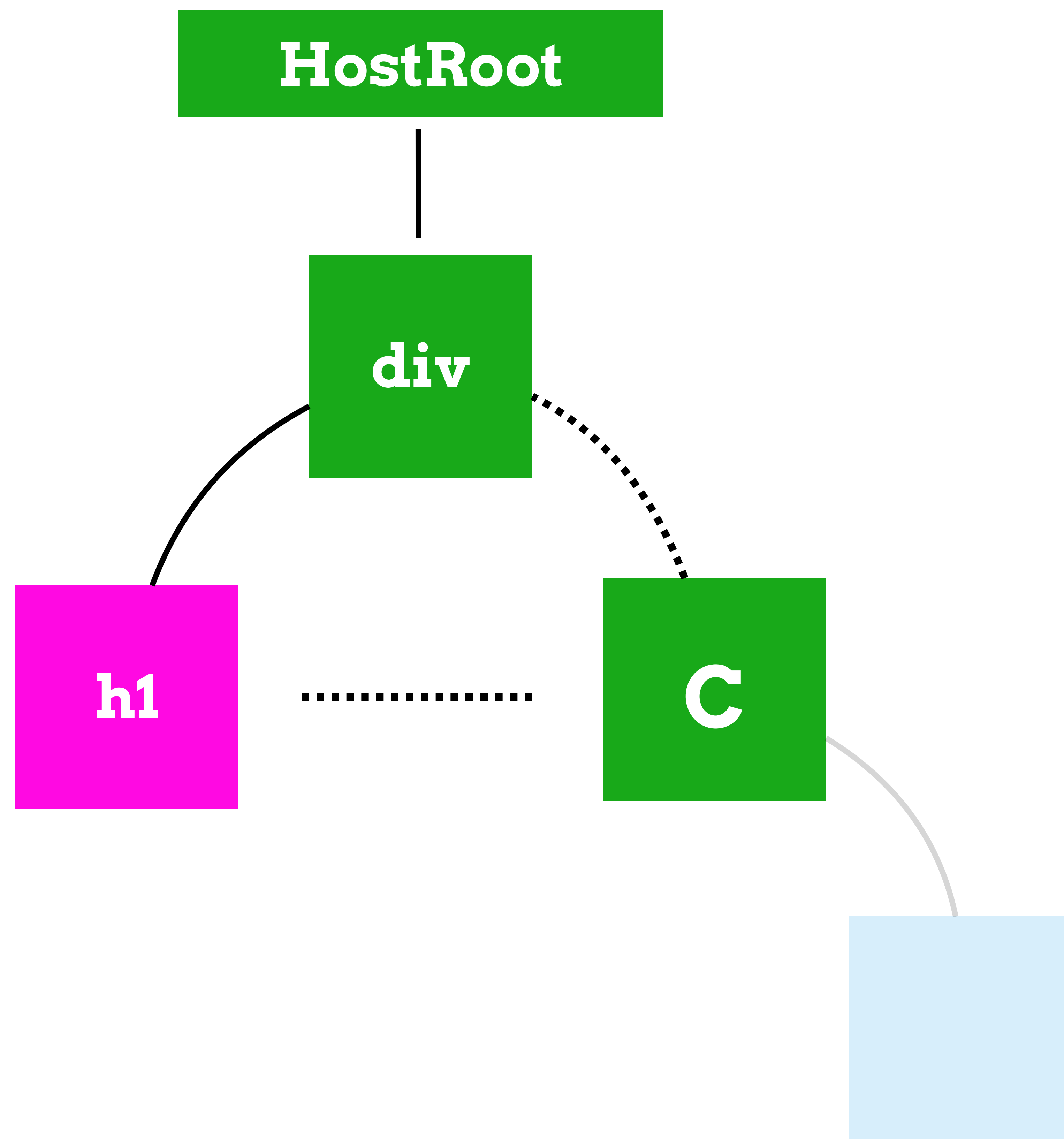
div

h1

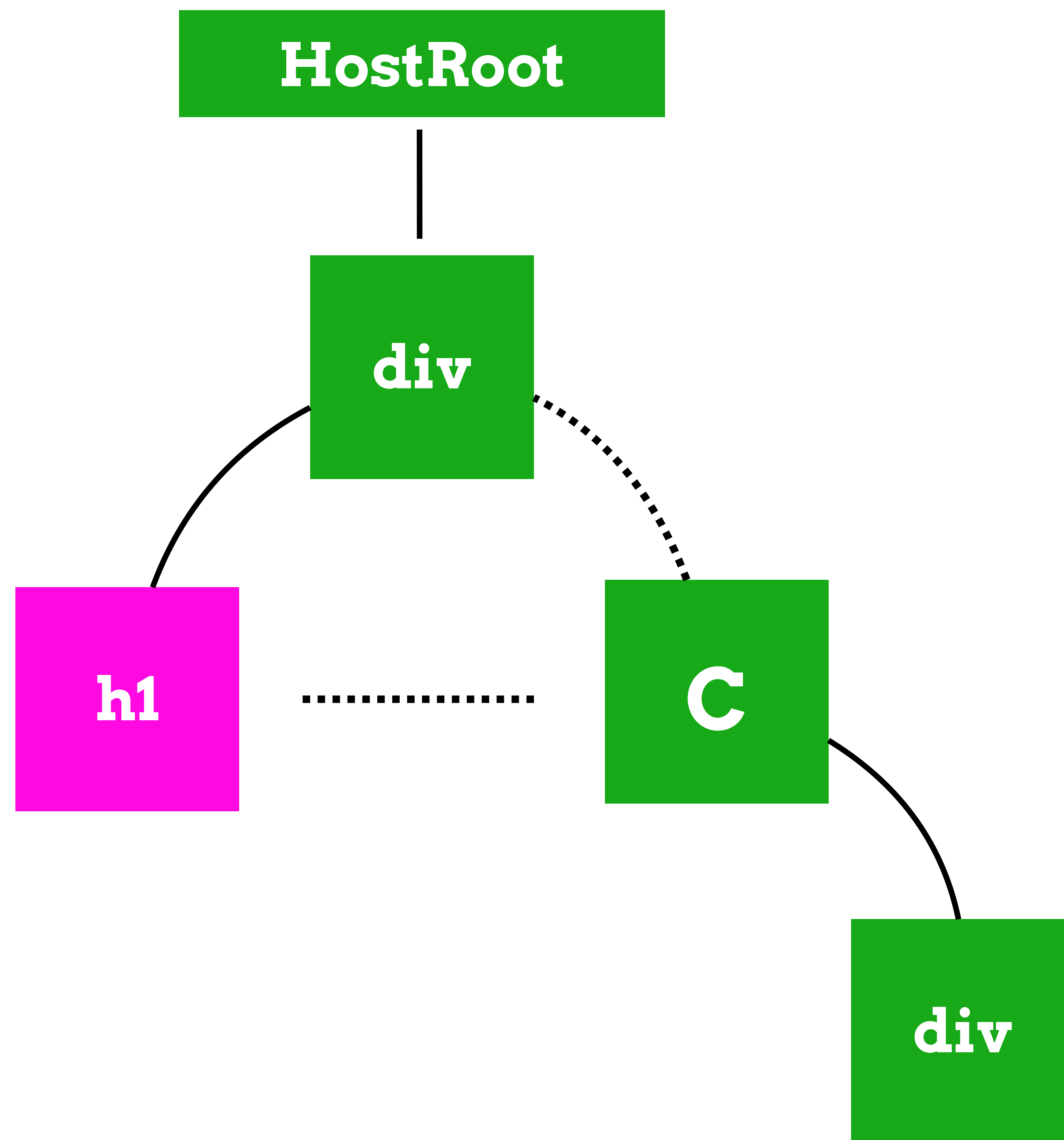
C

nextUnitOfWork = performUnitOfWork

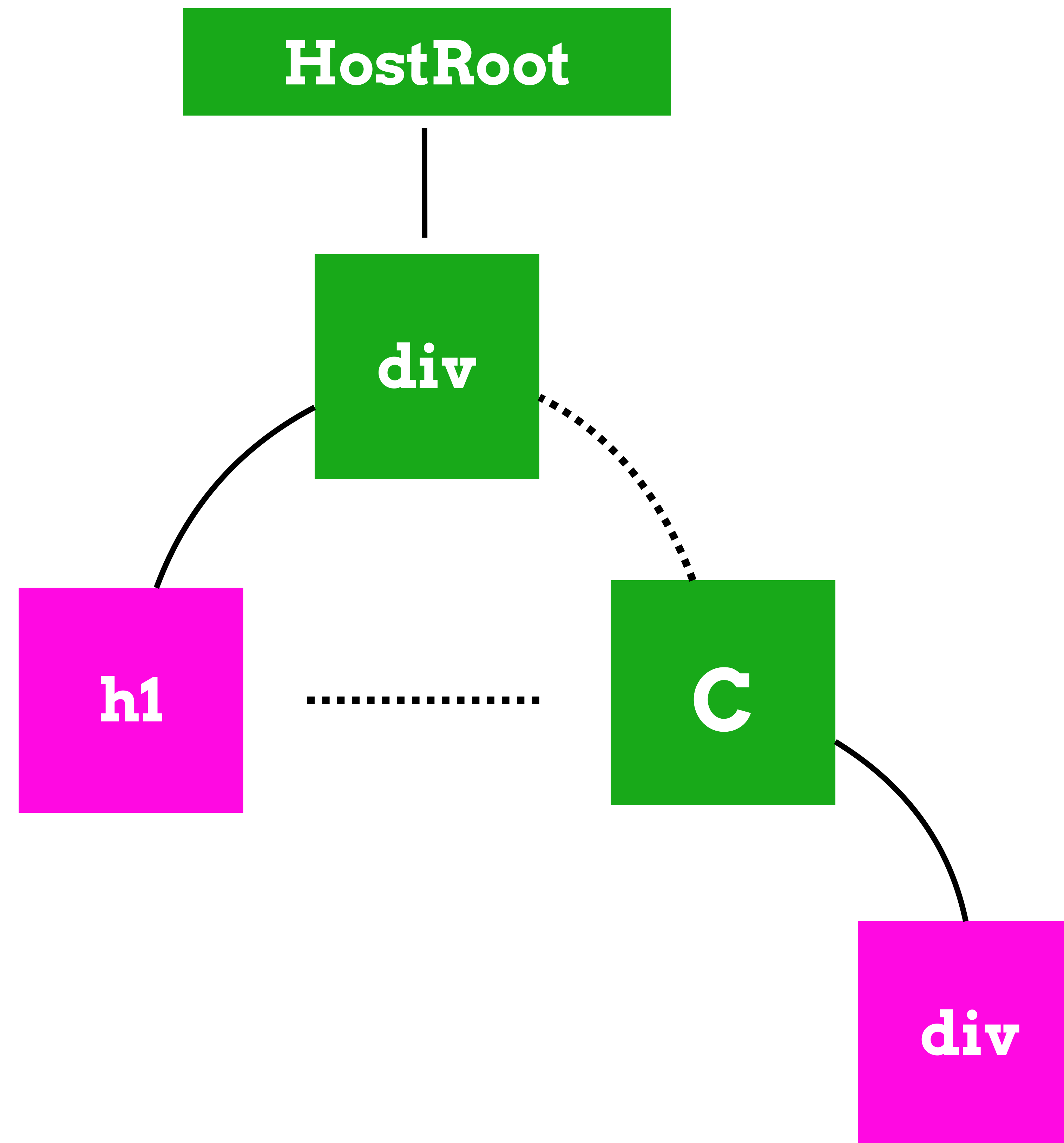
@GURLCODE



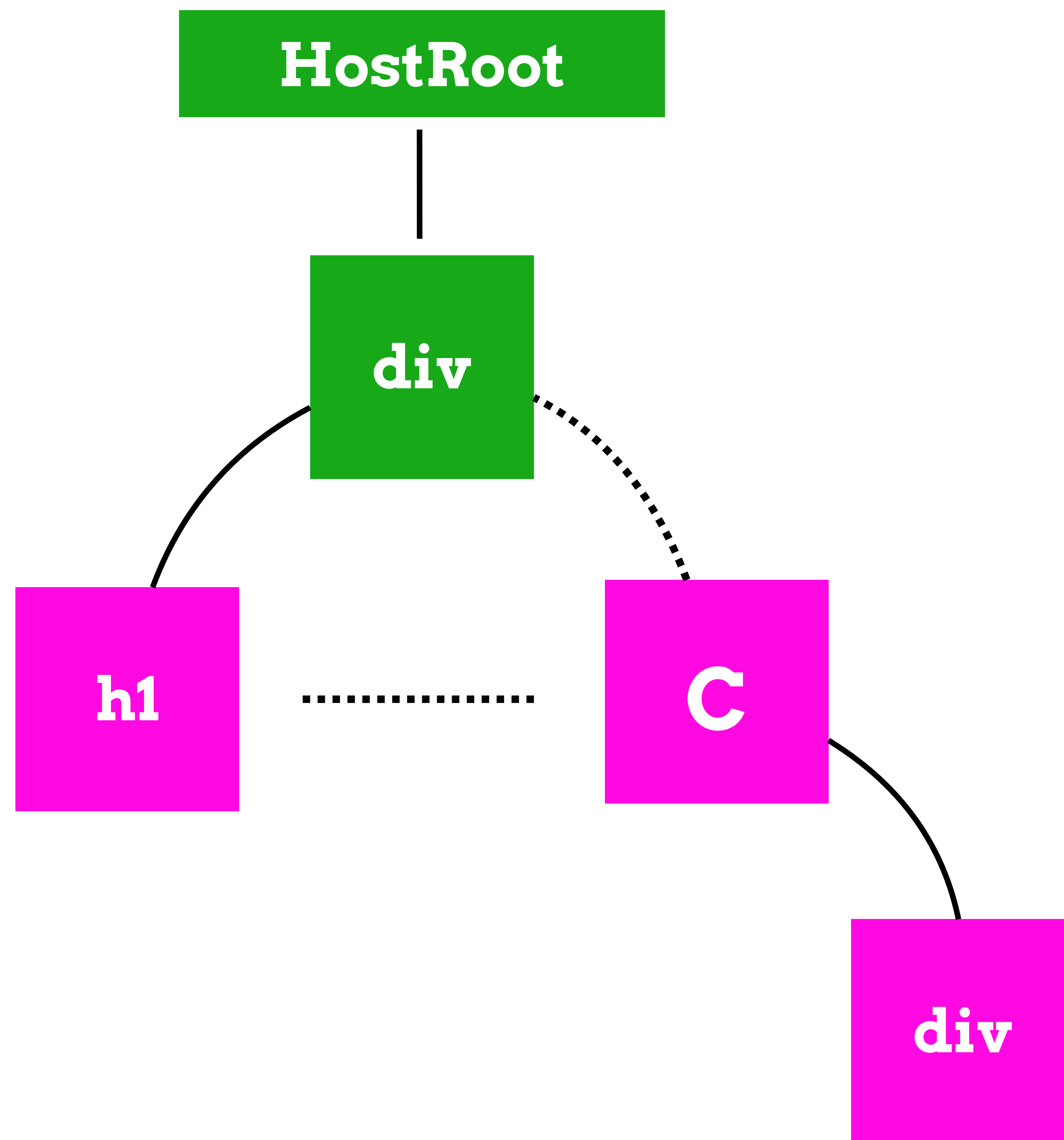
@GURLCODE



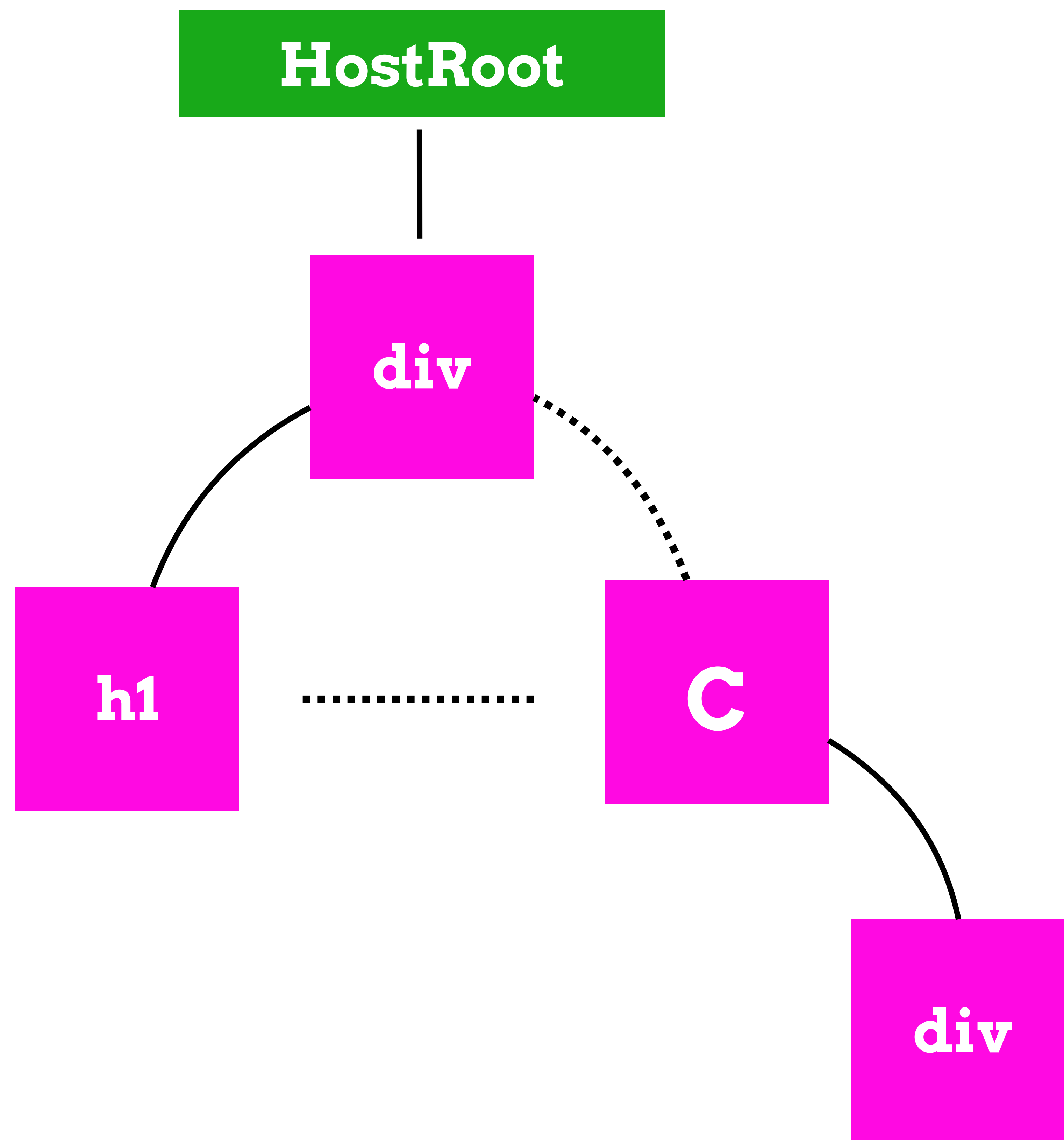
@GURLCODE



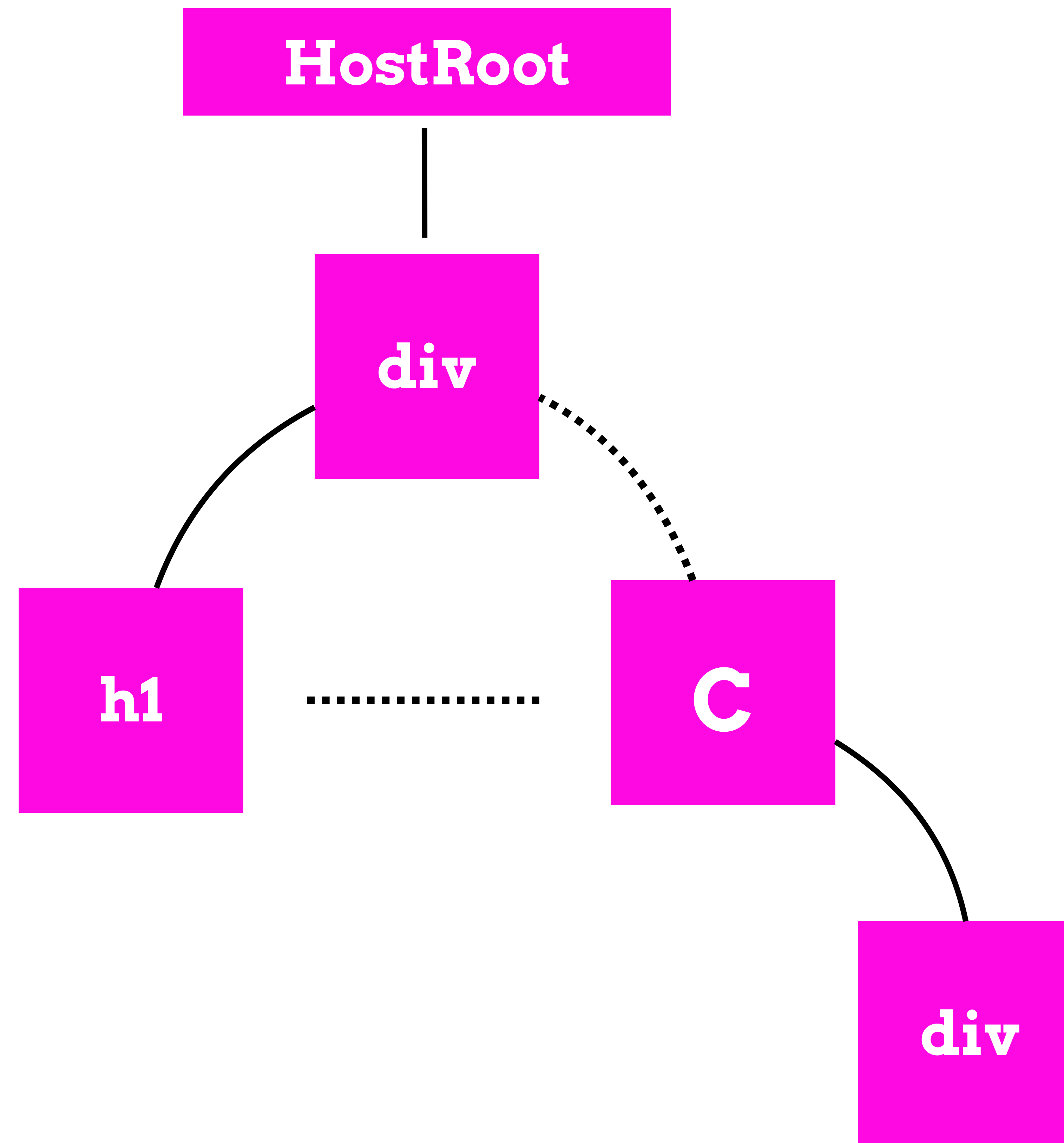
@GURLCODE



@GURLCODE



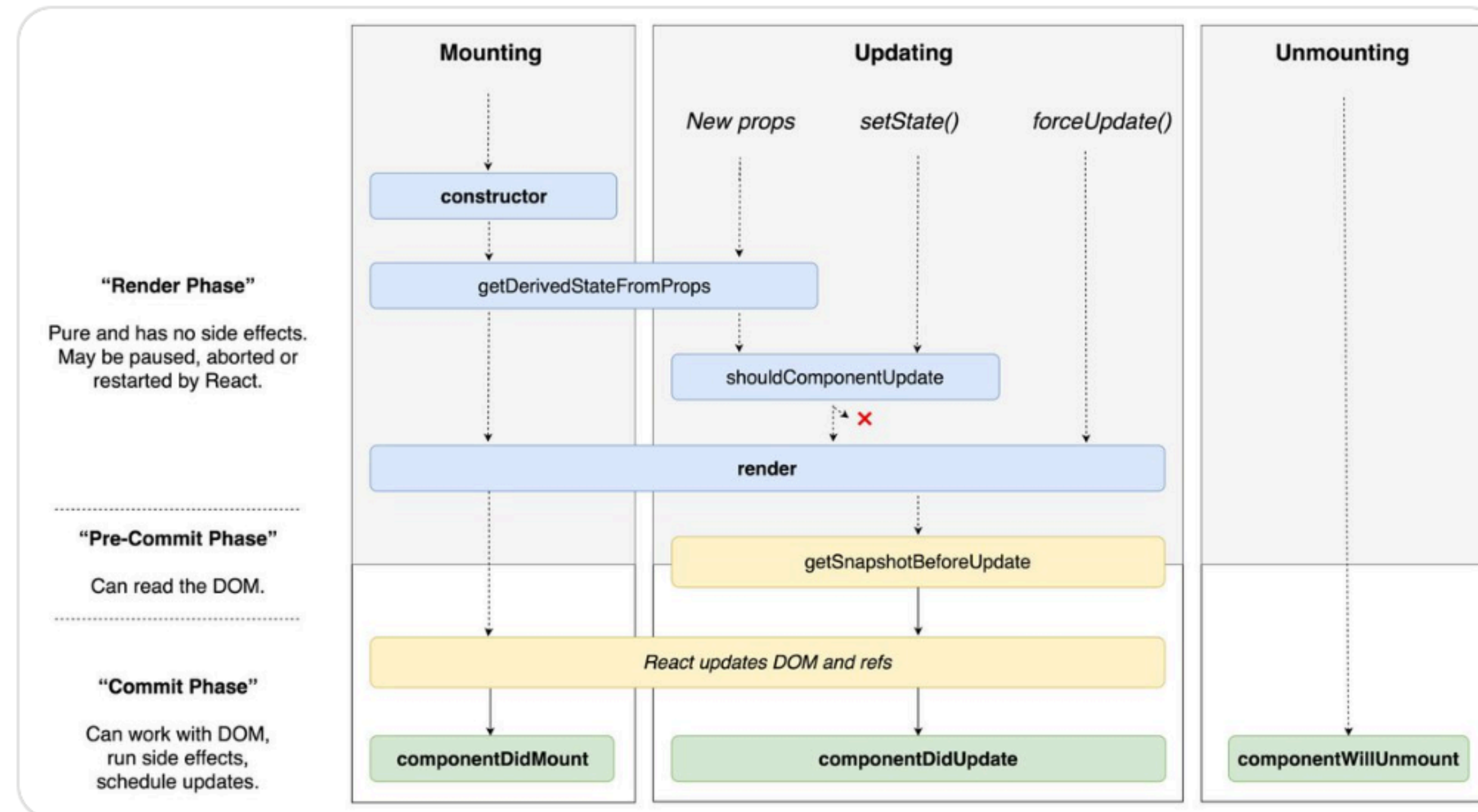
@GURLCODE



@GURLCODE

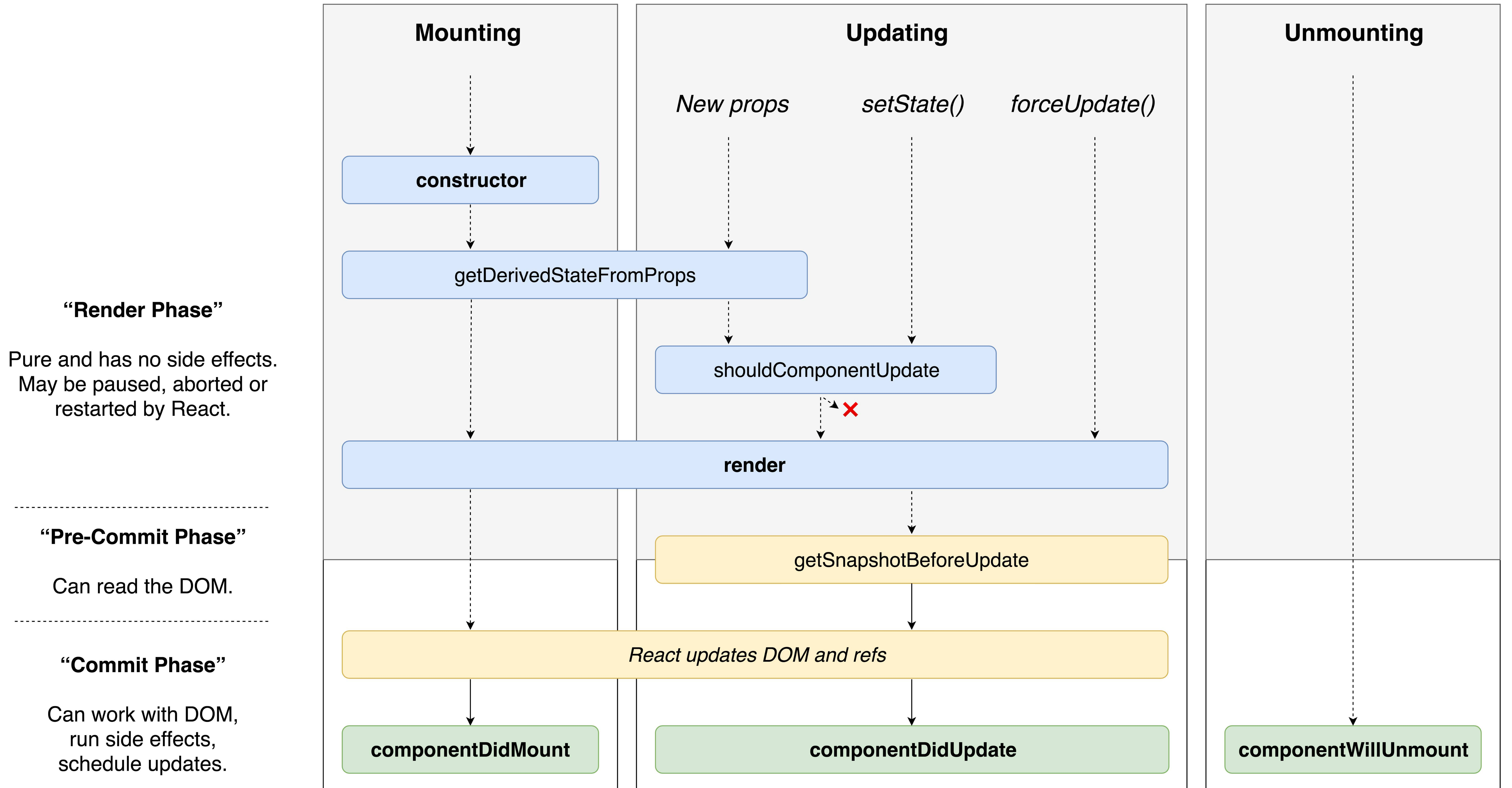
Dan Abramov
@dan_abramov

I just made this diagram of modern React lifecycle methods. Hope you'll find it helpful!



9:56 PM - 4 Apr 2018

@GURLCODE



Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
“Render Phase”
Pure and has no side effects. May be paused, aborted or restarted by React.
“Pre-Commit Phase”
Can read the DOM.
“Commit Phase”
Can work with DOM, run side effects, schedule updates.

Andrew Clark
@acdlite

Crucially, React guarantees that the final state is always deterministic based on the order of updates. Intermediate states may vary according to resources, but eventually everything resolves to a predictable state.

7:27 PM - 26 Mar 2018

@GURLCODE

sophiebits commented 4 days ago

Collaborator

We yield to the host environment periodically – every 16ms or so – to allow the browse to process incoming events including user input. `frameDeadline` is the timestamp that we plan to yield at (originally set to something like `now() + 16ms`), so shouldYieldToHost returns true once that time has passed. Then we use some combination of requestIdleCallback and requestAnimationFrame so we can process the next piece of work soon.

