Cost analysis — work and span

15-150

Lecture 7: September 16, 2025

Stephanie Balzer
Carnegie Mellon University

Last week

Last week

In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.

Last week

In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.

recursive call corresponds to inductive hypothesis

Last week

In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.

recursive call corresponds to inductive hypothesis

“programs as proofs”!

Last week

In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.

recursive call corresponds to inductive hypothesis

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.
  - recursive calls give rise to recurrence

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.
  - recursive calls give rise to recurrence
  - closed form solutions of recurrences for **work**

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.
  - recursive calls give rise to recurrence
  - closed form solutions of recurrences for **work**

“programs as recurrences”!

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.
  - recursive calls give rise to recurrence
  - closed form solutions of recurrences for **work**

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.
  - recursive calls give rise to recurrence
  - closed form solutions of recurrences for **work**
- This week, we revisit work and also look at **span**

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.
  - recursive calls give rise to recurrence
  - closed form solutions of recurrences for **work**
- This week, we revisit work and also look at **span**
  - today: work (sequential evaluation), span (parallel evaluation)

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.
  - recursive calls give rise to recurrence
  - closed form solutions of recurrences for **work**
- This week, we revisit work and also look at **span**
  - today: work (sequential evaluation), span (parallel evaluation)
  - Thursday: sorting

Last week

- In week 2 we discovered (and exploited) the correspondence between programs and **proofs**.
  - recursive call corresponds to inductive hypothesis
- In week 3 we discovered (and exploited) the correspondence between programs and **asymptotic analysis**.
  - recursive calls give rise to recurrence
  - closed form solutions of recurrences for **work**
- This week, we revisit work and also look at **span**
  - today: work (sequential evaluation), span (parallel evaluation)
  - Thursday: sorting

mergesort

Work analysis: list reversal

Reversing a list

Reversing a list

```
(* rev: int list -> int list
   REQUIRES: true
   ENSURES: rev L returns the elements of L
            in reverse order.
*)
```

Reversing a list

```
(* rev: int list -> int list
   REQUIRES: true
   ENSURES: rev L returns the elements of L
            in reverse order.
*)
```

```
val [] : int list = rev []
val [4,3,2,1] : int list = rev [1,2,3,4]
```

Reversing a list

```
(* rev: int list -> int list
   REQUIRES: true
   ENSURES: rev L returns the elements of L
            in reverse order.
*)

fun rev ([] : int list) : int list =



val [] : int list = rev []
val [4,3,2,1] : int list = rev [1,2,3,4]
```

Reversing a list

```
(* rev: int list -> int list
   REQUIRES: true
   ENSURES: rev L returns the elements of L
            in reverse order.
*)

fun rev ([] : int list) : int list = []



val [] : int list = rev []
val [4,3,2,1] : int list = rev [1,2,3,4]
```

Reversing a list

```
(* rev: int list -> int list
   REQUIRES: true
   ENSURES: rev L returns the elements of L
            in reverse order.
*)

fun rev ([] : int list) : int list = []
  | rev (x::xs) =


val [] : int list = rev []
val [4,3,2,1] : int list = rev [1,2,3,4]
```

Reversing a list

```
(* rev: int list -> int list
   REQUIRES: true
   ENSURES: rev L returns the elements of L
            in reverse order.
*)

fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]


val [] : int list = rev []
val [4,3,2,1] : int list = rev [1,2,3,4]
```

Reversing a list

```
(* rev: int list -> int list
   REQUIRES: true
   ENSURES: rev L returns the elements of L
            in reverse order.
*)

fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]


val [] : int list = rev []
val [4,3,2,1] : int list = rev [1,2,3,4]
```

What is the **work** of reverse?