In the ideal case, we can finish all of the rendering in these small 16ms slices. However, if there are many other things happening at the same time, React work may "starve" and not be able to fully render in the small slices. So we have a second check: every pending render or state update has an "expiration time" (usually somewhere from 100ms to 5000ms) – if that time passes without the render finishing, we switch to a synchronous mode until that update can be finished. This is not ideal but it ensures that all updates get processed without waiting too long.

We set a timer in the browser (e.g., with setTimeout) for that same expiration time. If that timer fires, we know we need to perform the work synchronously. If this happens, `currentDidTimeout` is set to true, so we won't yield.

In the future, we plan to use a new `isInputPending` browser API (https://github.com/WICG/is-input-pending) so we can continue processing work and only yield when there is new user input, instead of always yielding every 16ms.

4

@GURLCODE

Andrew Clark
@acdlite

To avoid starvation, low priority updates will eventually expire, effectively upgrading them to high priority. This prevents the case where a low priority can never finish because it keeps being interrupted.

7:27 PM - 26 Mar 2018

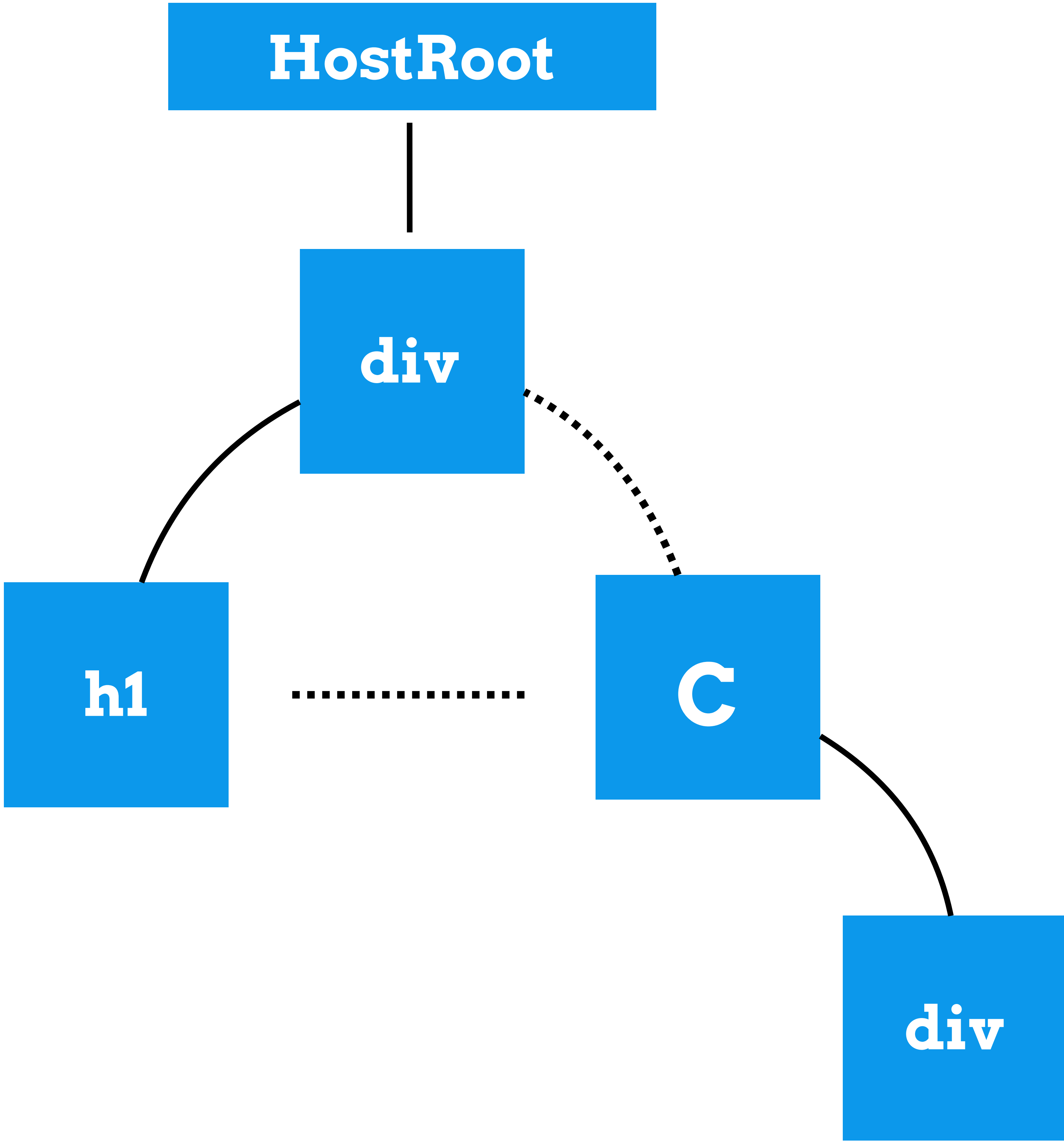
@GURLCODE

ColorBox

@GURLCODE

Component.prototype.SetState

@GURLCODE

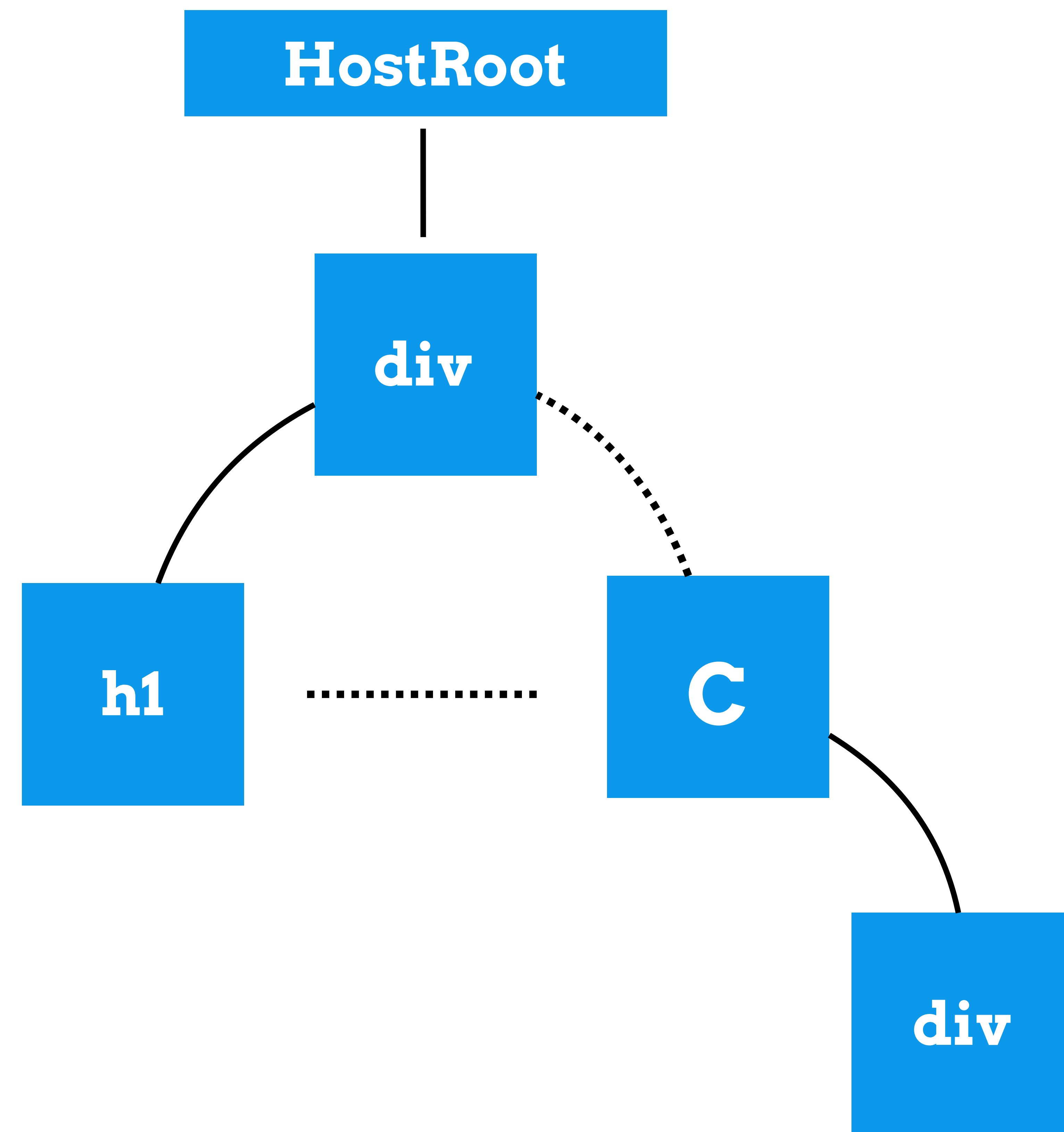


enqueueSetState(fiber, update)

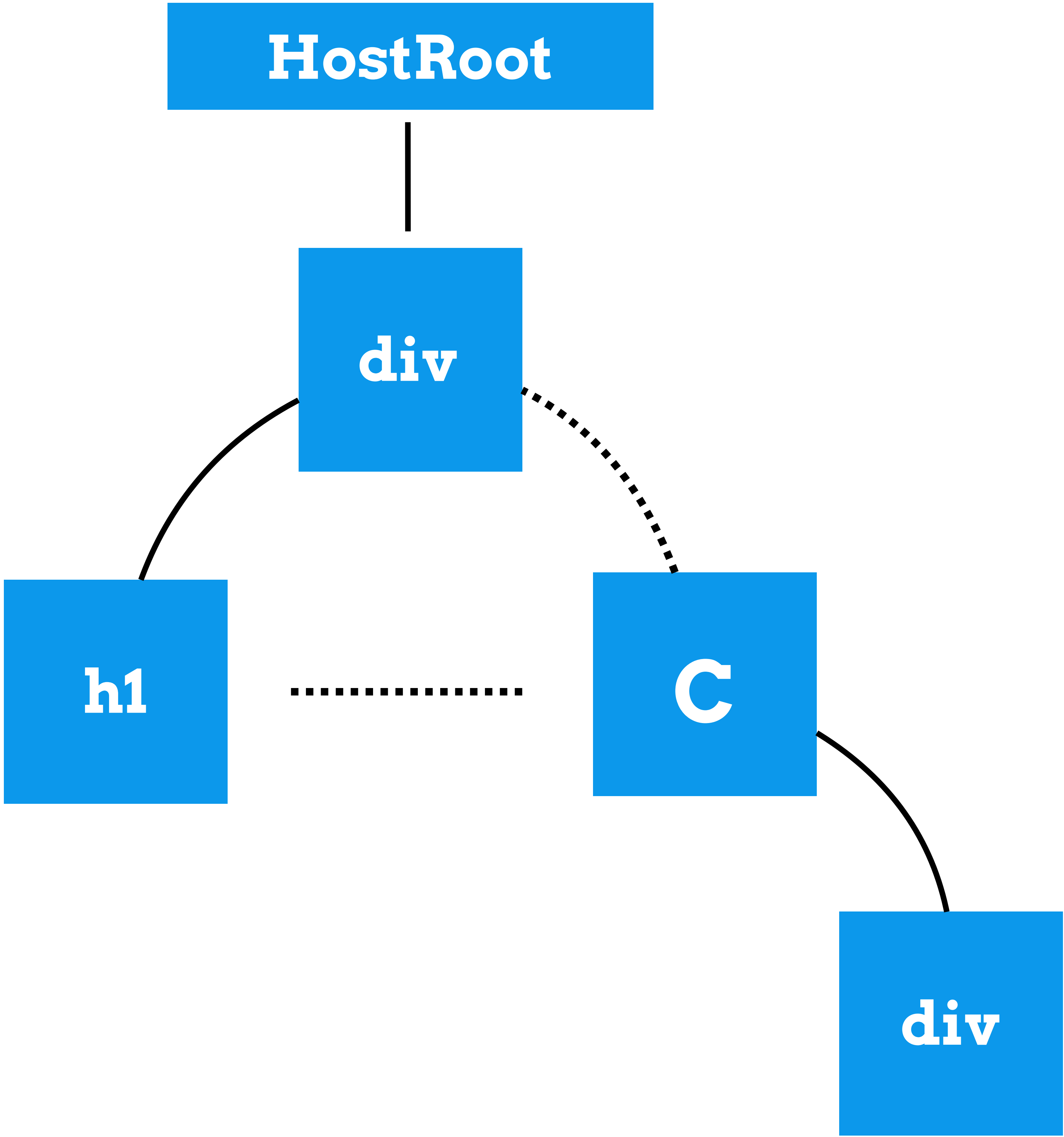
@GURLCODE

enqueueUpdate(fiber, update)

enqueueSetState(fiber, update)



@GURLCODE

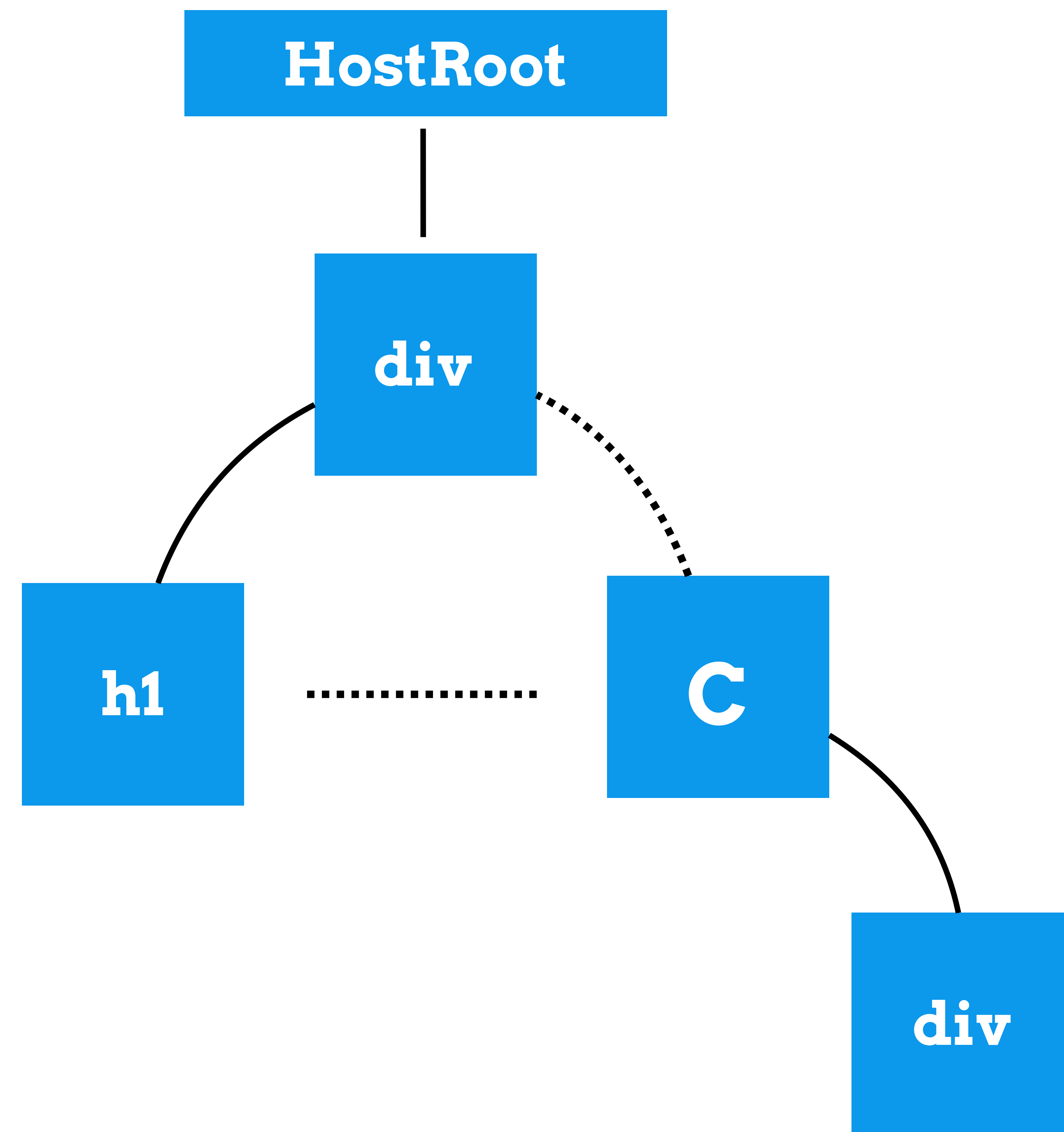


enqueueSetState(fiber, update)

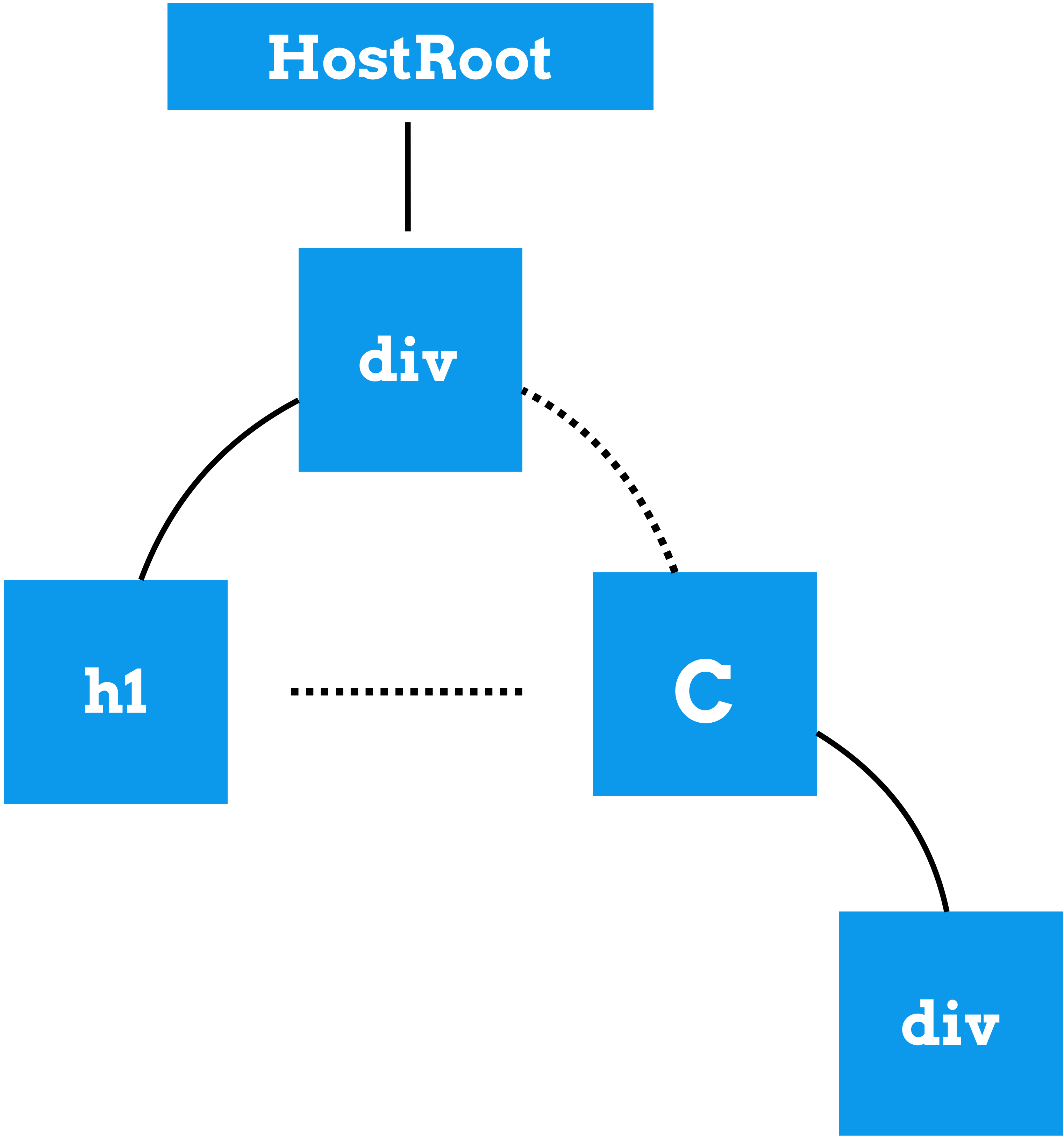
@GURLCODE

scheduleWork()

enqueueSetState(fiber, update)

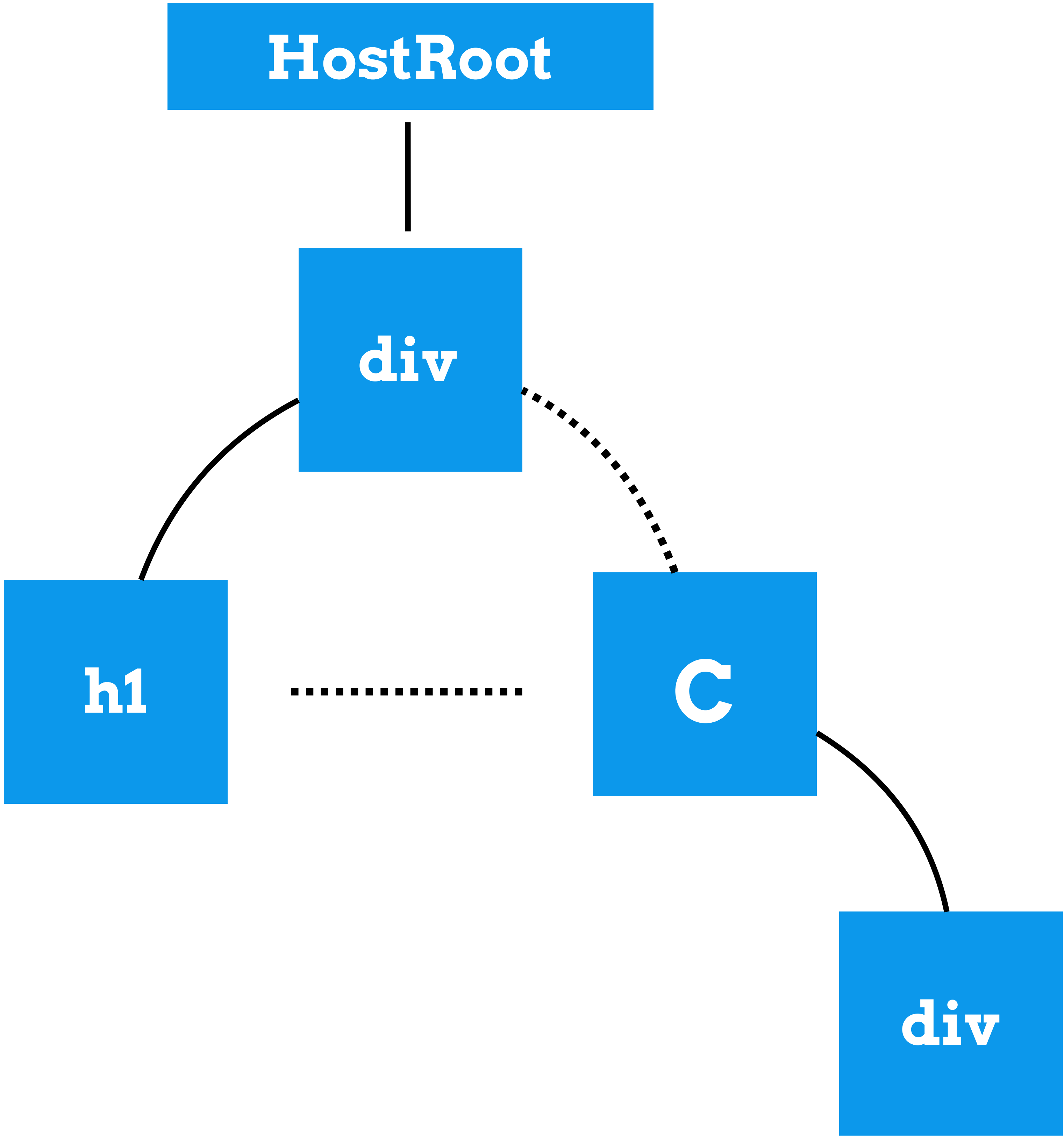


@GURLCODE



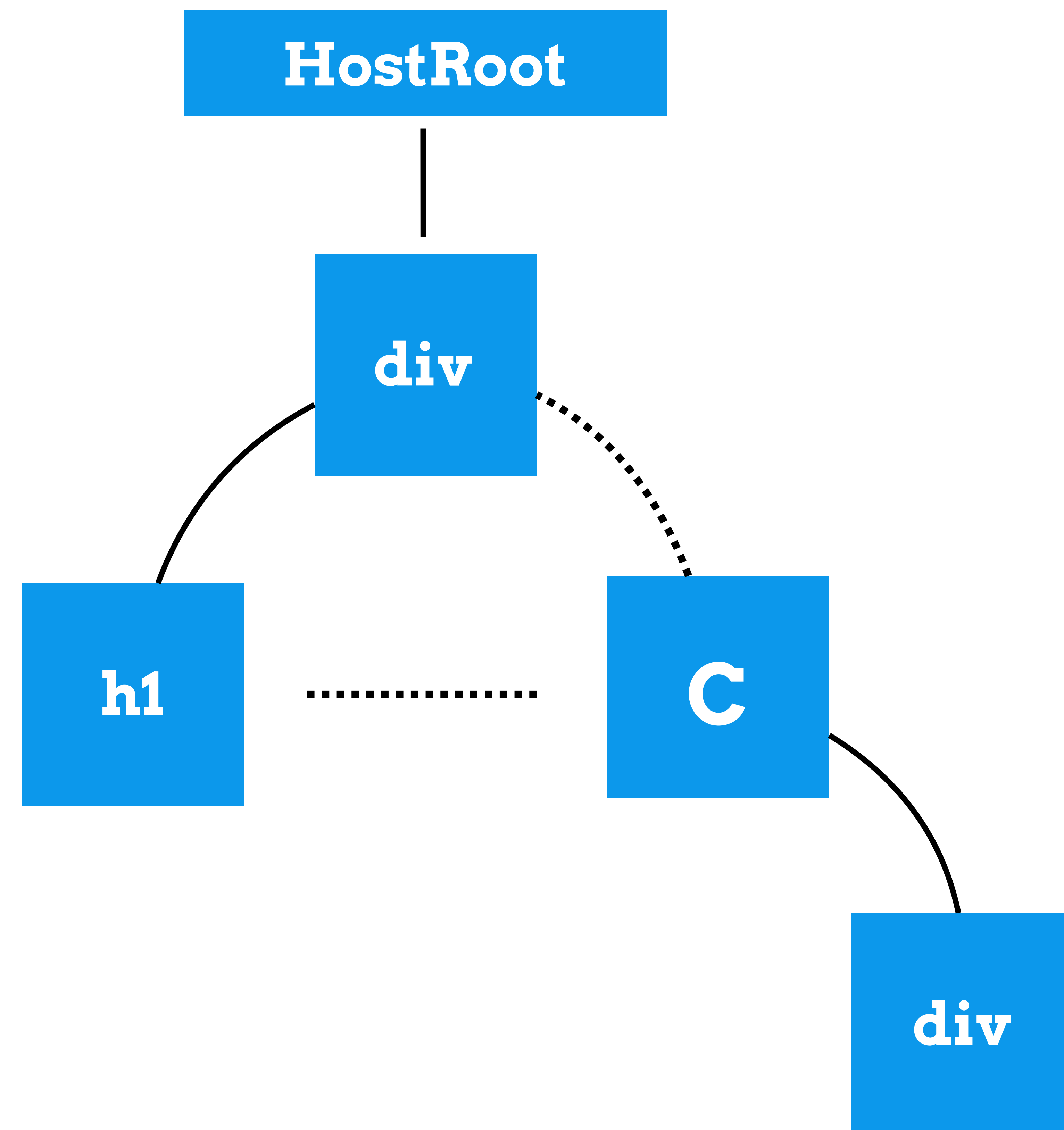
enqueueSetState(fiber, update)