Work analysis for rev

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) =$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

$W_{rev}(n) =$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + ?$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + W_{@}(n-1, 1)$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + W_{@}(n-1, 1)$

Finding closed form:

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + W_{@}(n-1, 1)$

Finding closed form:

$W_{rev}(n) \leq$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n\text{-}1) + W_{@}(n\text{-}1, 1)$

Finding closed form:

$W_{rev}(n) \leq$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + W_{@}(n-1, 1)$

Recall: $W_{@}(m_1, m_2)$ is $O(m_1)$

Finding closed form:

$W_{rev}(n) \leq$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

Recall: $W_@(m_1, m_2)$ is $O(m_1)$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + W_@(n-1, 1)$

Finding closed form:

$W_{rev}(n) \leq$

Lemma: For all list values L, `length`(`rev`(L)) $\cong$ `length`(L).

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

Recall: $W_{@}(m_1, m_2)$ is $O(m_1)$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + \boxed{W_{@}(n-1, 1)}$

Finding closed form:

$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + \boxed{c_2 \cdot (n-1)}$

Lemma: For all list values L, length(rev(L)) ≅ length(L).

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

Recall: $W_{@}(m_1, m_2)$ is $O(m_1)$

$W_{rev}(n) = c_1 + W_{rev}(n-1) + W_{@}(n-1, 1)$

Finding closed form:

$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$

$W_{rev}(n) \leq$

Lemma: For all list values L, `length`(`rev`(L)) $\cong$ `length`(L).

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Recurrence relation:

$W_{rev}(0) = c_0$

Recall: $W_{@}(m_1, m_2)$ is $O(m_1)$

$W_{rev}(n) = c_1 + W_{rev}(n\text{-}1) + W_{@}(n\text{-}1, 1)$

Finding closed form:

$W_{rev}(n) \leq c_1 + W_{rev}(n\text{-}1) + c_2 \cdot (n\text{-}1)$

$W_{rev}(n) \leq c_1 + c_2 \cdot n + W_{rev}(n\text{-}1)$

Lemma: For all list values L, `length`(`rev`(L)) ≅ `length`(L).

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$

$W_{rev}(n) \leq c_1 + c_2 \cdot n + W_{rev}(n-1)$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$W_{rev}(n) \leq c_1 + c_2 \cdot n + W_{rev}(n-1)$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$W_{rev}(n) \leq c_1 + W_{rev}(n\text{-}1) + c_2 \cdot (n\text{-}1)$

$W_{rev}(n) \leq c_1 + c_2 \cdot n + W_{rev}(n\text{-}1)$

$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n\text{-}1) + W_{rev}(n\text{-}2))$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$W_{rev}(n) \leq c_1 + c_2 \cdot n + W_{rev}(n-1)$$

$$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2))$$

$$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3)))$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n\text{-}1) + c_2 \cdot (n\text{-}1)$$

$$W_{rev}(n) \leq c_1 + c_2 \cdot n + W_{rev}(n\text{-}1)$$

$$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n\text{-}1) + W_{rev}(n\text{-}2))$$

$$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n\text{-}1) + (c_1 + c_2 \cdot (n\text{-}2) + W_{rev}(n\text{-}3)))$$

$$\vdots$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$\begin{aligned} W_{rev}(n) &\leq c_1 + c_2 \cdot n + W_{rev}(n-1) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2)) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3))) \\ &\vdots \end{aligned}$$

$$W_{rev}(n) =$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \le c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$\begin{aligned} W_{rev}(n) &\le c_1 + c_2 \cdot n + W_{rev}(n-1) \\ &\le c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2)) \\ &\le c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3))) \\ &\vdots \end{aligned}$$

$$W_{rev}(n) =$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$\begin{aligned} W_{rev}(n) &\leq c_1 + c_2 \cdot n + W_{rev}(n-1) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2)) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3))) \\ &\vdots \end{aligned}$$

$$W_{rev}(n) =$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$\begin{aligned} W_{rev}(n) &\leq c_1 + c_2 \cdot n + W_{rev}(n-1) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2)) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3))) \\ &\vdots \end{aligned}$$

$$W_{rev}(n) =$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \le c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$\begin{aligned}
W_{rev}(n) &\le c_1 + c_2 \cdot n + W_{rev}(n-1) \\
&\le c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2)) \\
&\le c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3))) \\
&\vdots \\
W_{rev}(n) &= c_0 + n \cdot c_1 + 1/2 \cdot n \cdot (n+1) \cdot c_2
\end{aligned}$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$\begin{aligned}
W_{rev}(n) &\leq c_1 + c_2 \cdot n + W_{rev}(n-1) \\
&\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2)) \\
&\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3))) \\
&\vdots
\end{aligned}$$