@GURLCODE



performWork

@GURLCODE



renderRoot

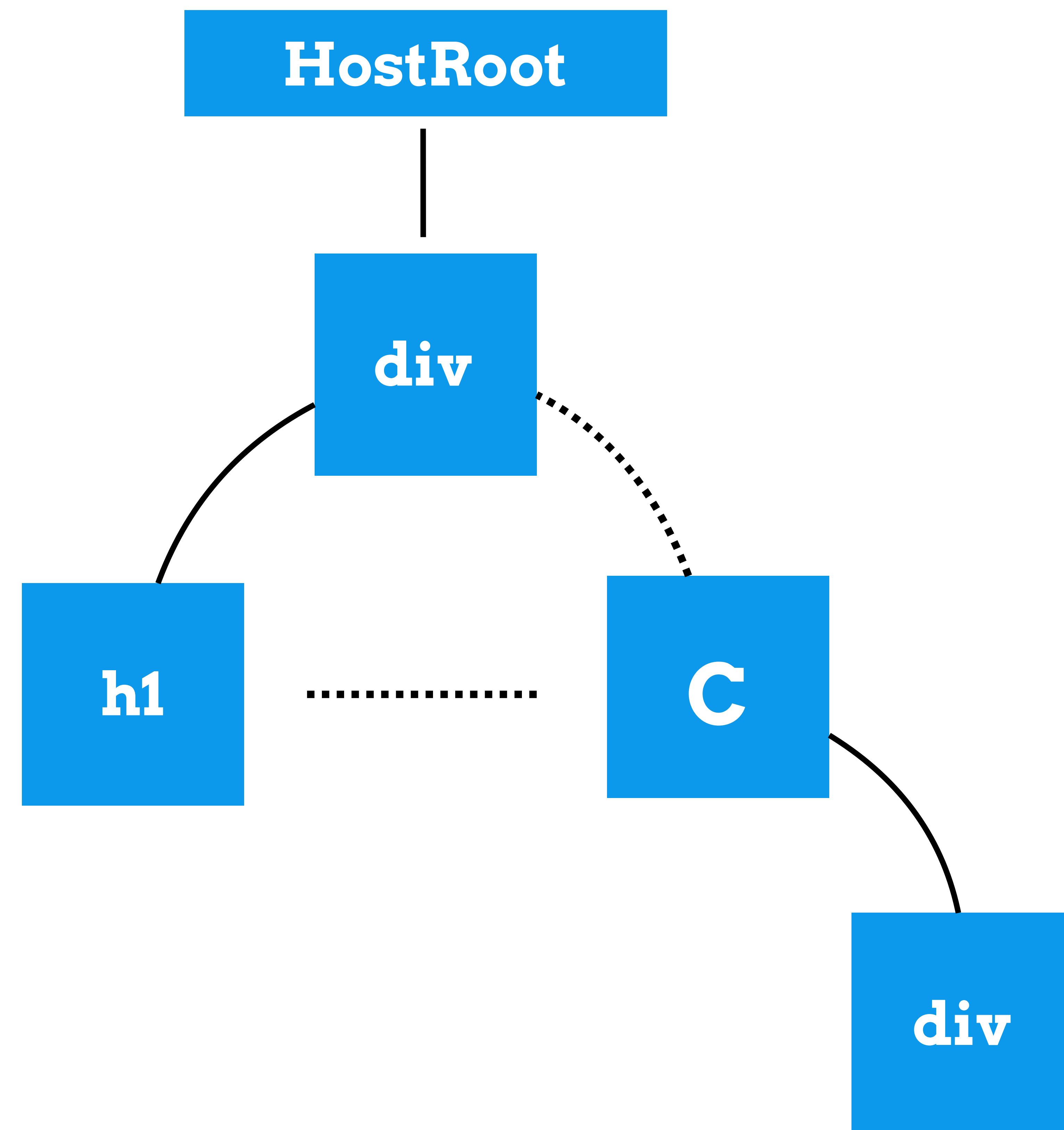
performWork

@GURLCODE

workInProgress = current.alternate

renderRoot

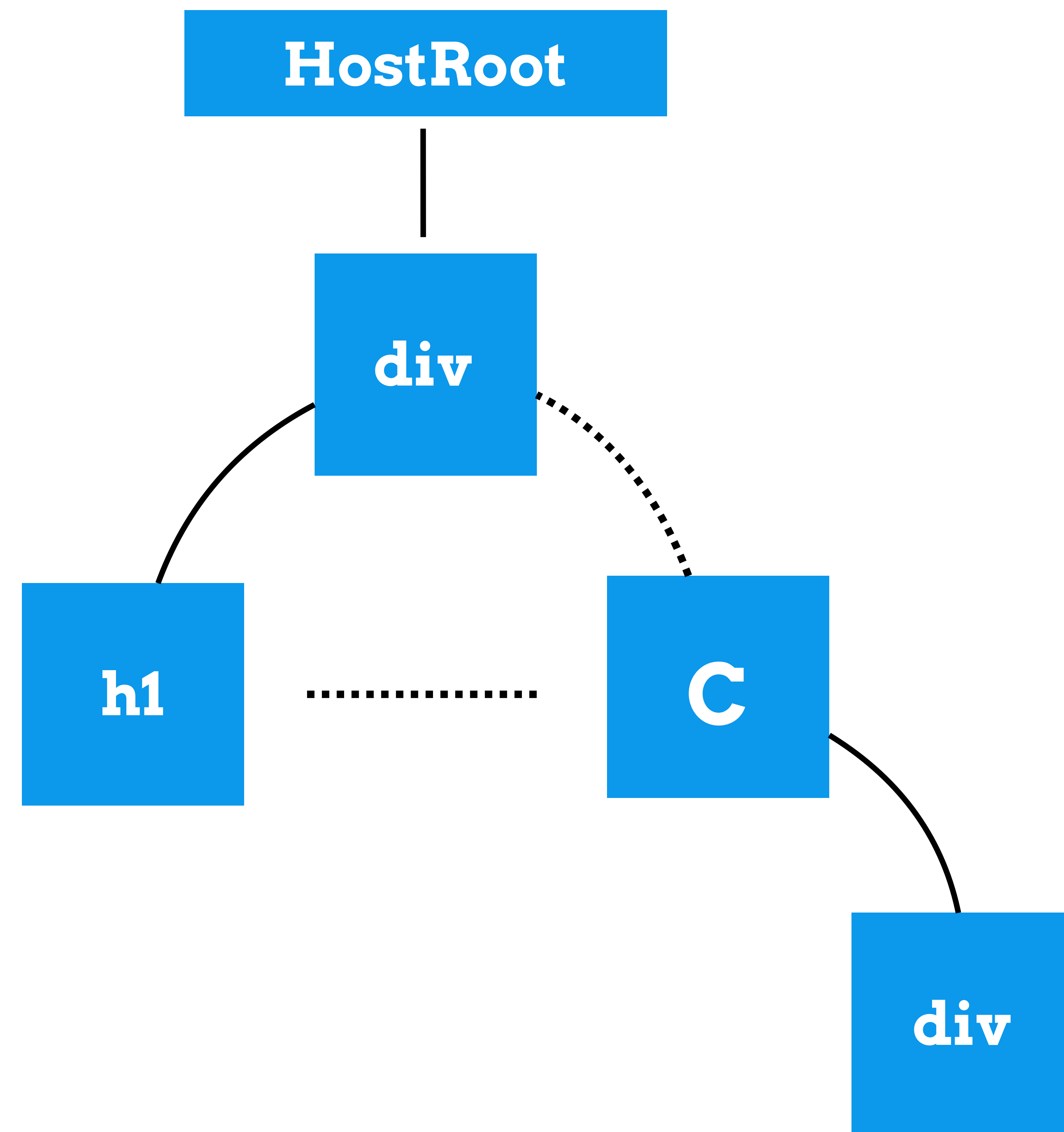
performWork



@GURLCODE

nextUnitOfWork = workInProgress

performWork

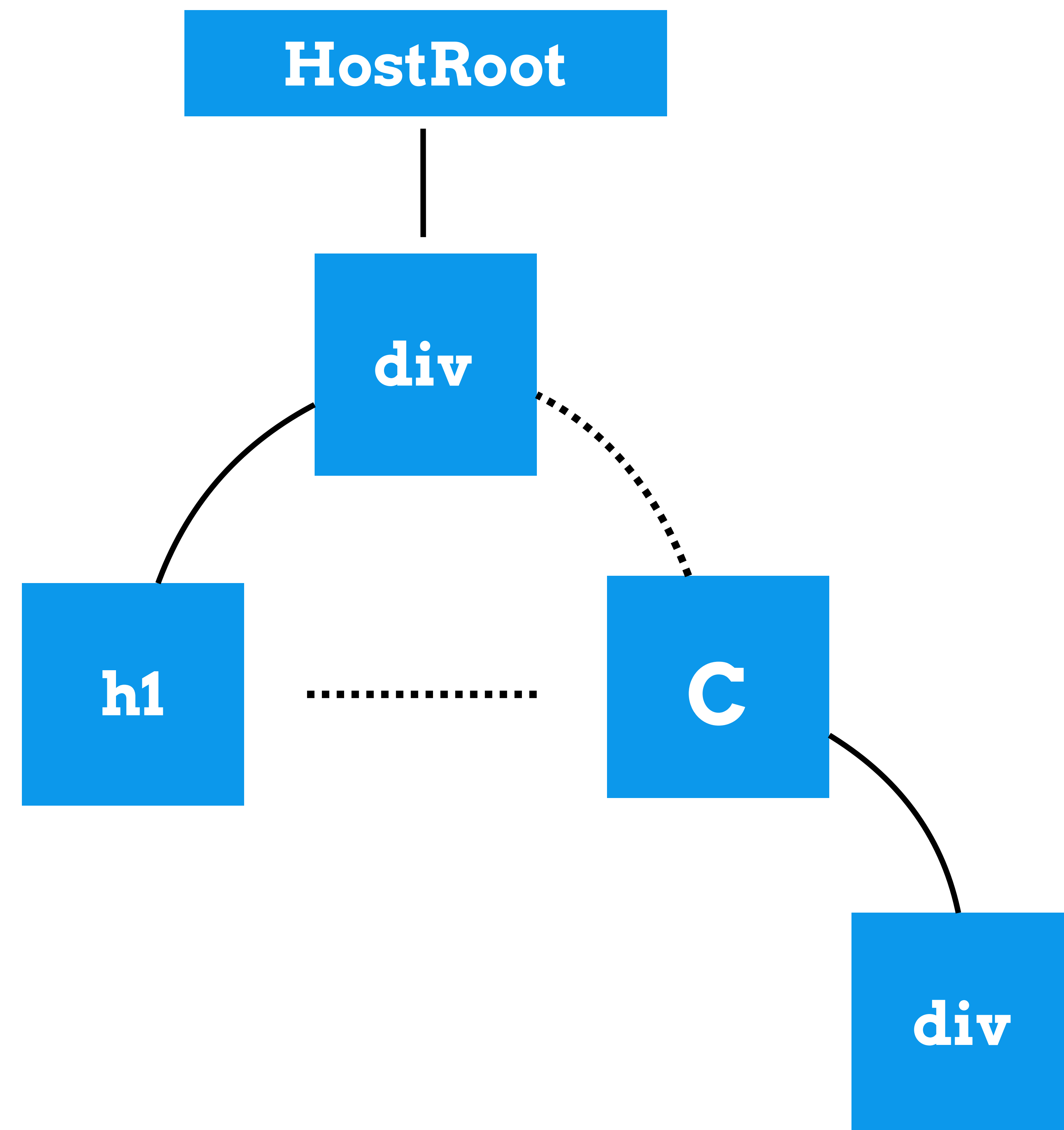


@GURLCODE

workLoop

nextUnitOfWork = workInProgress

performWork



@GURLCODE

HostRoot

div

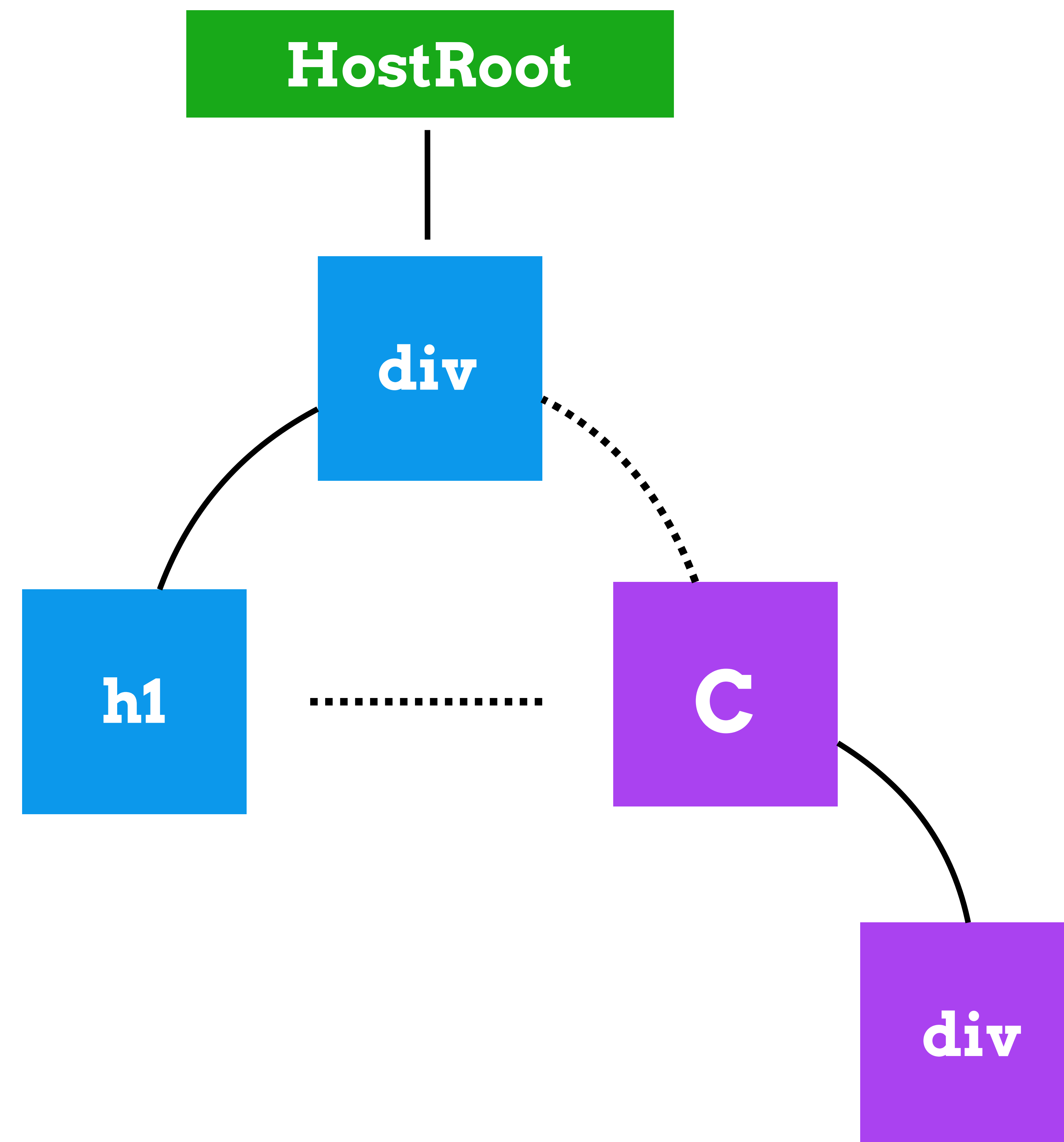
h1

C

div

workLoop

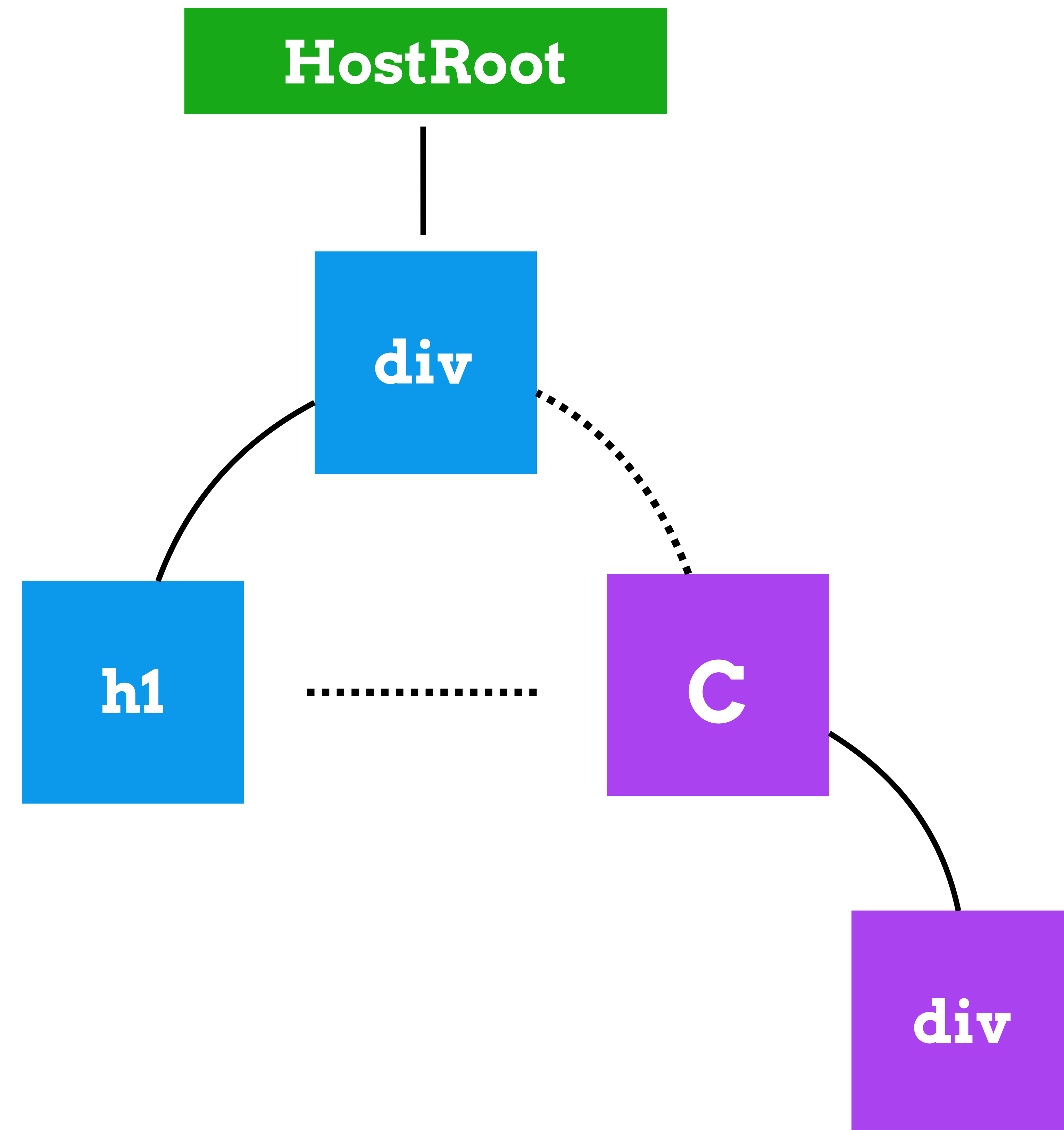
@GURLCODE

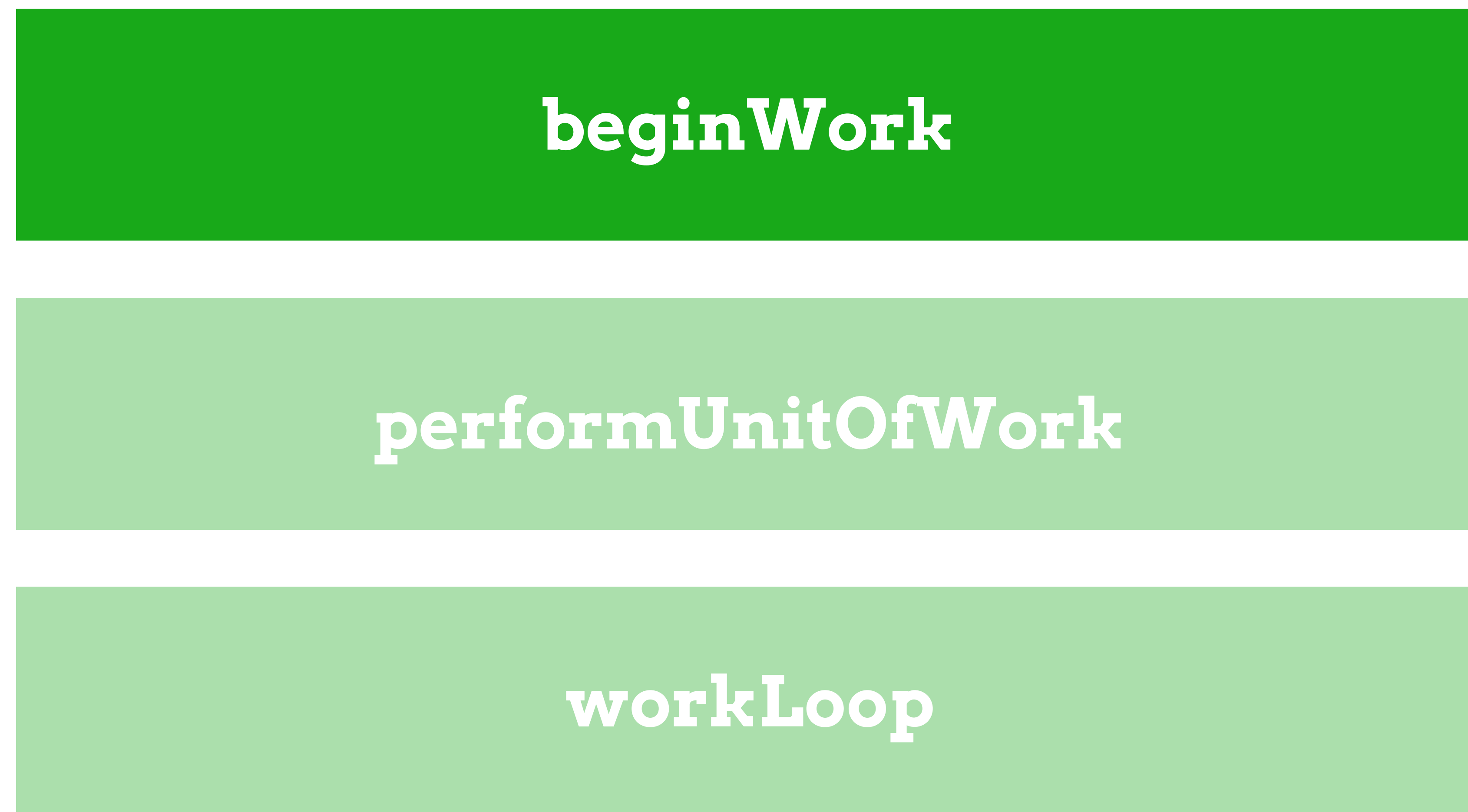


performUnitOfWork

workLoop

@GURLCODE



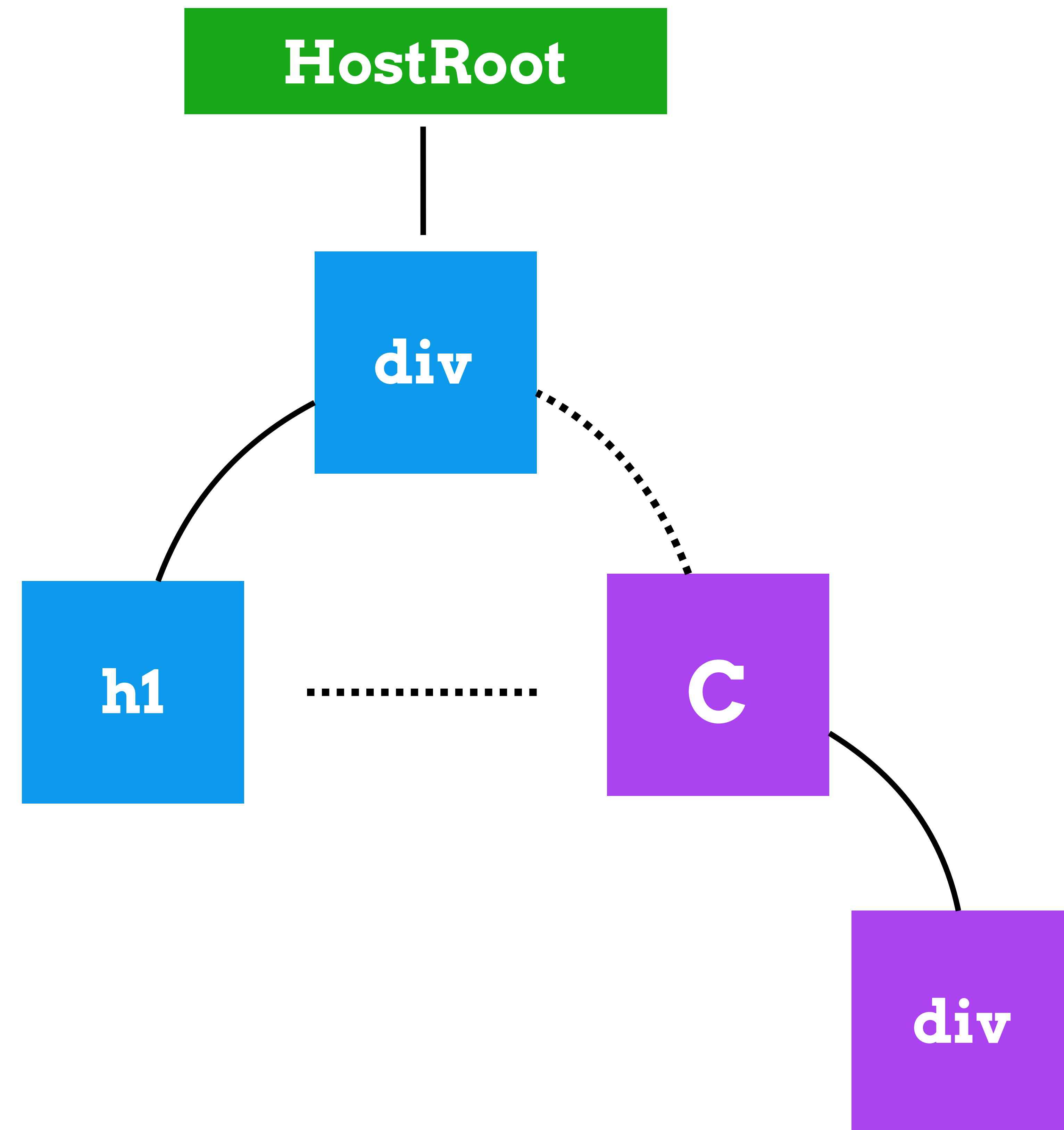


@GURLCODE

oldProps !== newProps

performUnitOfWork

workLoop

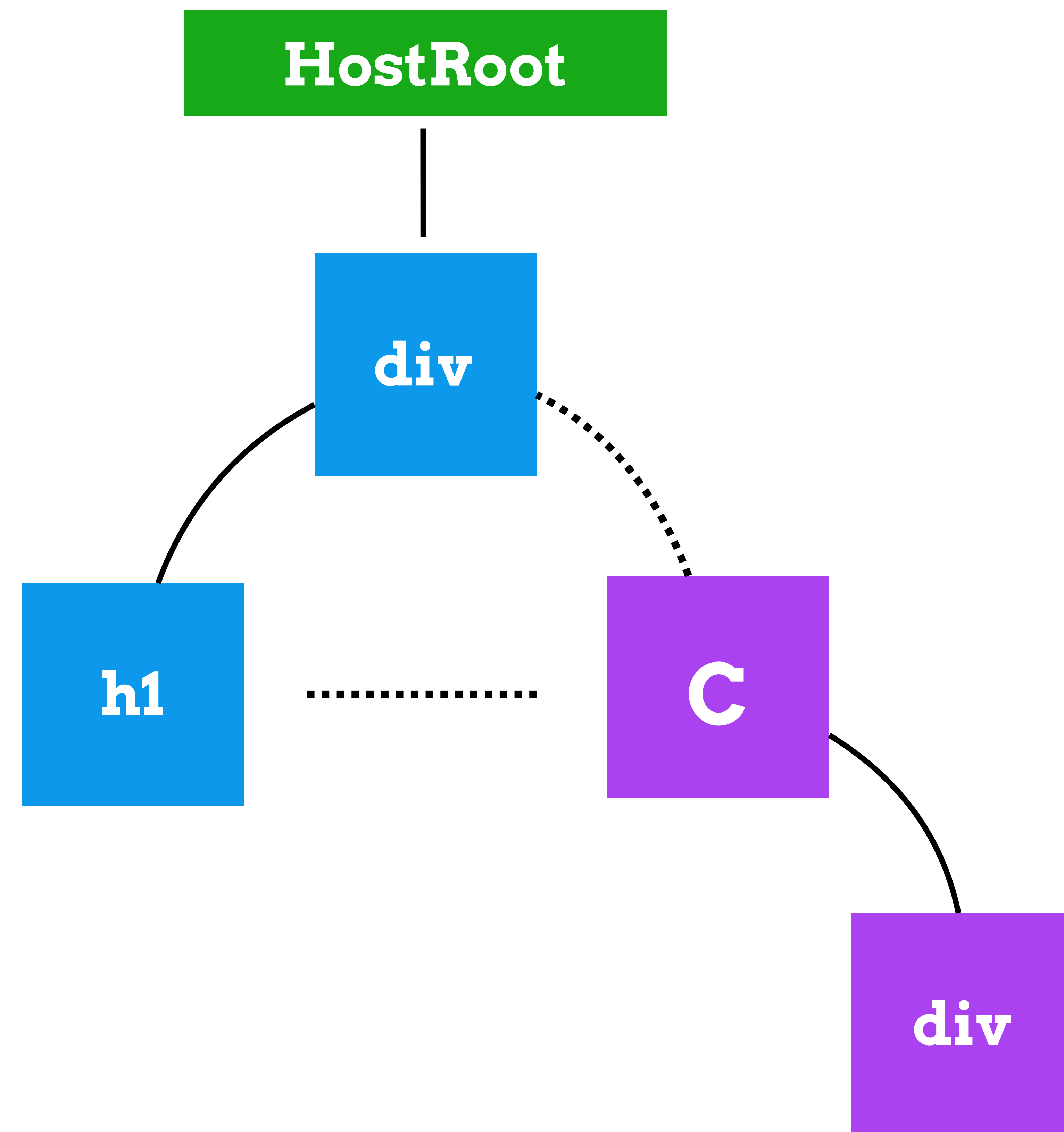


@GURLCODE

return workInProgress.child

performUnitOfWork

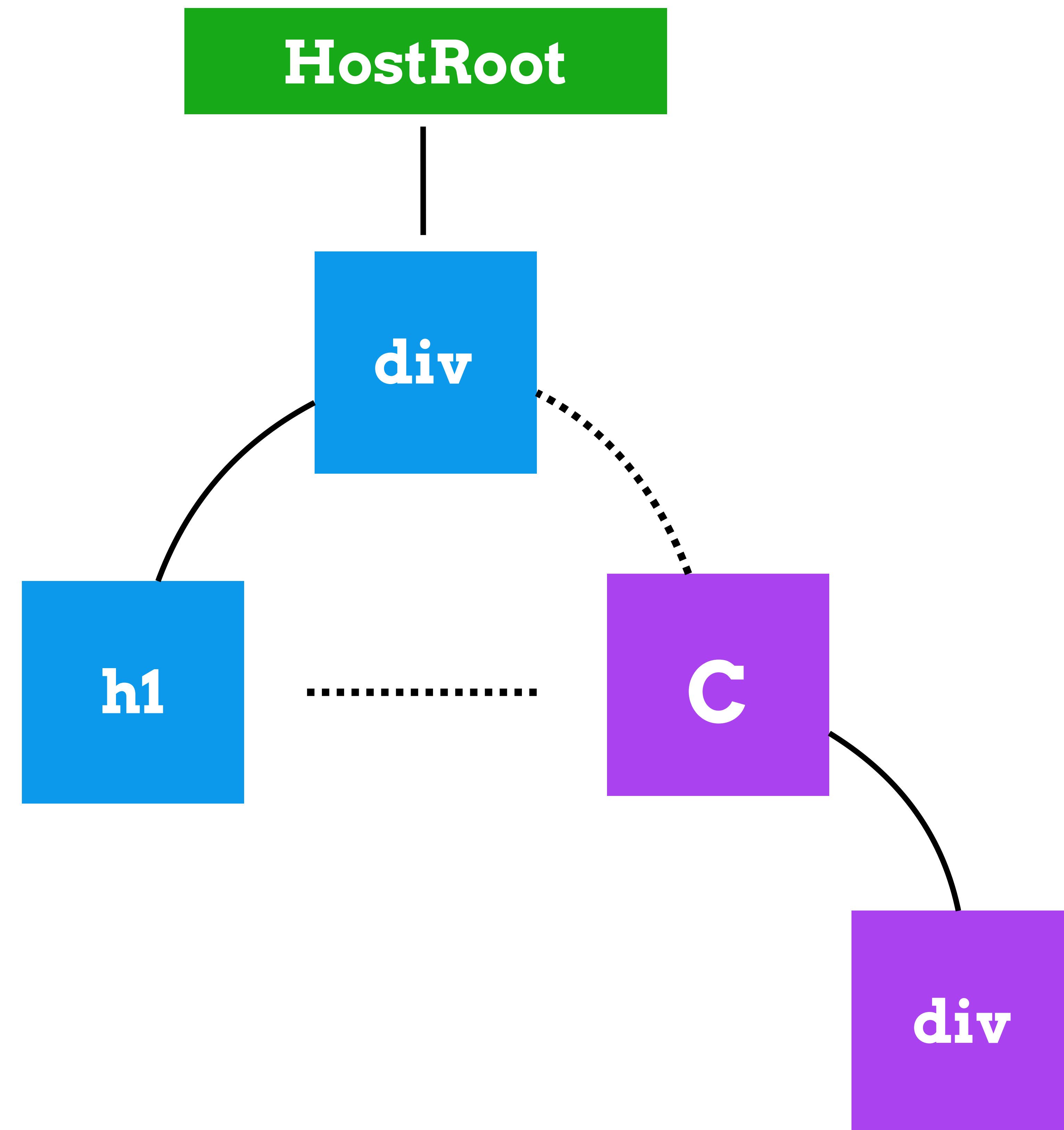
workLoop



@GURLCODE

return workInProgress.child

workLoop



@GURLCODE

HostRoot

div

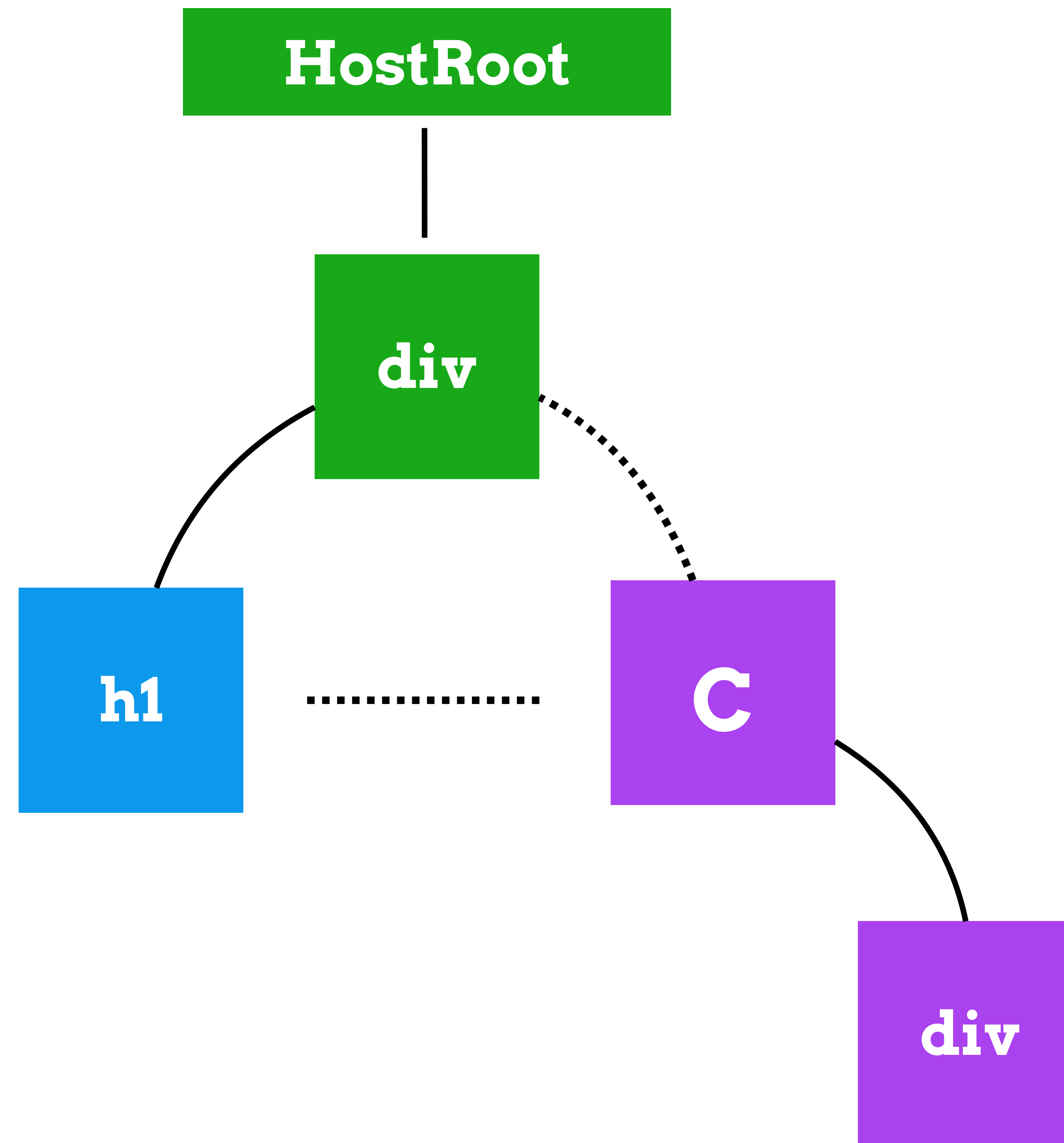
h1

C

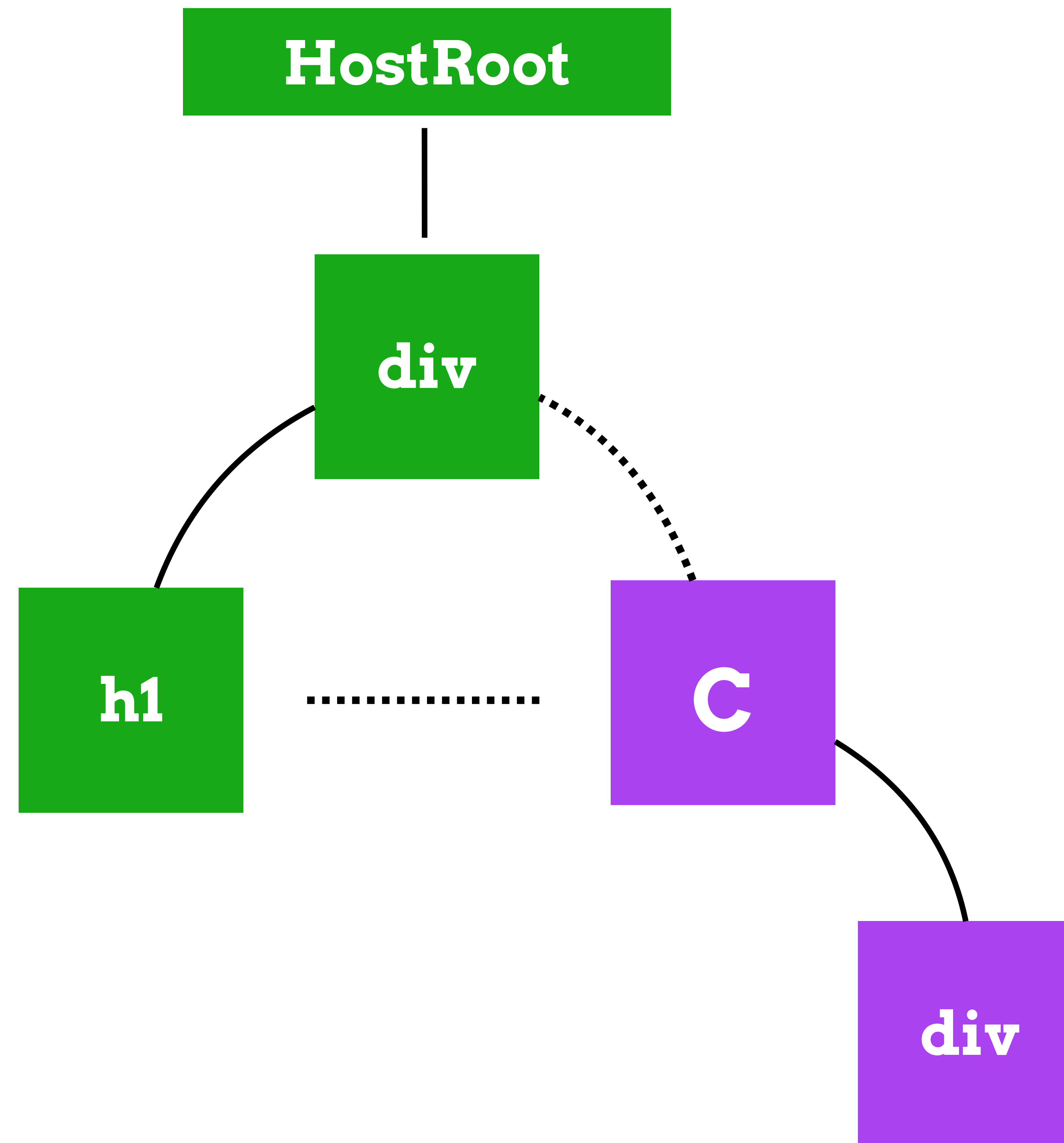
div

workLoop

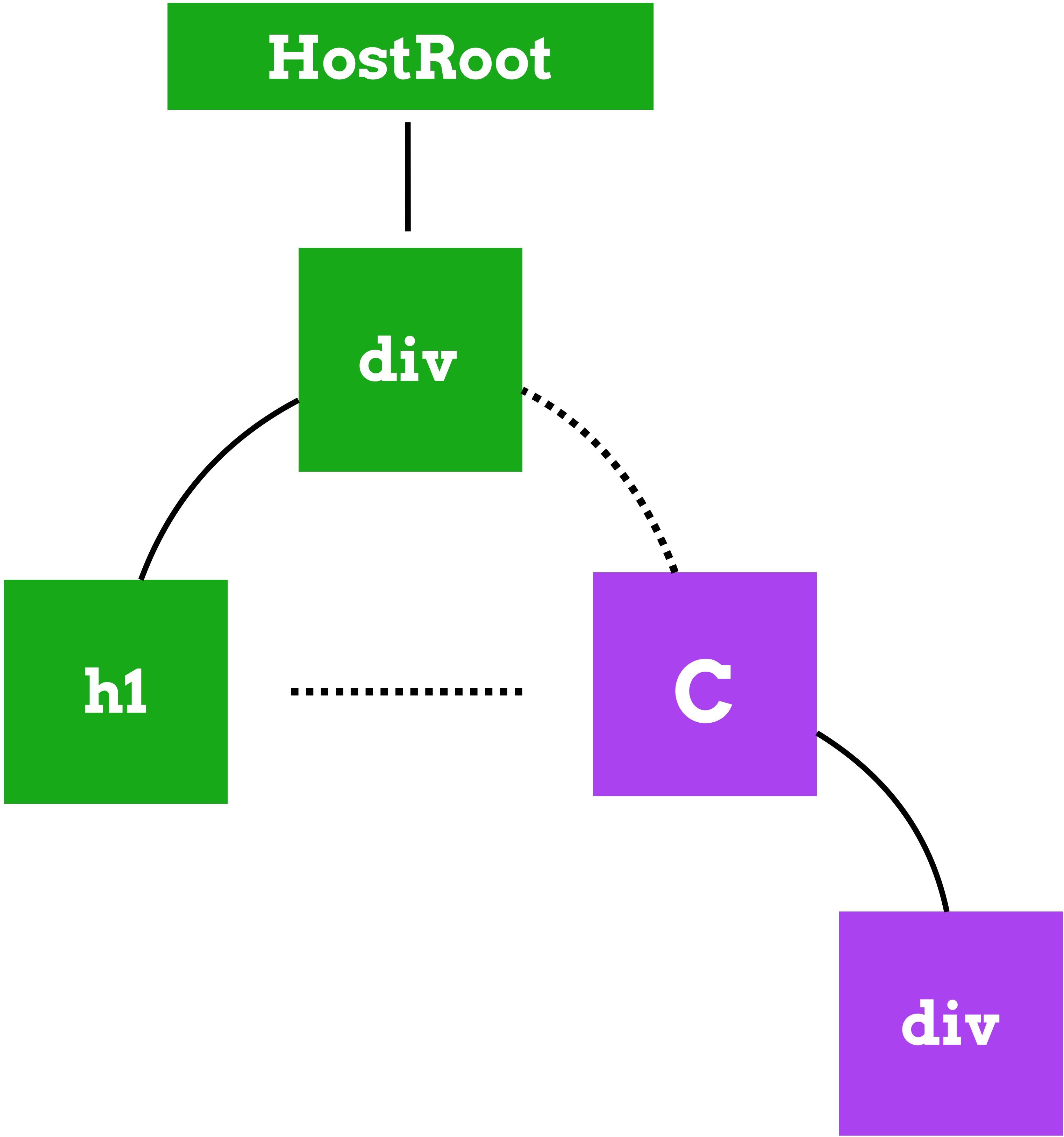
@GURLCODE



@GURLCODE

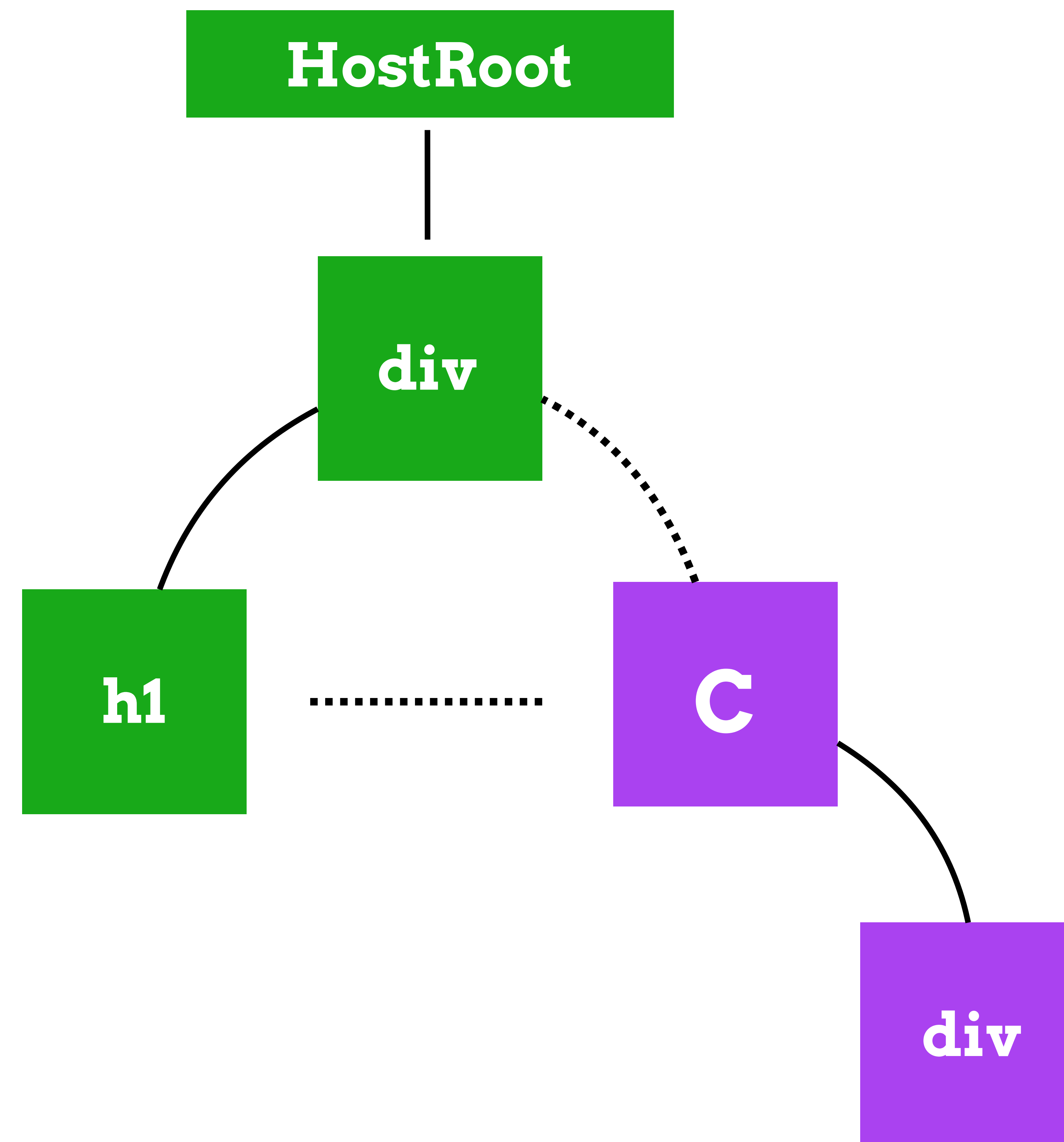


@GURLCODE



workLoop

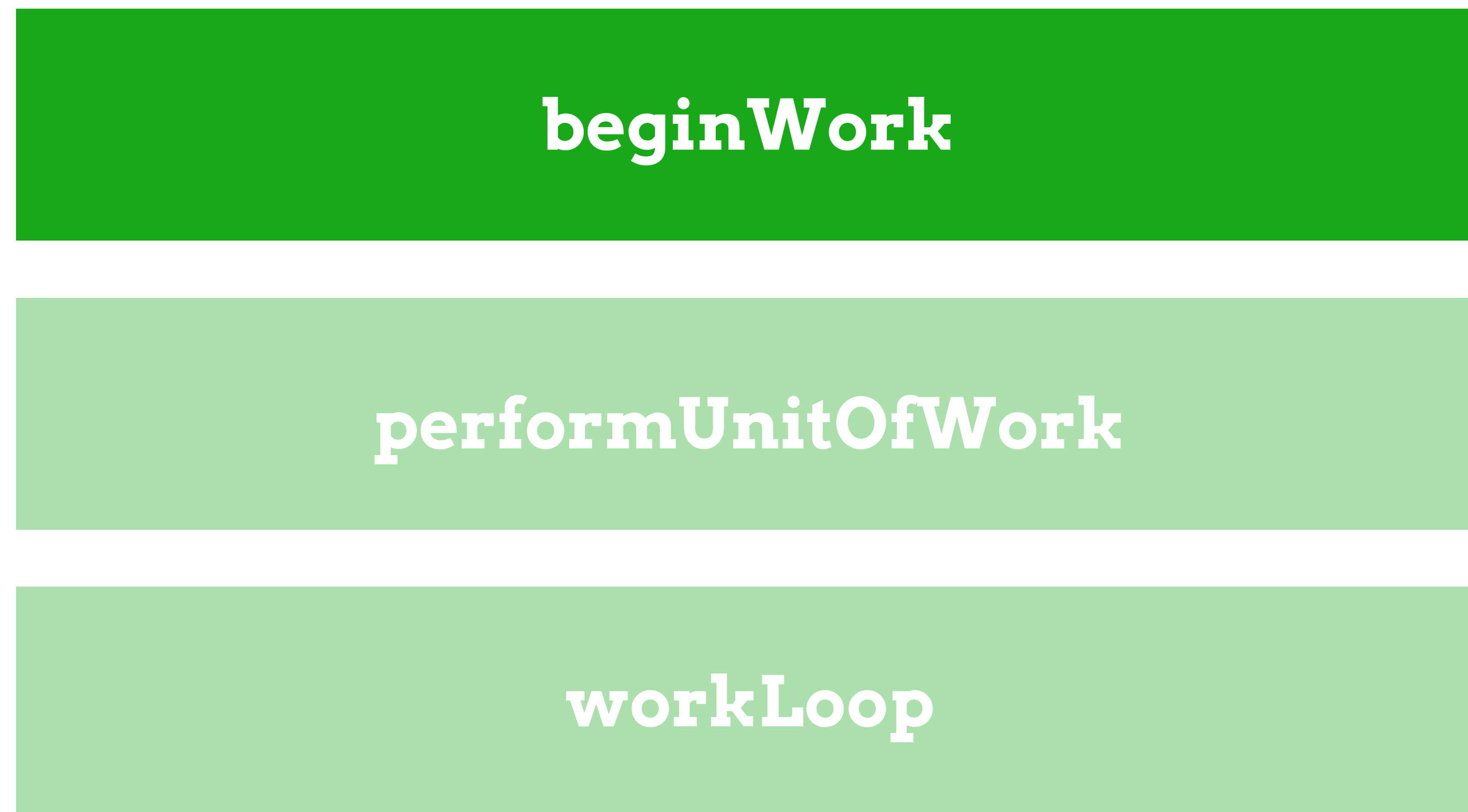
@GURLCODE

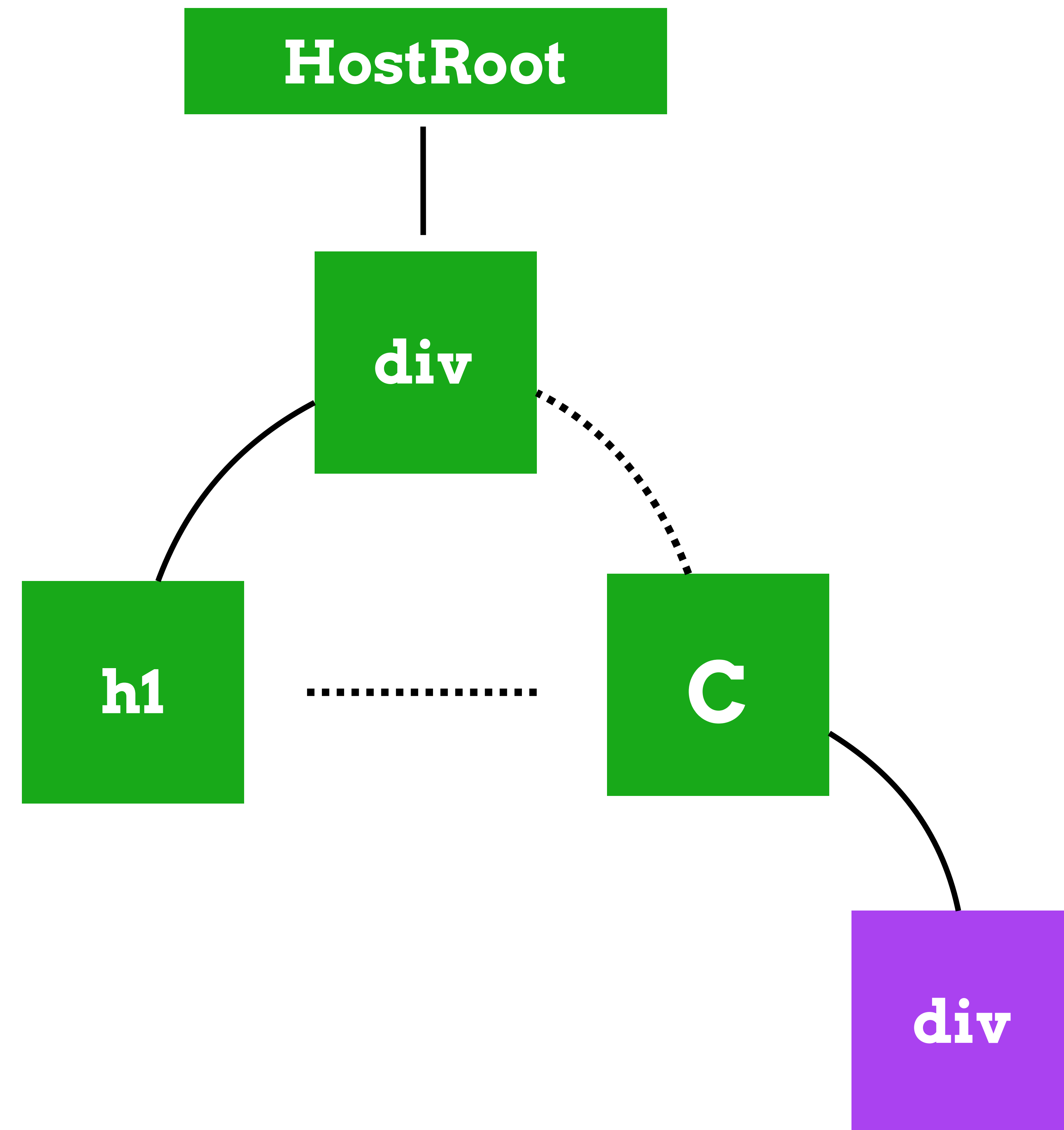


performUnitOfWork

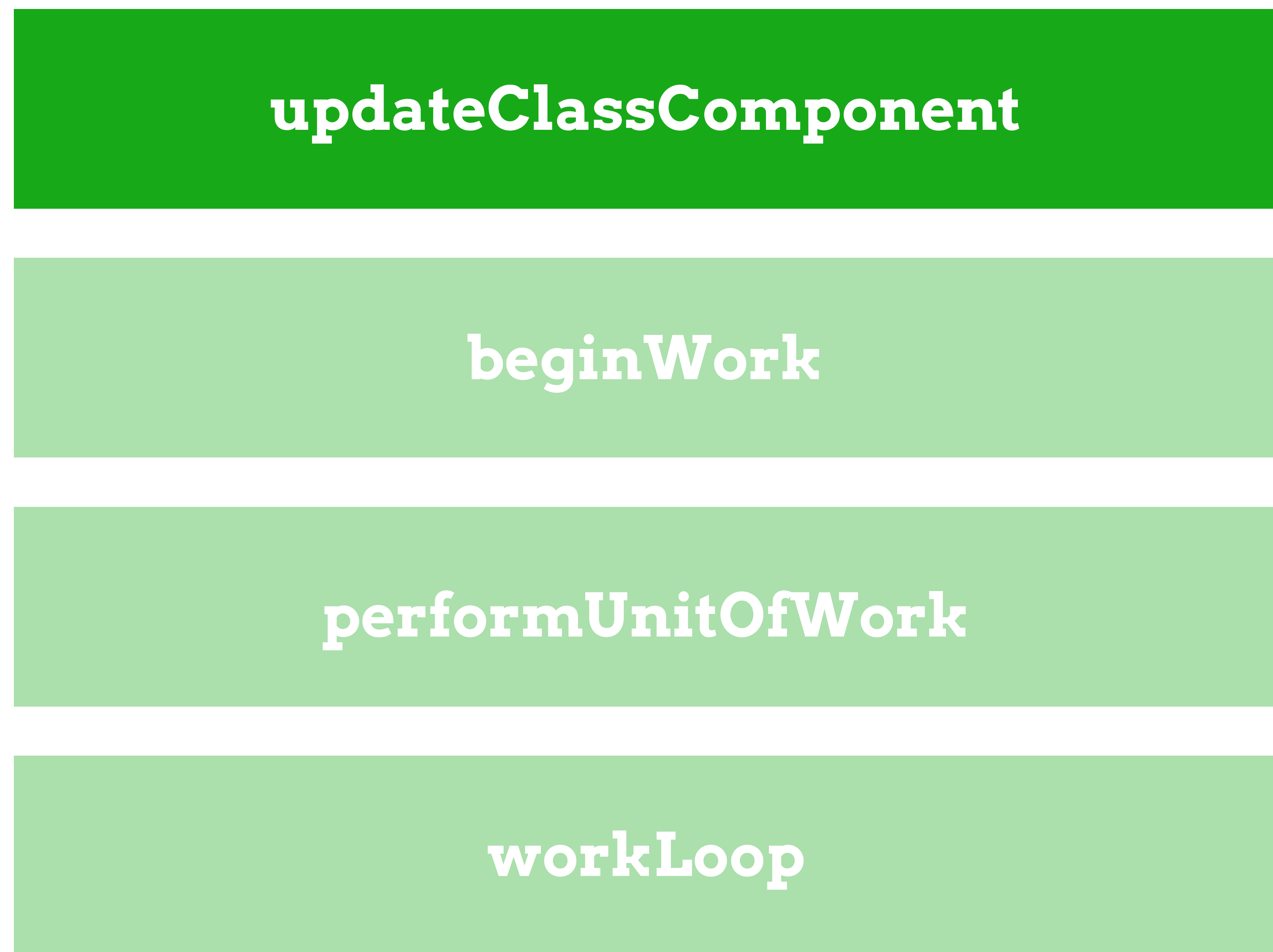
workLoop

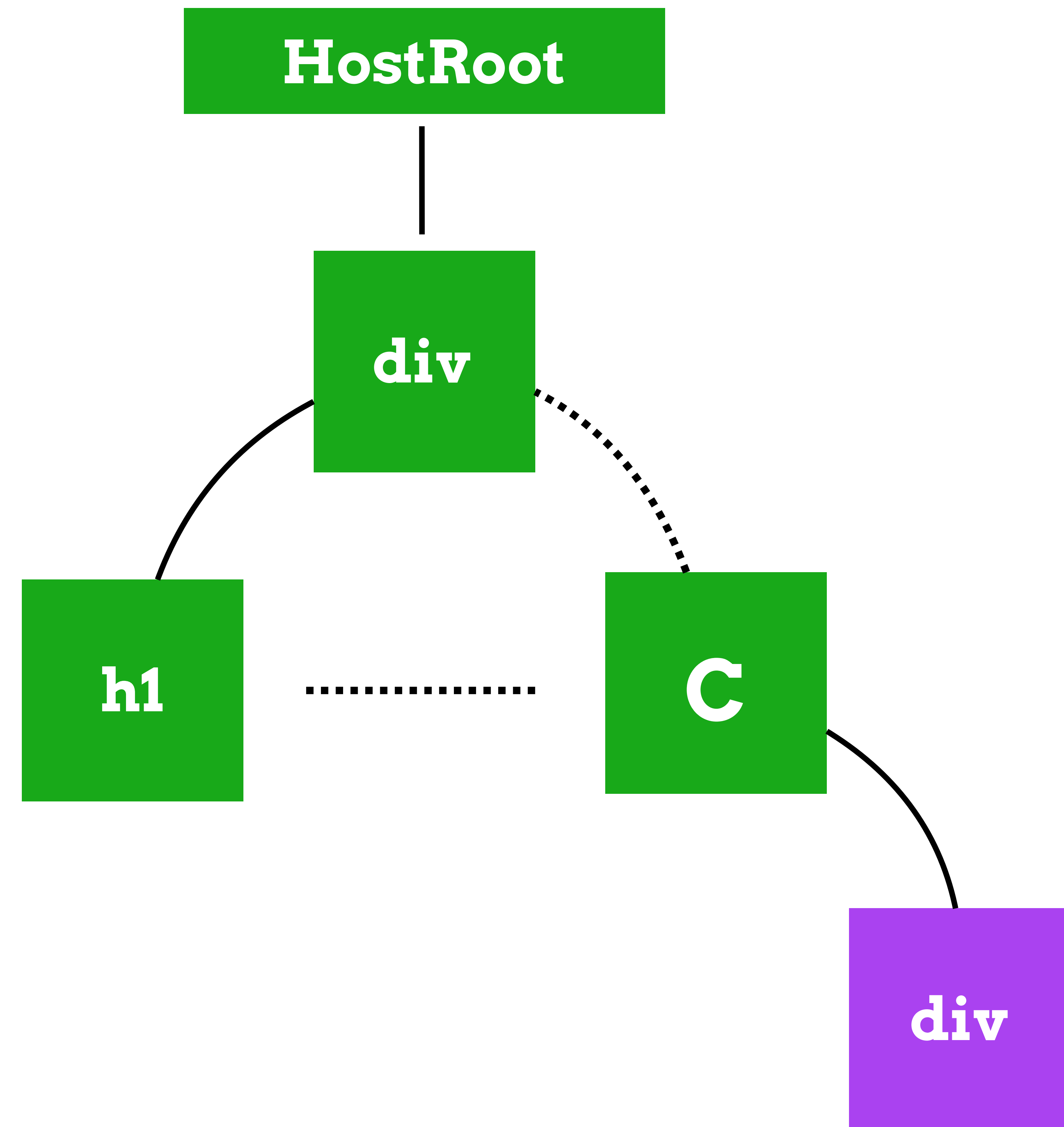
@GURLCODE



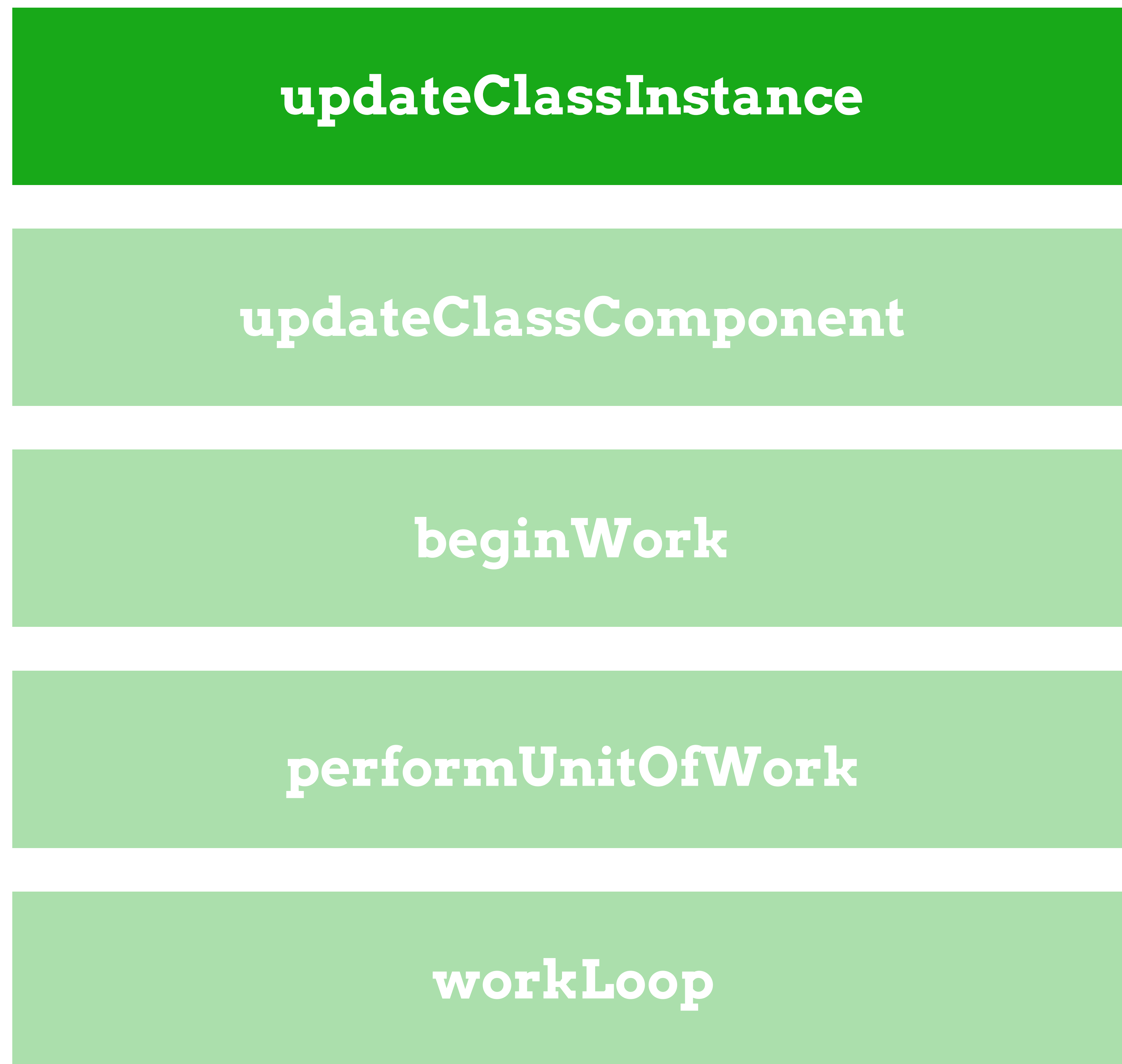


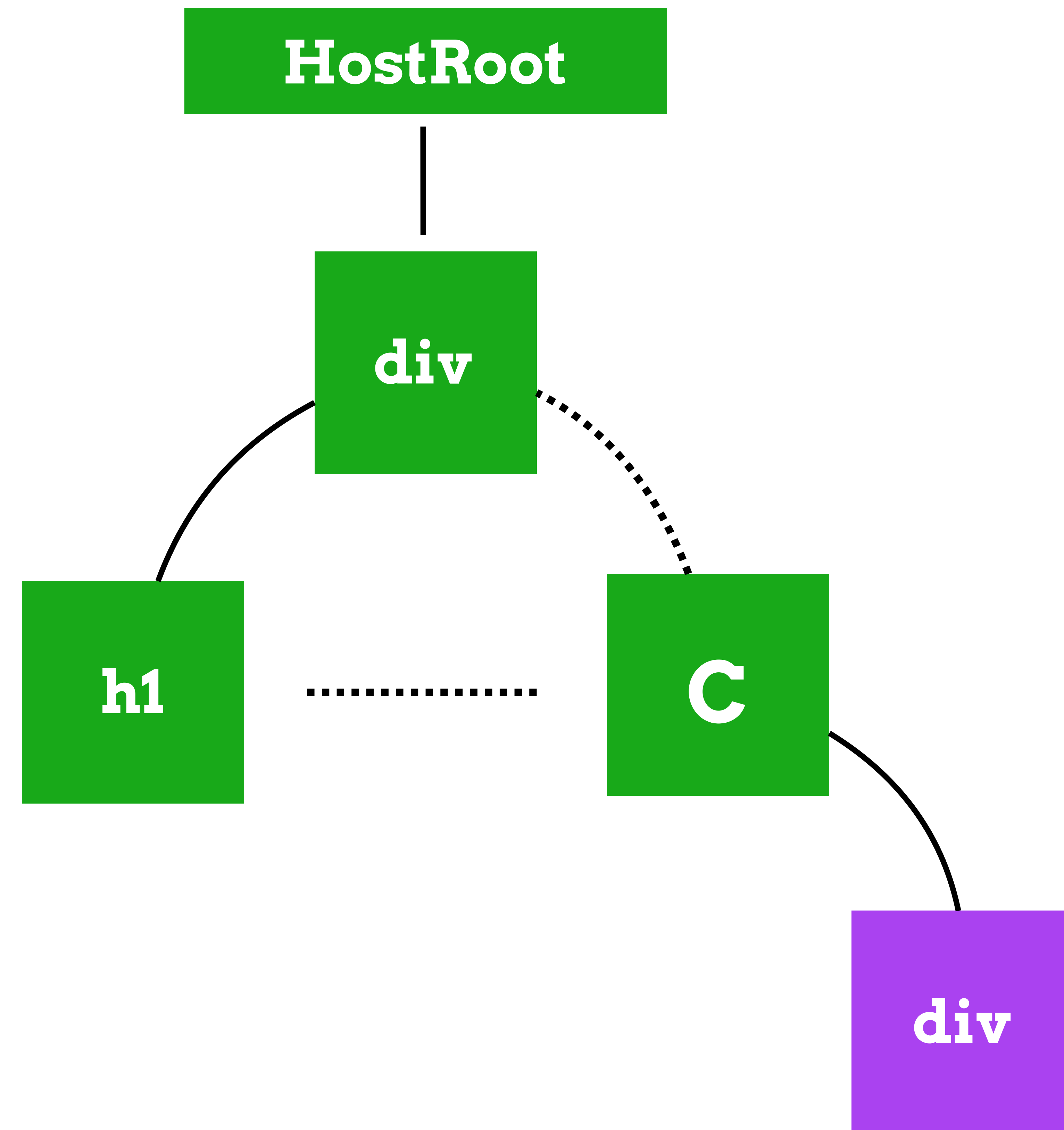
@GURLCODE





@GURLCODE





@GURLCODE

processUpdateQueue

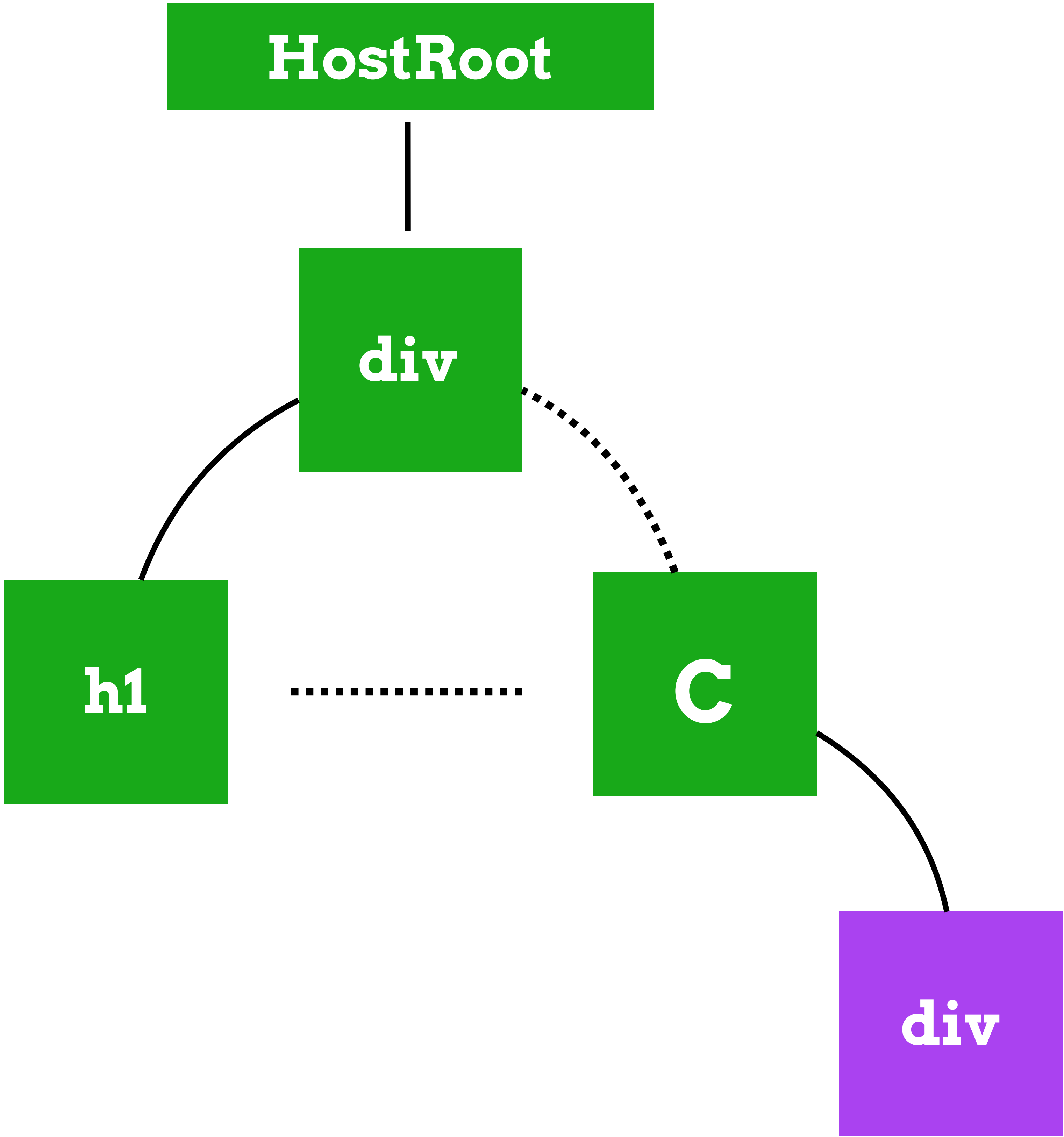
updateClassInstance

updateClassComponent

beginWork

performUnitOfWork

workLoop



@GURLCODE

fiber.memoizedState = newState

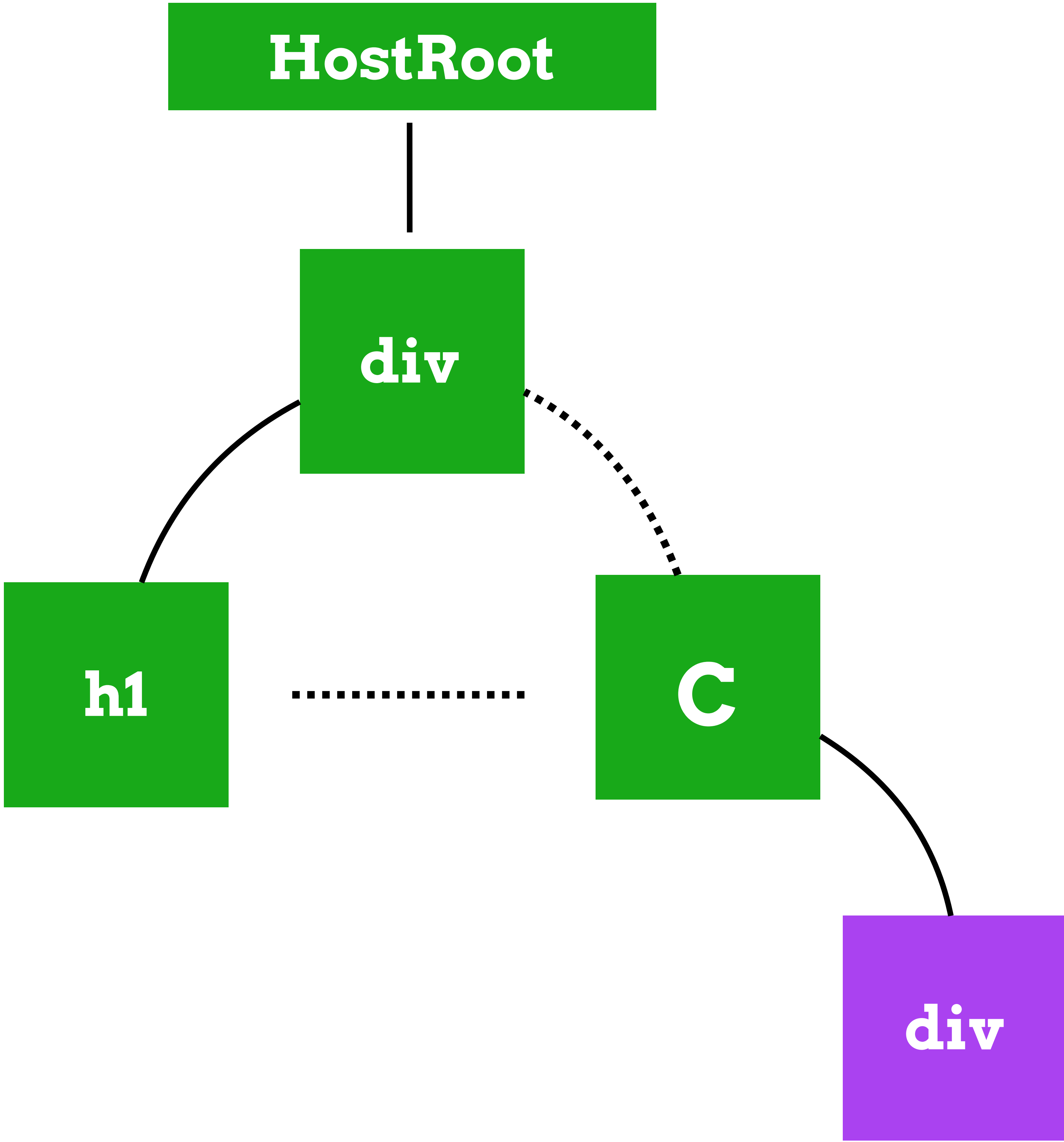
updateClassInstance

updateClassComponent

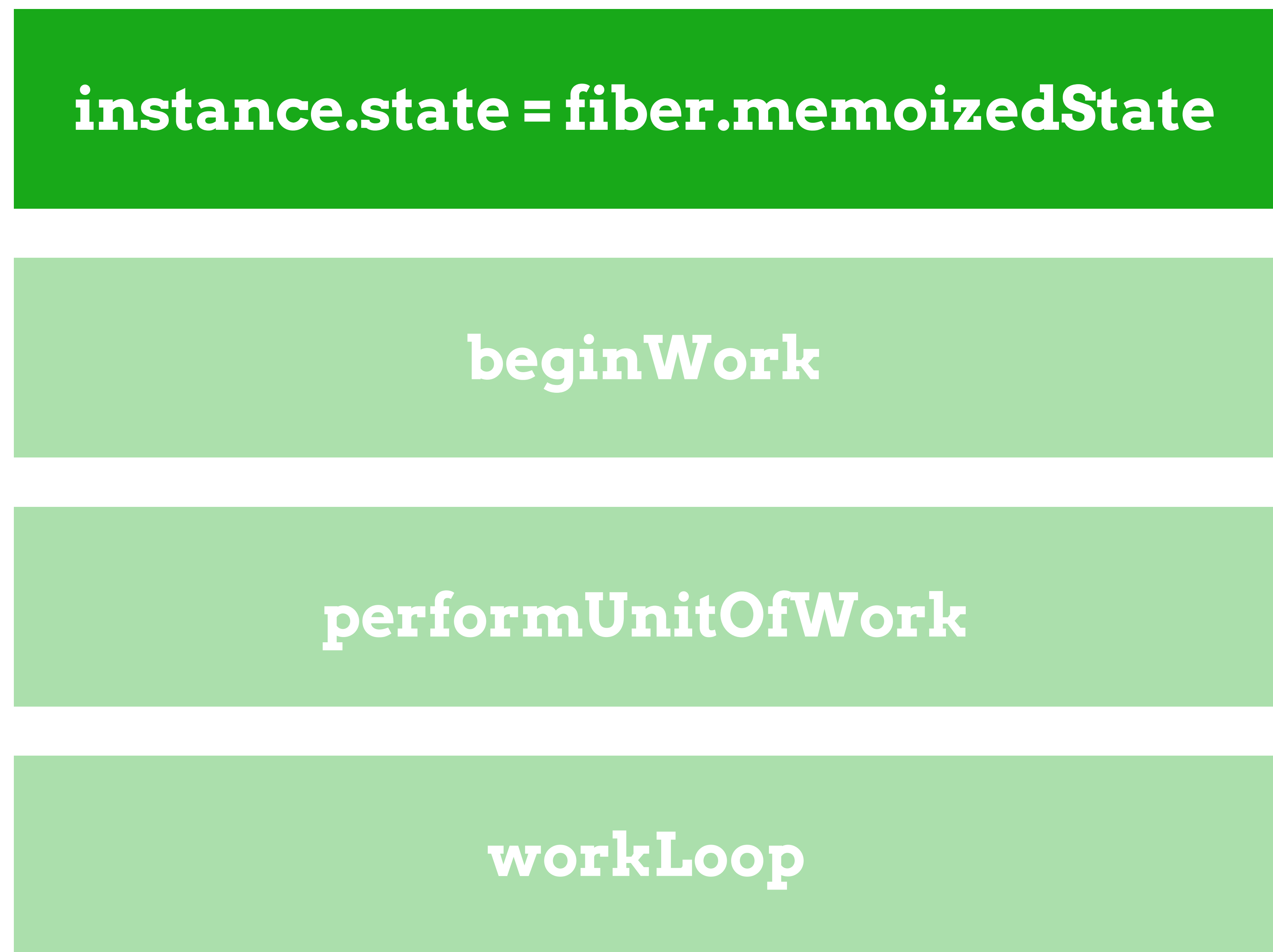
beginWork

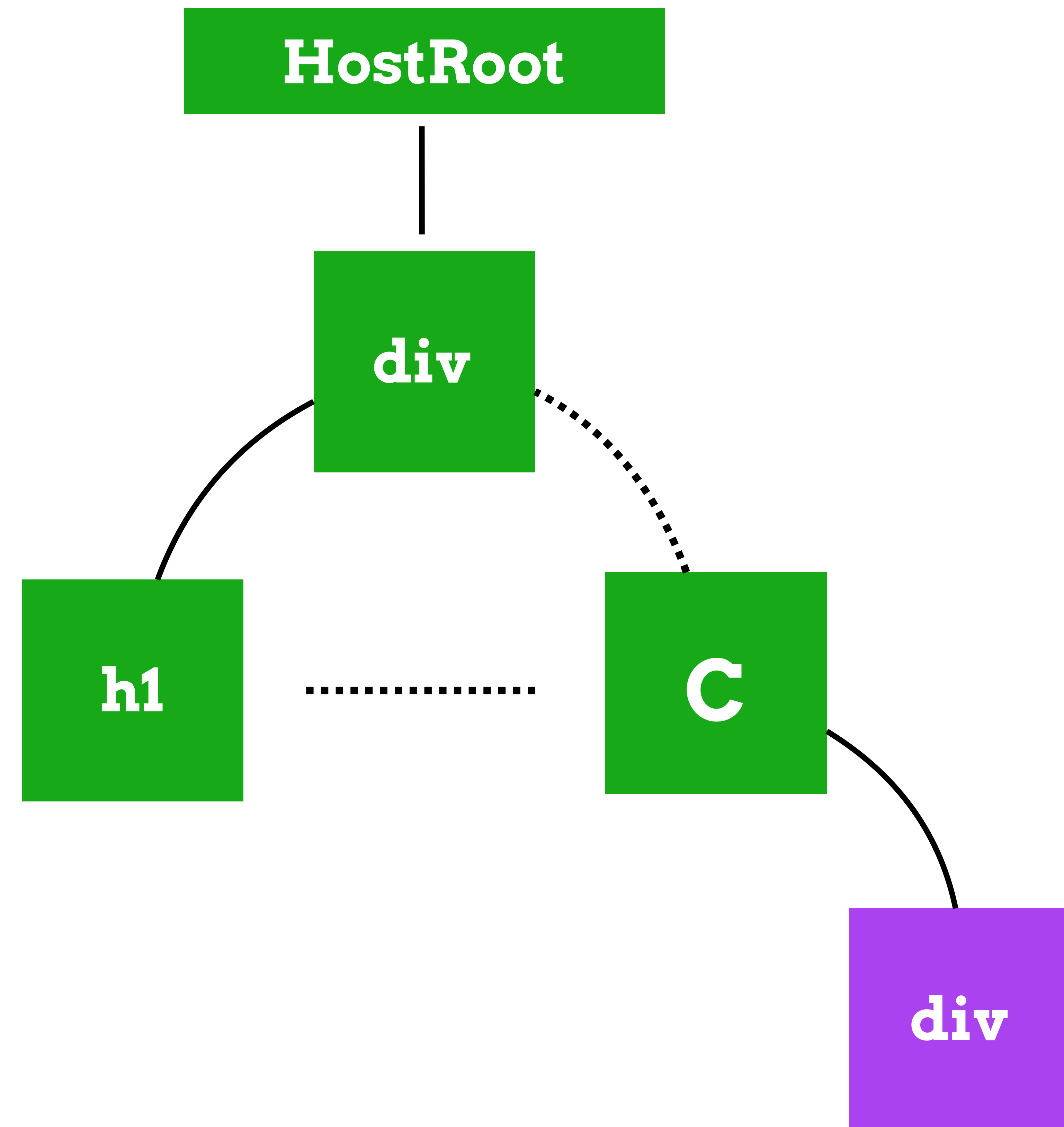
performUnitOfWork

workLoop

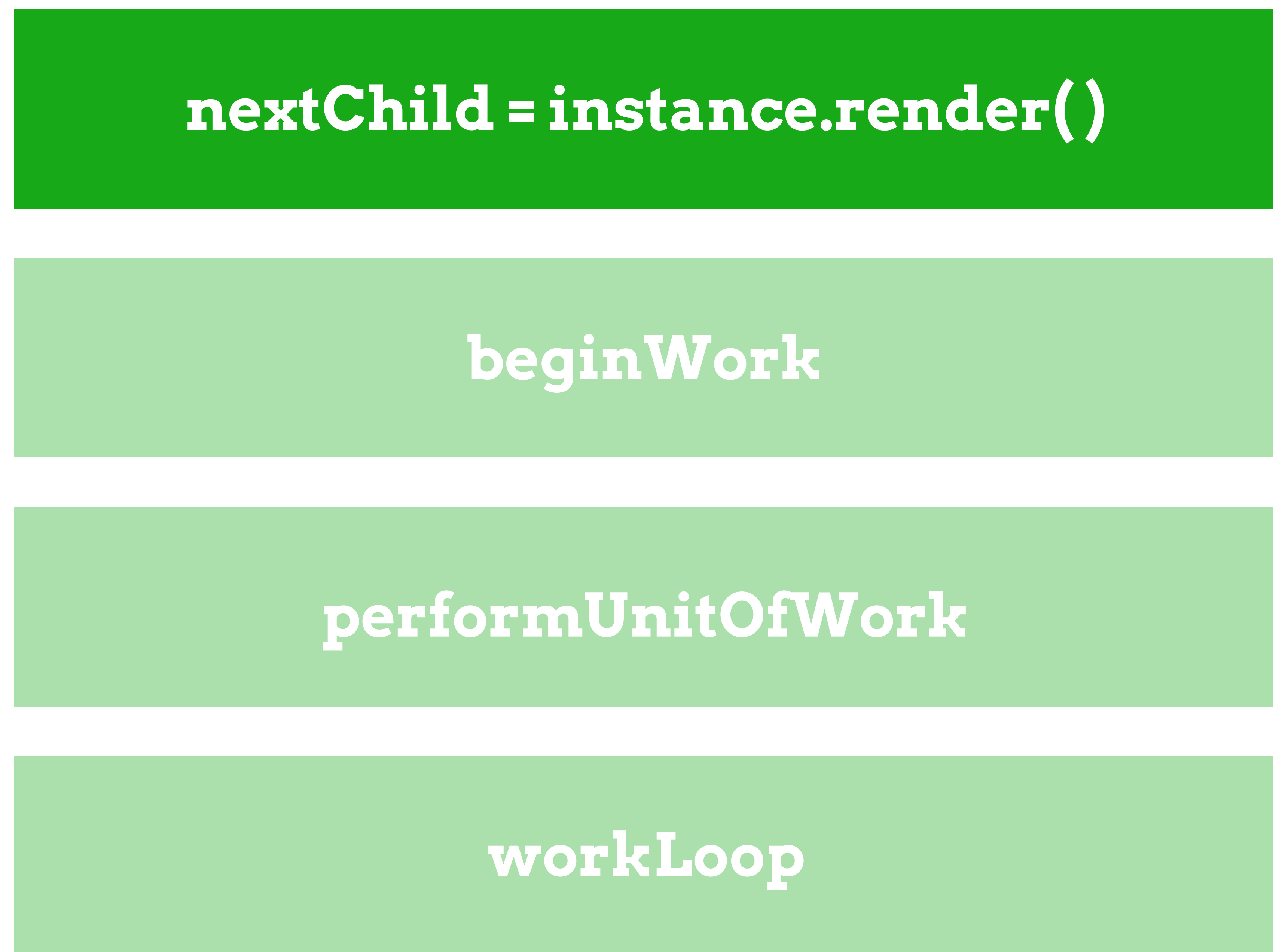


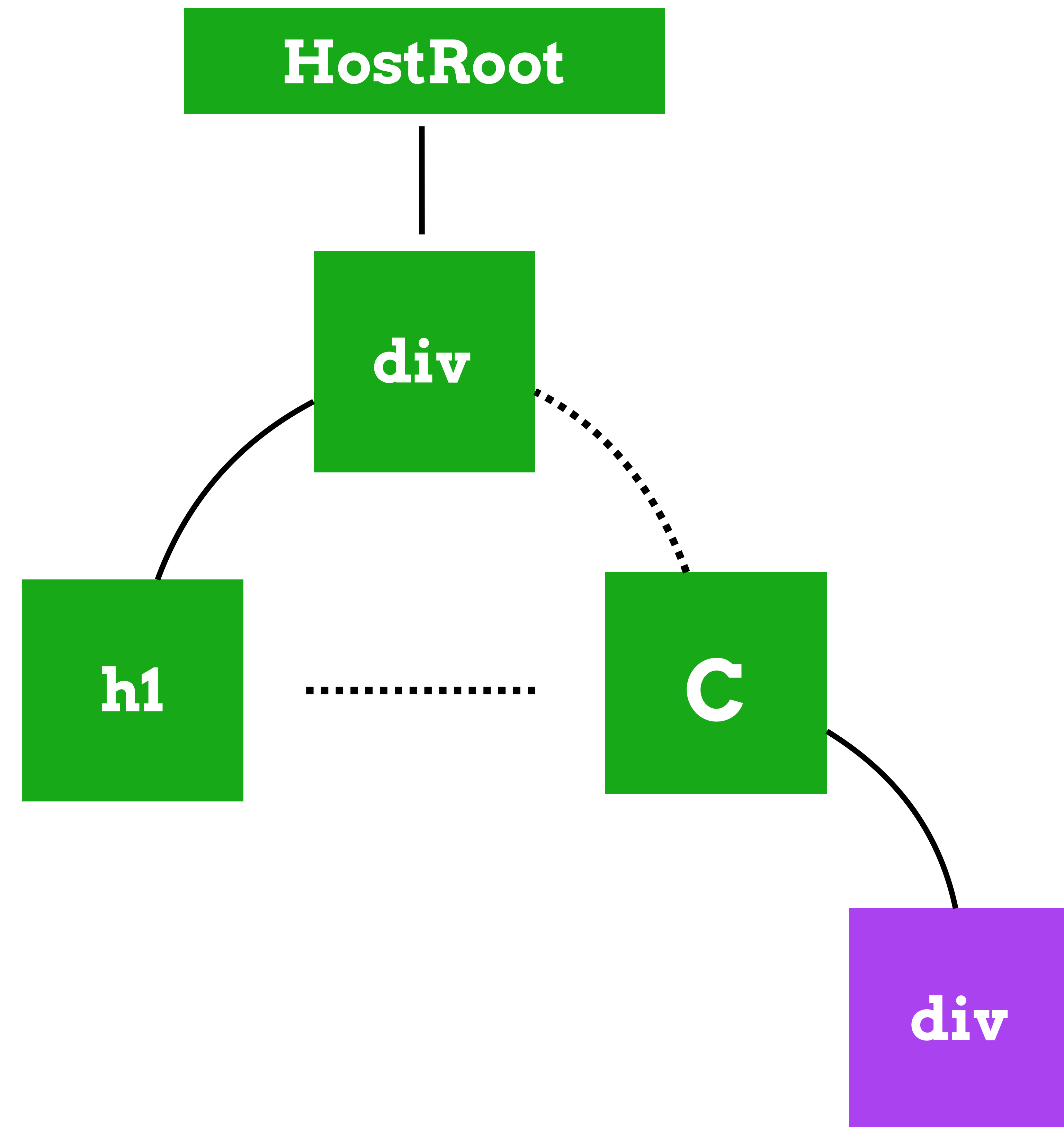
@GURLCODE



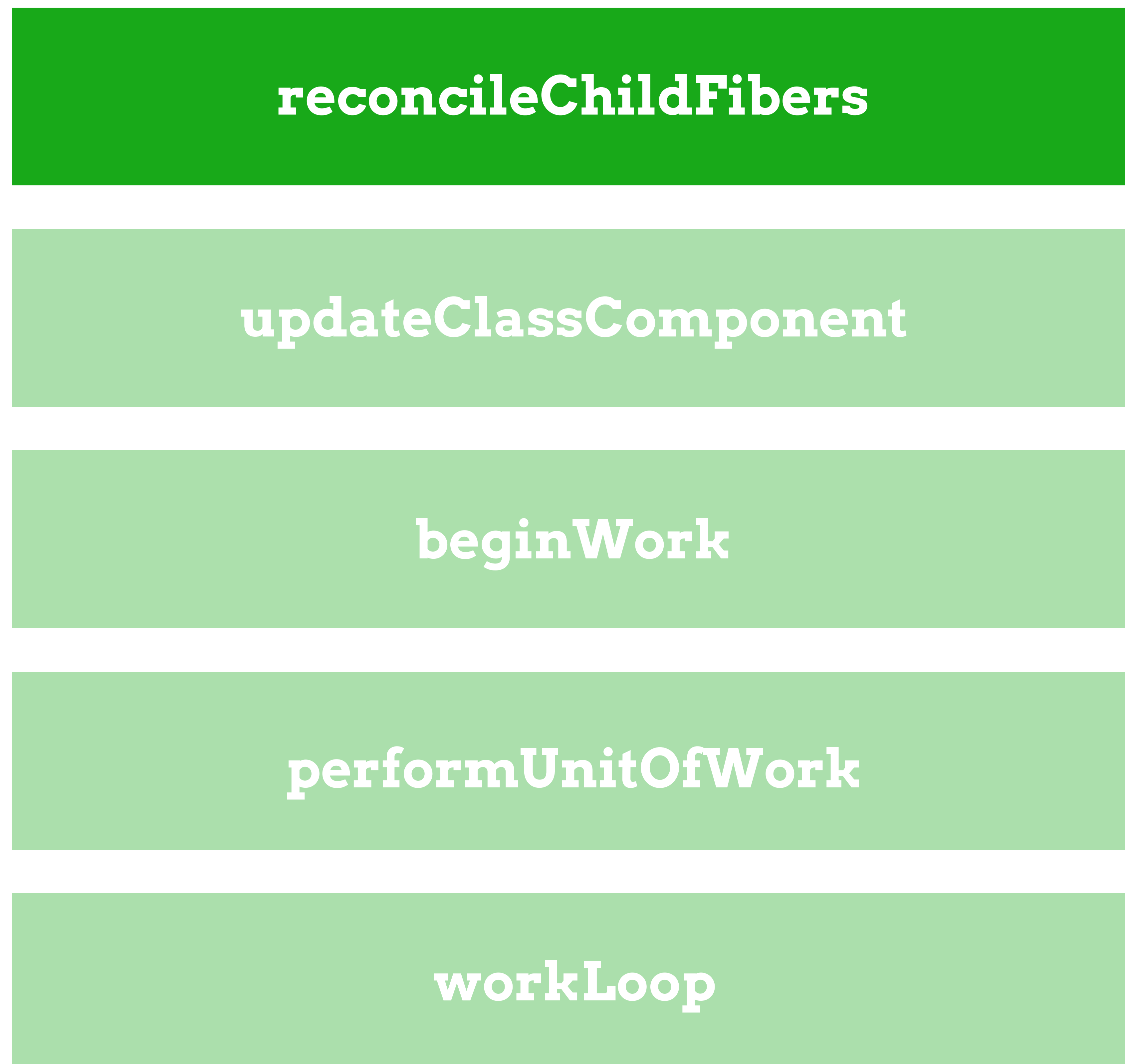


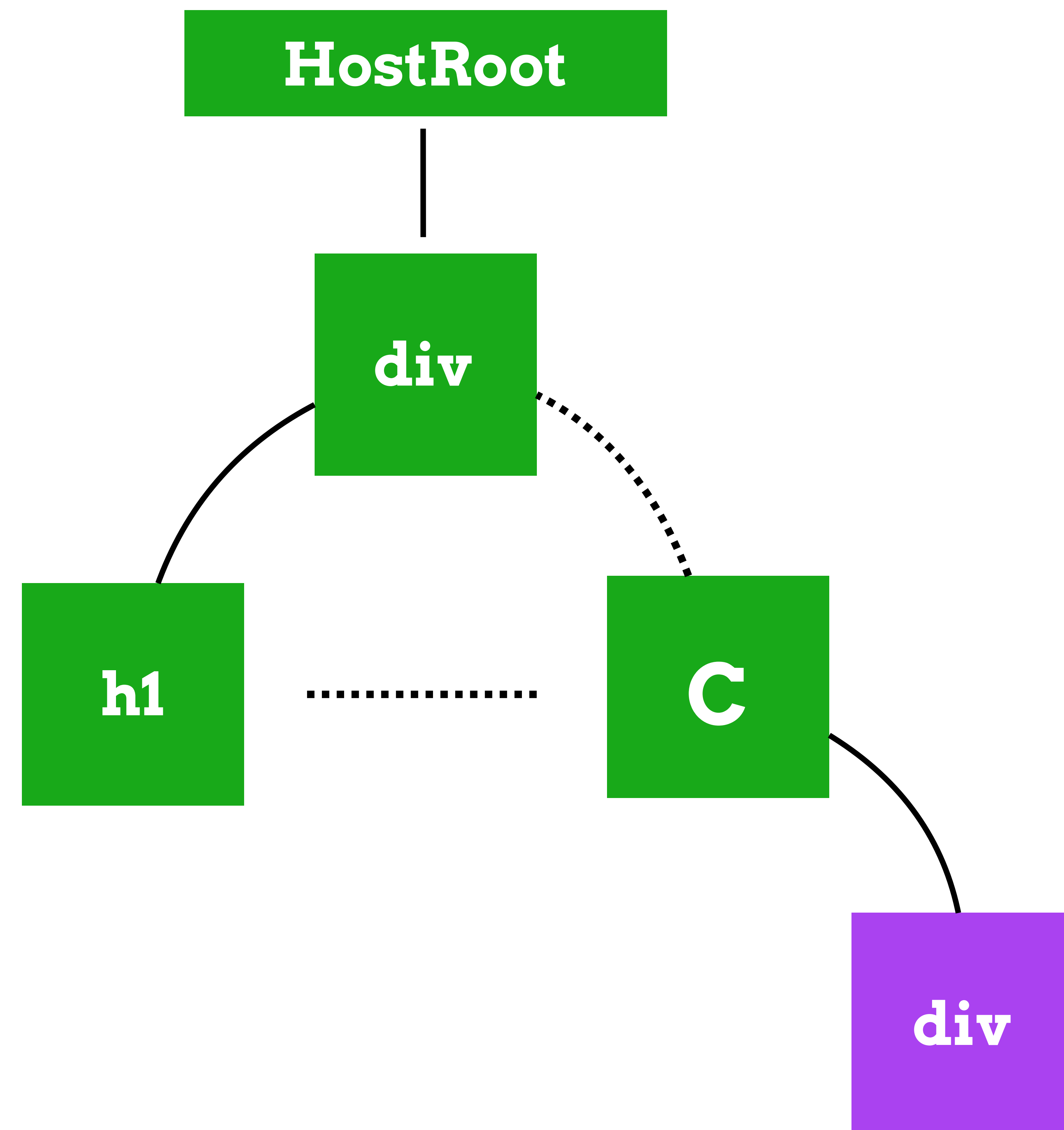
@GURLCODE



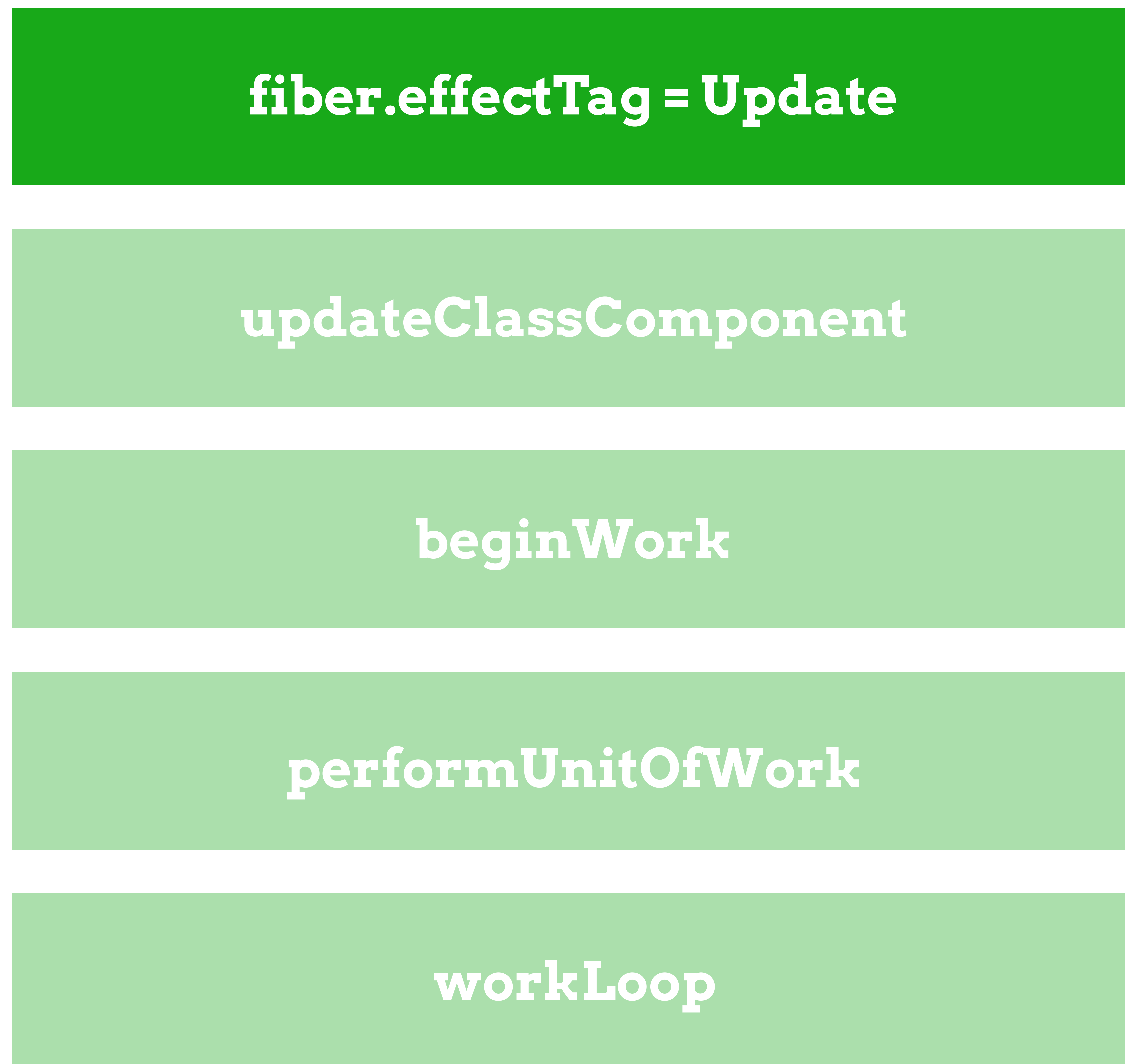


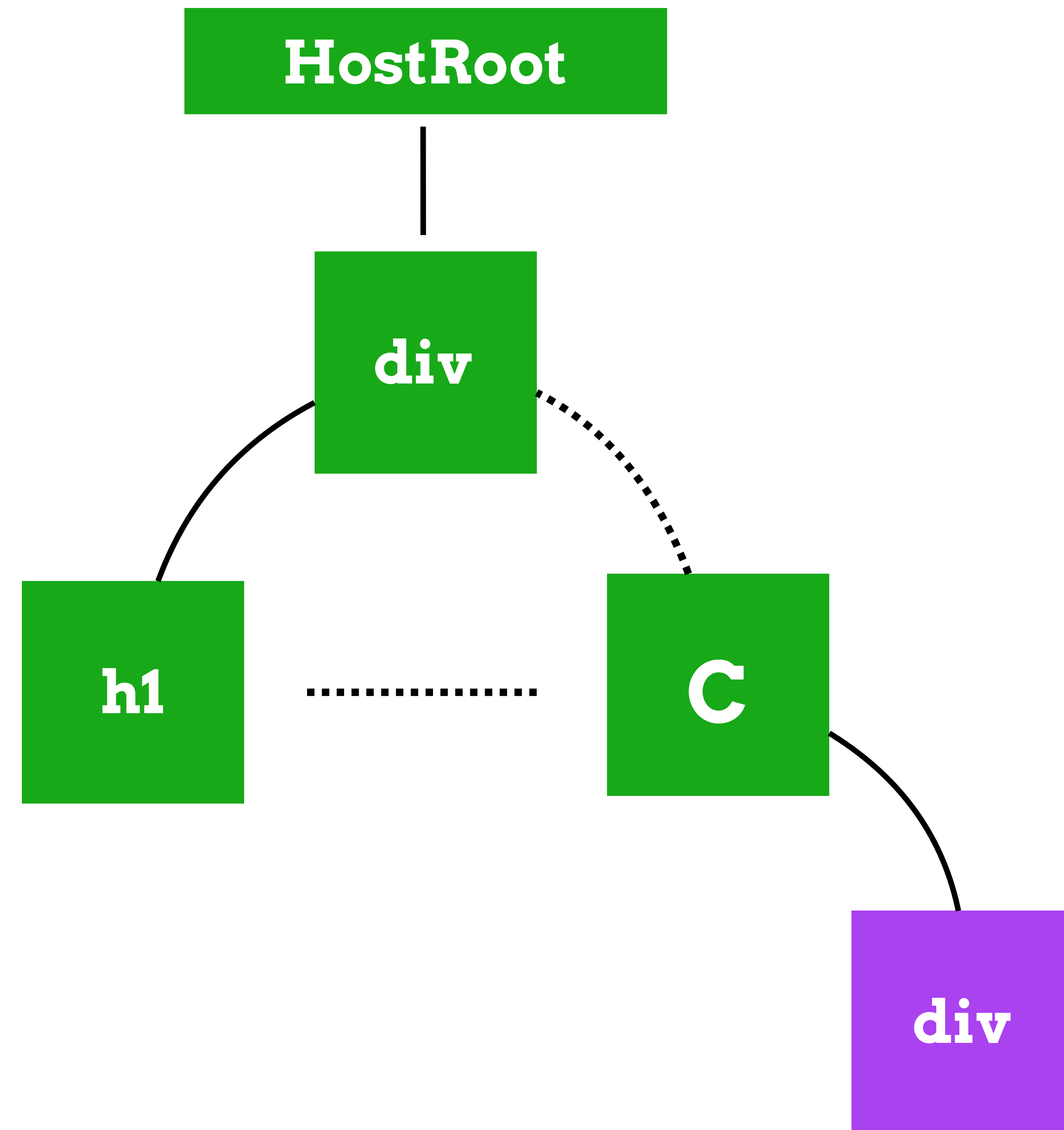
@GURLCODE





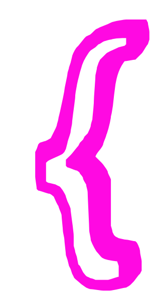
@GURLCODE





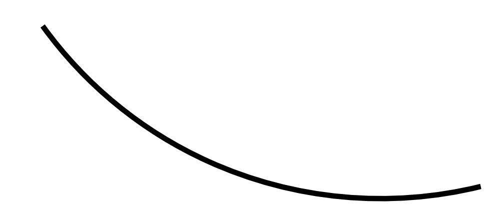
@GURLCODE

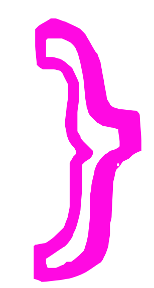
FIBER



alternate
tag
child
sibling
return
memoizedState
pendingProps
effectTag
...

work to be done





@GURLCODE

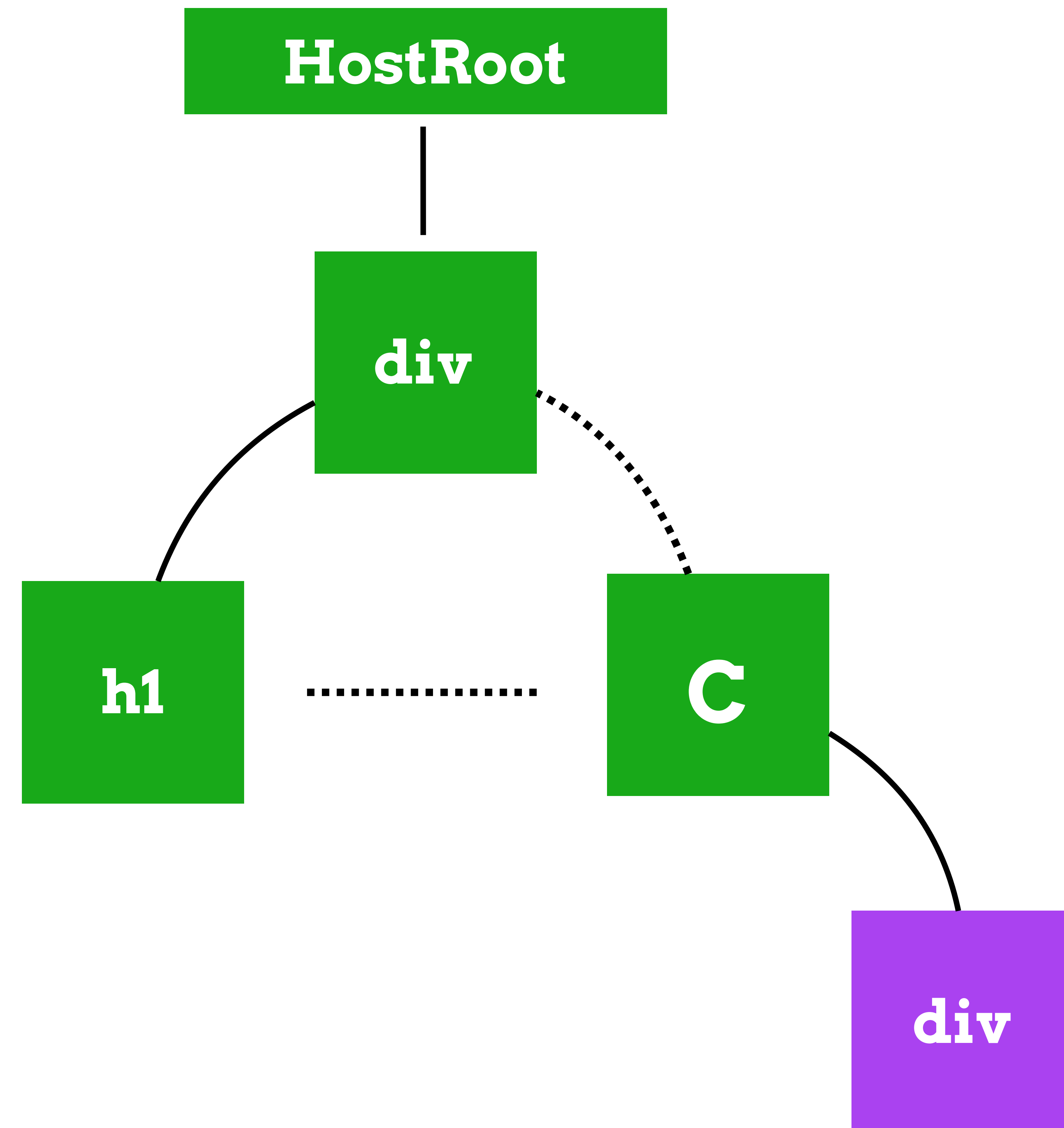
return workInProgress.child

updateClassComponent

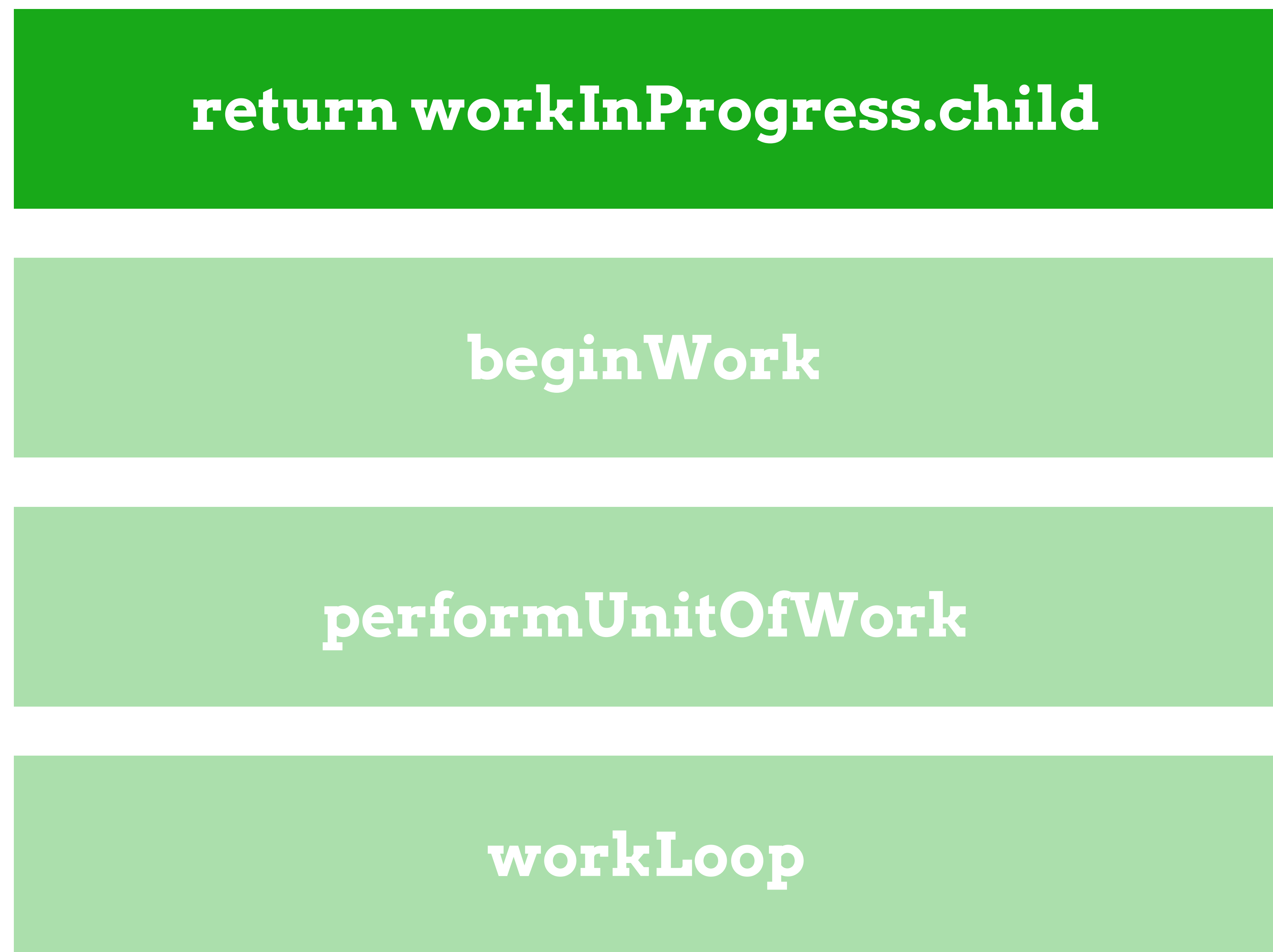
beginWork

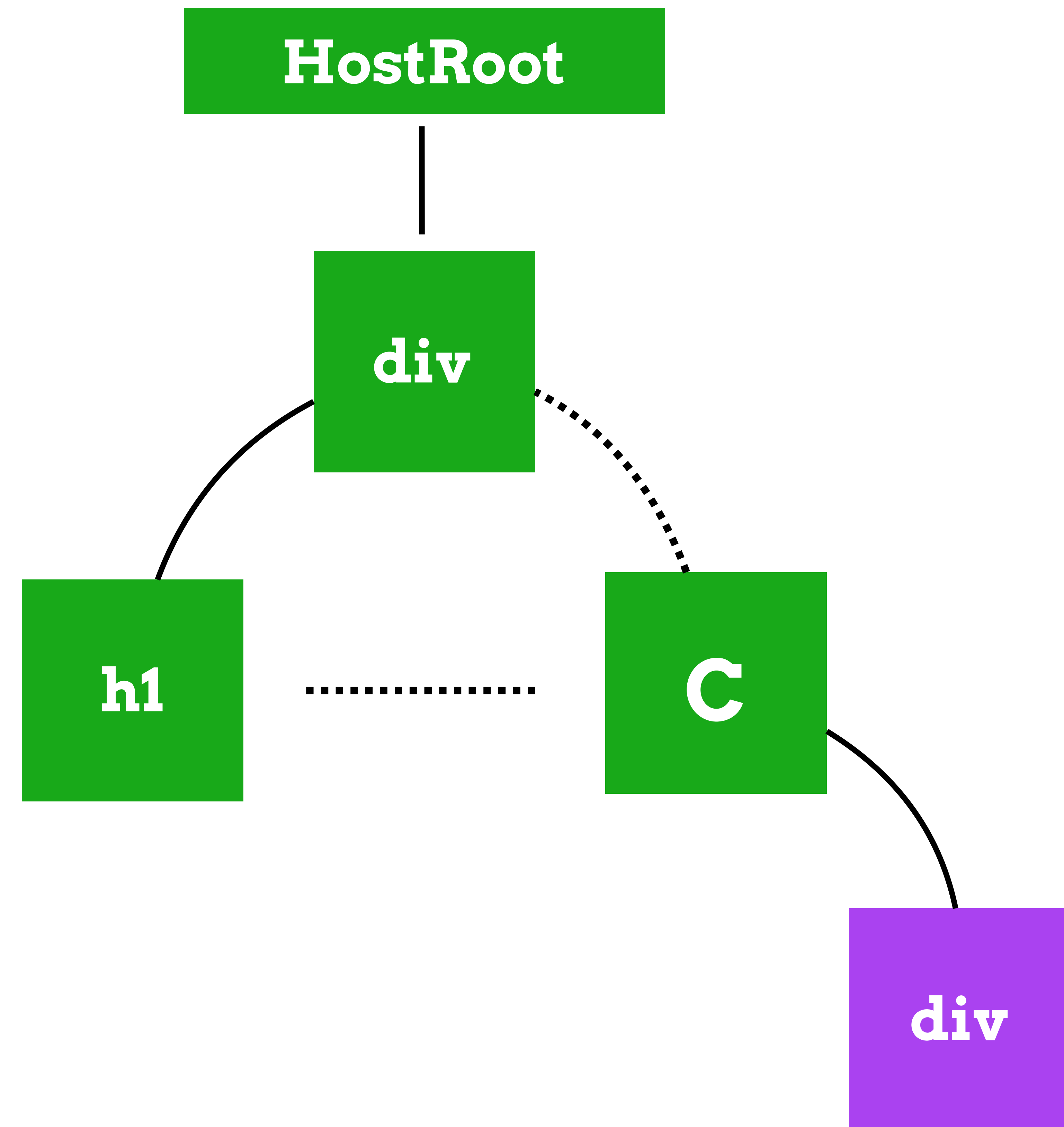
performUnitOfWork

workLoop



@GURLCODE



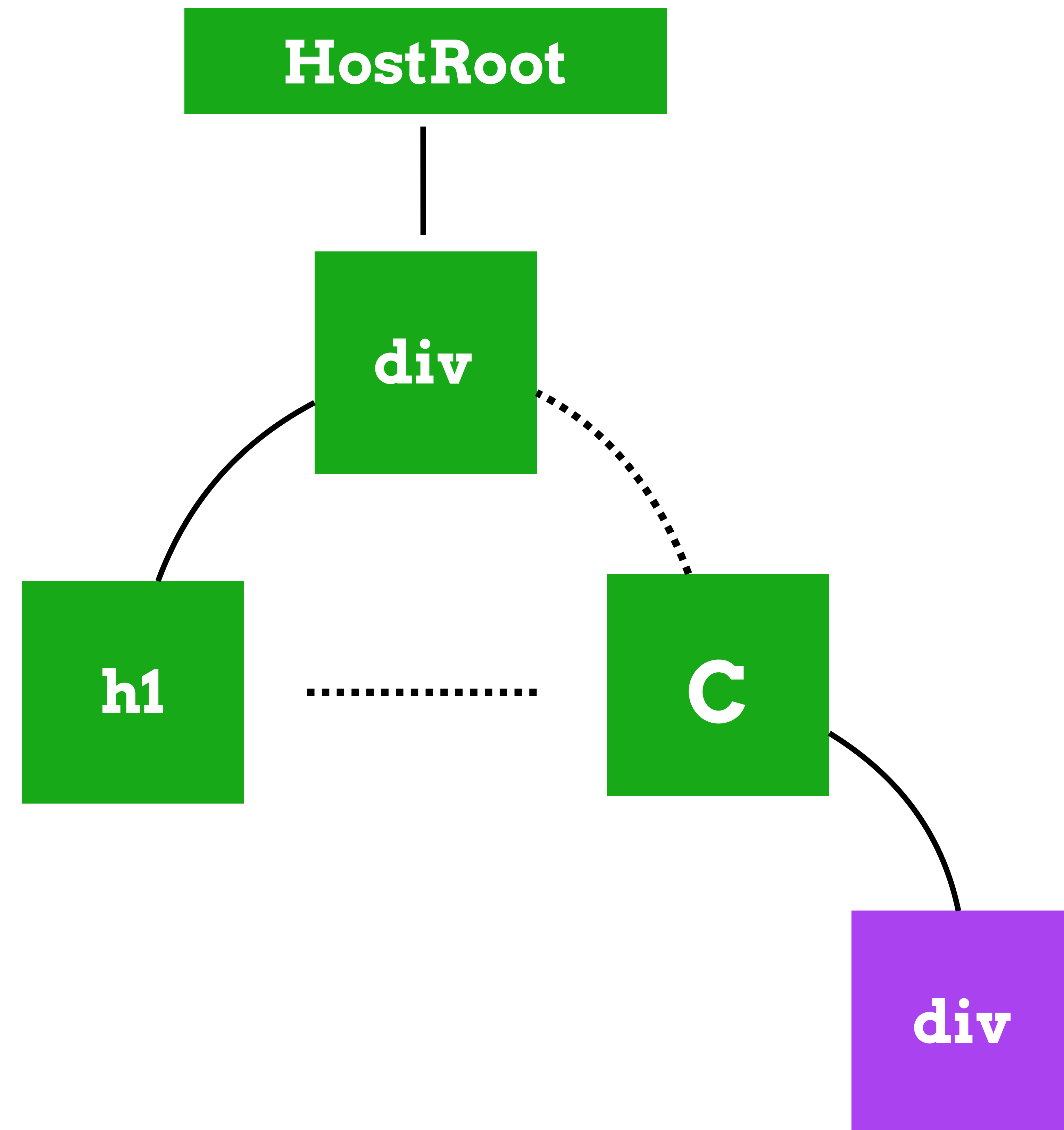


@GURLCODE

return workInProgress.child

performUnitOfWork

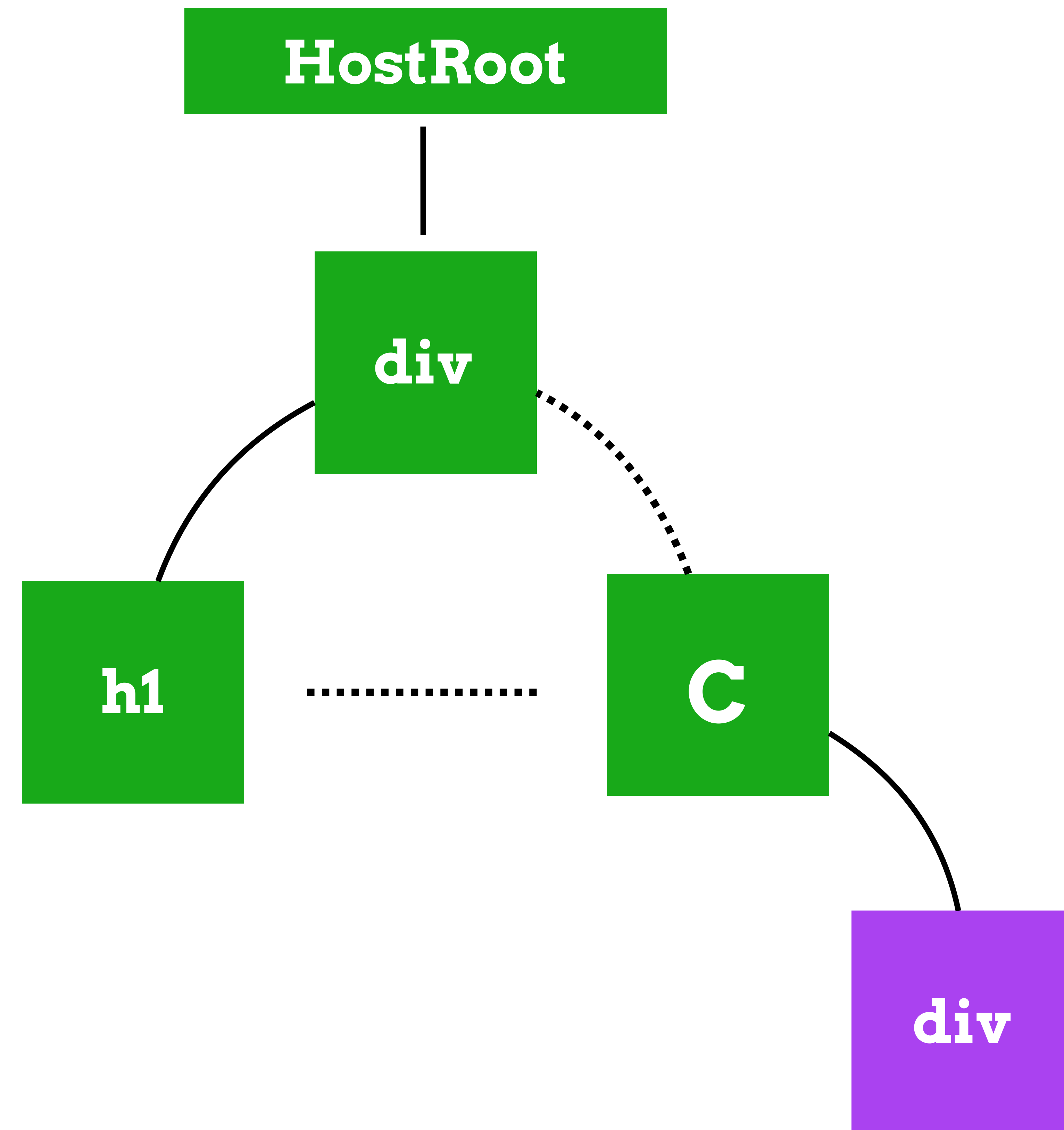
workLoop



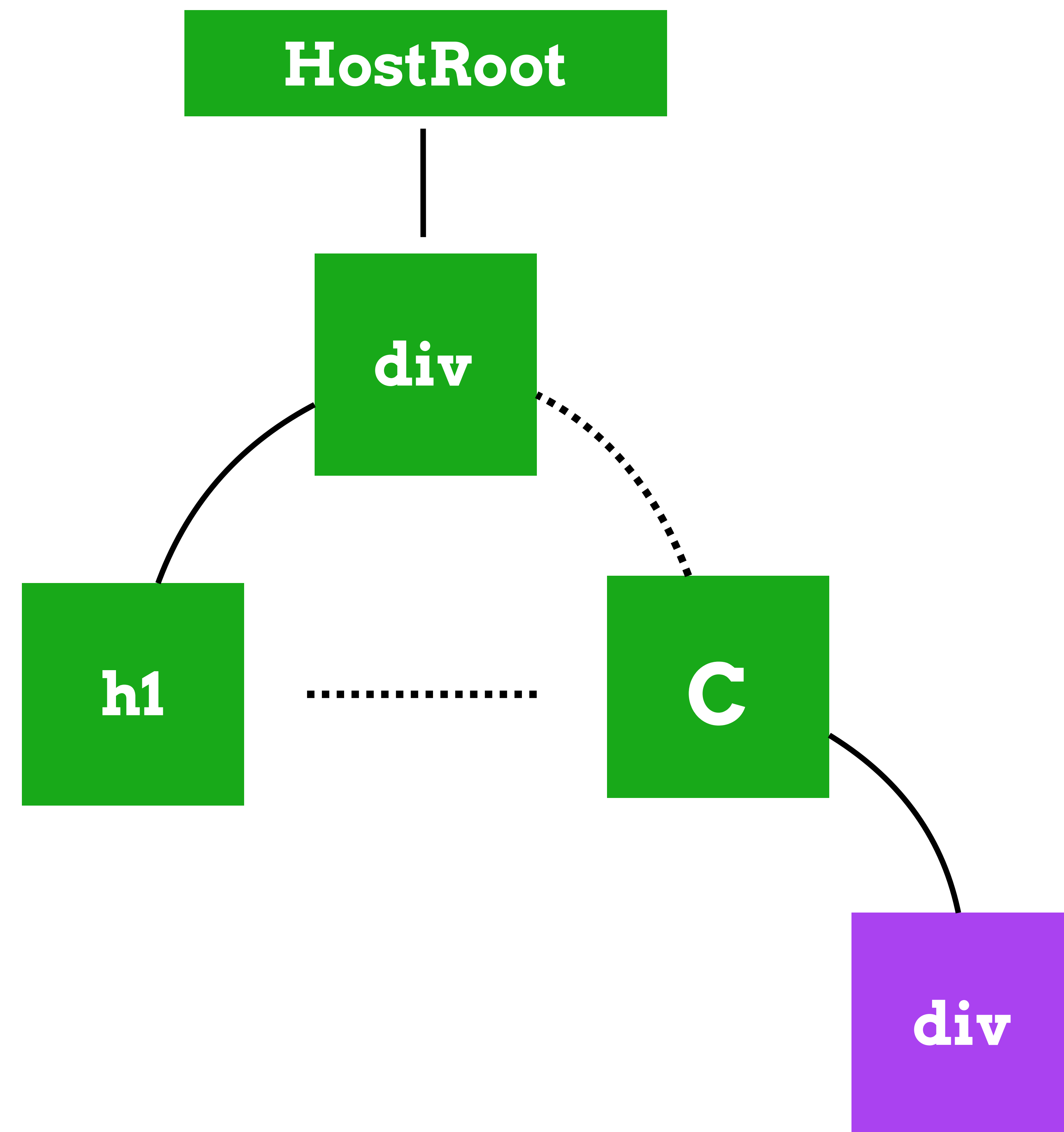
@GURLCODE

return workInProgress.child

workLoop

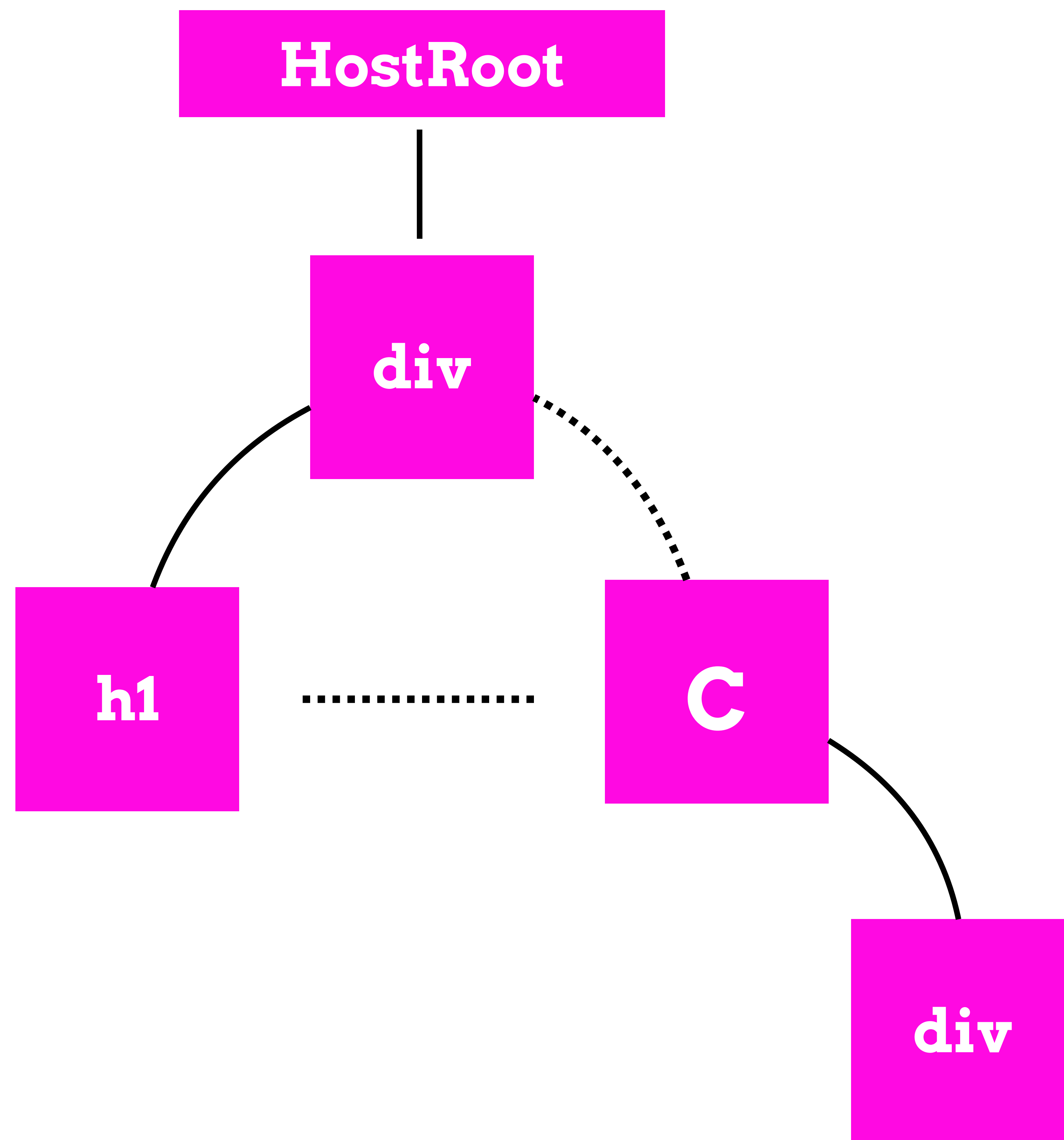


@GURLCODE

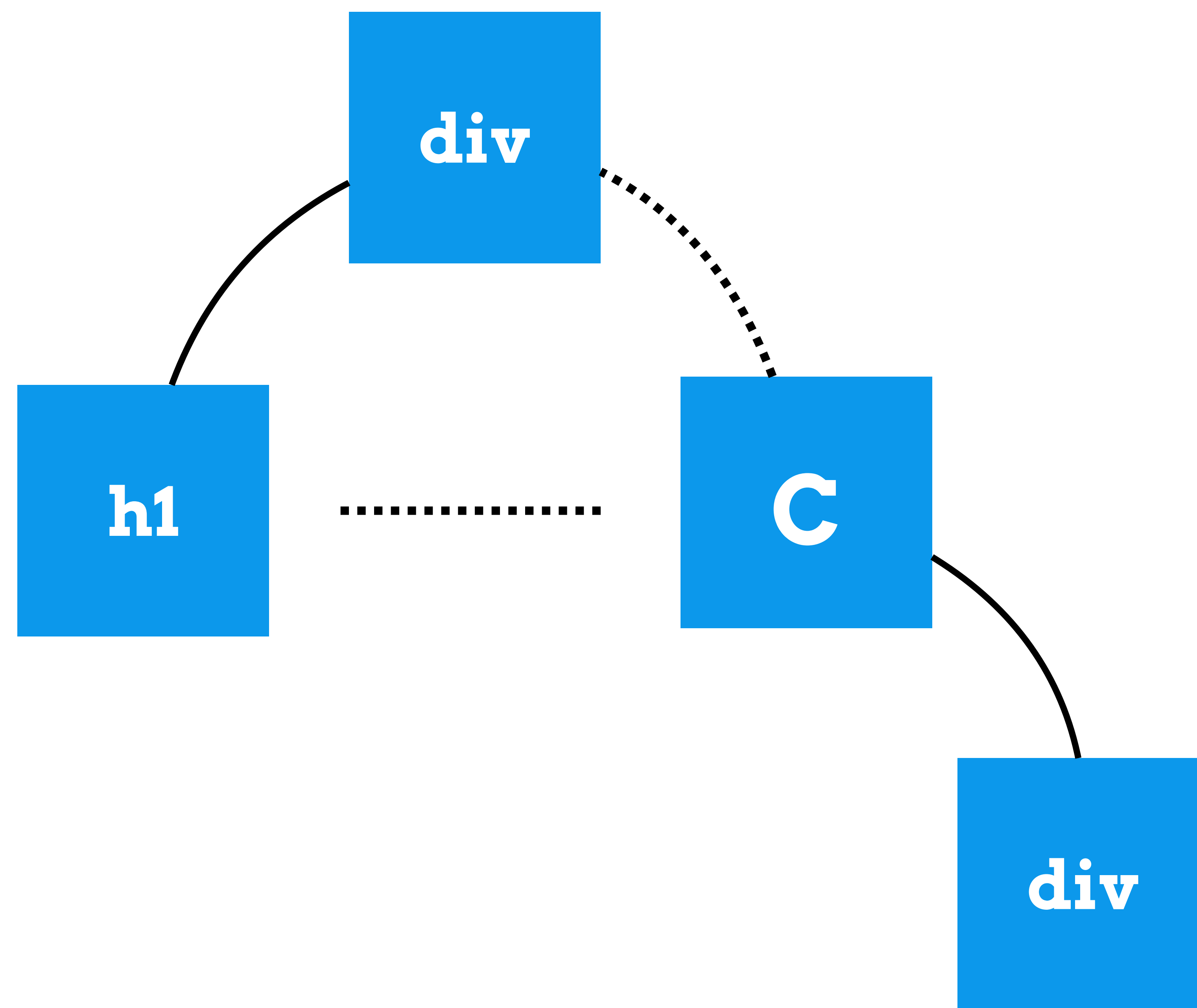


workLoop

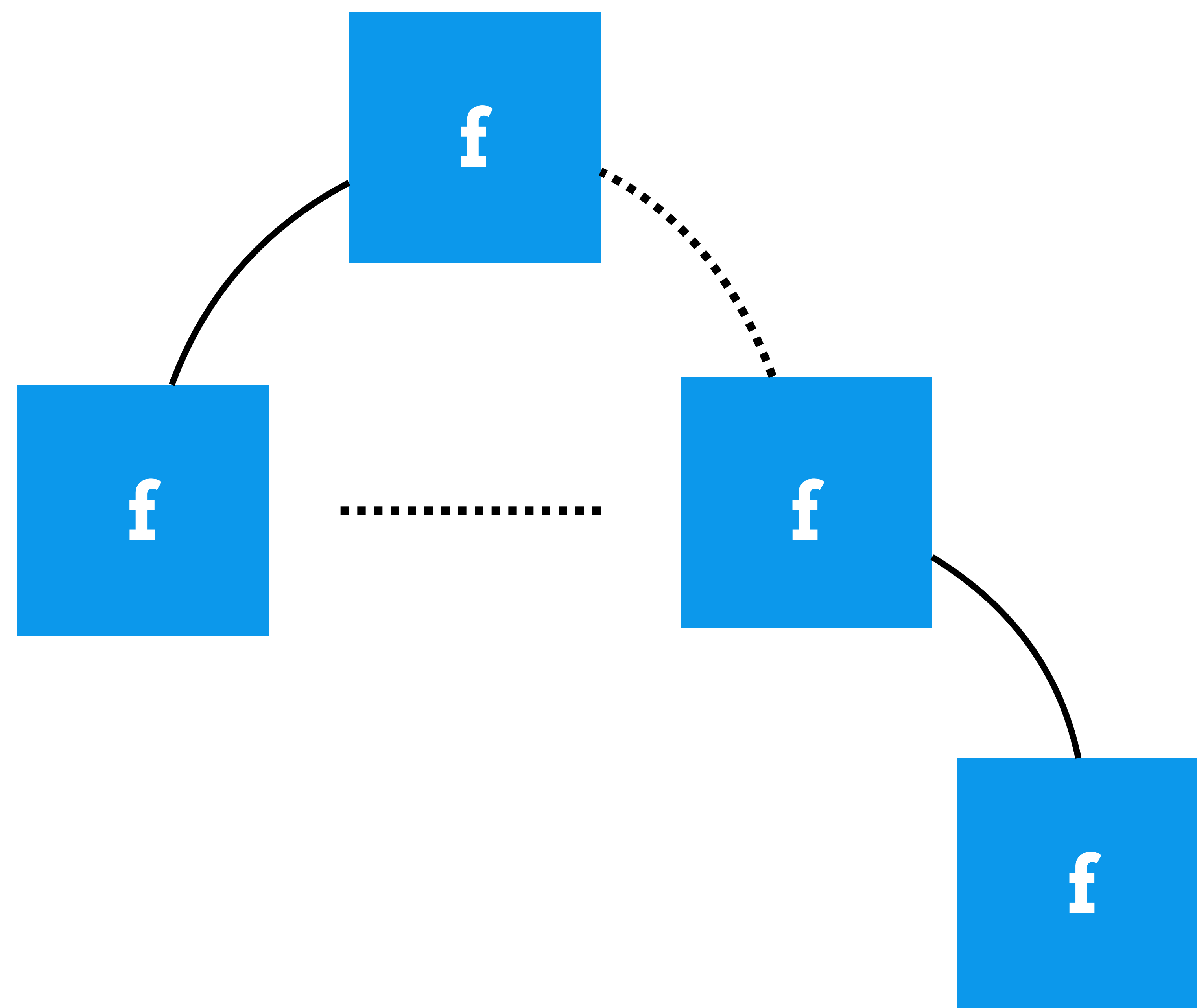
@GURLCODE



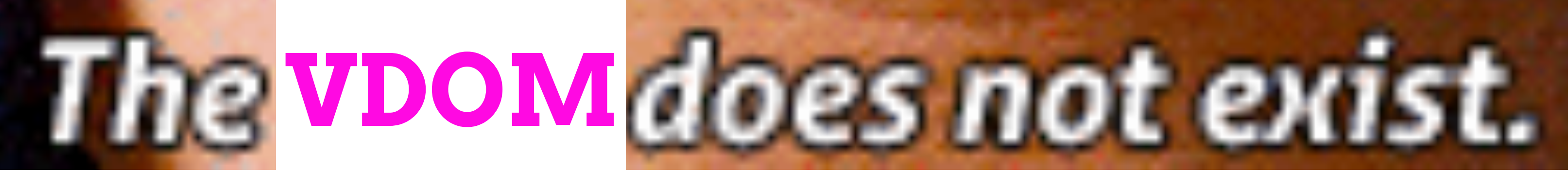
@GURLCODE



@GURLCODE



@GURLCODE



@GURLCODE

Sebastian Markbåge
@sebmarkbage

Fun fact: React Fiber doesn't have any JavaScript function recursion in its implementation because it reimplements the stack.

3:41 PM - 26 Sep 2016

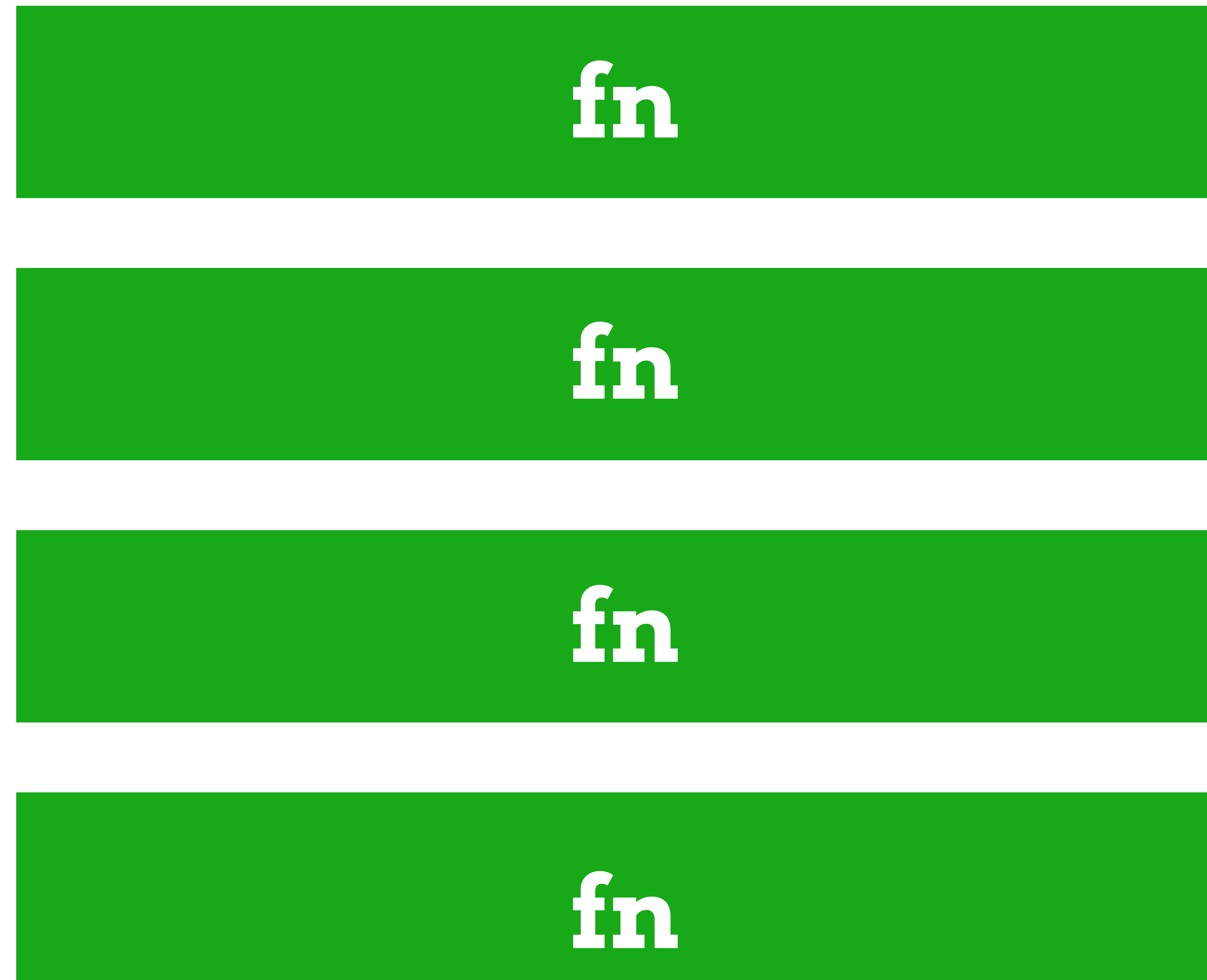
fn

fn

fn

@GURLCODE

When a function is called, a new stack frame is added



@GURLCODE

Fibers are virtual stack frames

fiber

fiber

fiber

fiber

@GURLCODE

Dan Abramov
@dan_abramov

If React is a horse then Fiber is a horse on acid.

2:03 PM - 31 Oct 2016

@GURLCODE

THANK YOU!