$$W_{rev}(n) = c_0 + n \cdot c_1 + 1/2 \cdot n \cdot (n+1) \cdot c_2$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$\begin{aligned} W_{rev}(n) &\leq c_1 + c_2 \cdot n + W_{rev}(n-1) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2)) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3))) \\ &\vdots \\ W_{rev}(n) &= c_0 + n \cdot c_1 + 1/2 \cdot n \cdot (n+1) \cdot c_2 \end{aligned}$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$W_{rev}(n) \leq c_1 + c_2 \cdot n + W_{rev}(n-1)$$

$$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2))$$

$$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3)))$$

$$\vdots$$

$$W_{rev}(n) = c_0 + n \cdot c_1 + 1/2 \cdot n \cdot (n+1) \cdot c_2$$

Sum of first n positive integers is $1/2 \cdot n \cdot (n+1)$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n\text{-}1) + c_2\cdot(n\text{-}1)$$

$$\begin{aligned}
W_{rev}(n) &\leq c_1 + c_2\cdot n + W_{rev}(n\text{-}1) \\
&\leq c_1 + c_2\cdot n + (c_1 + c_2\cdot(n\text{-}1) + W_{rev}(n\text{-}2)) \\
&\leq c_1 + c_2\cdot n + (c_1 + c_2\cdot(n\text{-}1) + (c_1 + c_2\cdot(n\text{-}2) + W_{rev}(n\text{-}3))) \\
&\vdots \\
W_{rev}(n) &= c_0 + n\cdot c_1 + 1/2\cdot n\cdot(n+1)\cdot c_2
\end{aligned}$$

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n\text{-}1) + c_2 \cdot (n\text{-}1)$$

$$W_{rev}(n) \leq c_1 + c_2 \cdot n + W_{rev}(n\text{-}1)$$

$$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n\text{-}1) + W_{rev}(n\text{-}2))$$

$$\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n\text{-}1) + (c_1 + c_2 \cdot (n\text{-}2) + W_{rev}(n\text{-}3)))$$

$$\vdots$$

$$W_{rev}(n) = c_0 + n \cdot c_1 + 1/2 \cdot n \cdot (n+1) \cdot c_2$$

Consequently: $W_{rev}(n)$ is $O(n^2)$.

Work analysis for rev

```
fun rev ([] : int list) : int list = []
  | rev (x::xs) = (rev xs) @ [x]
```

Work: $W_{rev}(n)$ with n the list length.

Finding closed form:

$$W_{rev}(n) \leq c_1 + W_{rev}(n-1) + c_2 \cdot (n-1)$$

$$\begin{aligned} W_{rev}(n) &\leq c_1 + c_2 \cdot n + W_{rev}(n-1) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + W_{rev}(n-2)) \\ &\leq c_1 + c_2 \cdot n + (c_1 + c_2 \cdot (n-1) + (c_1 + c_2 \cdot (n-2) + W_{rev}(n-3))) \\ &\vdots \end{aligned}$$

$$W_{rev}(n) = c_0 + n \cdot c_1 + 1/2 \cdot n \cdot (n+1) \cdot c_2$$

Consequently: $W_{rev}(n)$ is $O(n^2)$.

Can we do better?

Reversing a list: can we do better?

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:
*)
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:
*)
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:
*)
```

accumulator

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:
*)
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:  trev(L, acc) ==
*)
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:  trev(L, acc) == (rev L) @ acc
*)
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:  trev(L, acc) == (rev L) @ acc
*)


fun trev ([] : int list, acc : int list) : int list =
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:  trev(L, acc) == (rev L) @ acc
*)


fun trev ([] : int list, acc : int list) : int list = acc
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:  trev(L, acc) == (rev L) @ acc
*)


fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) =
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:  trev(L, acc) == (rev L) @ acc
*)


fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:  trev(L, acc) == (rev L) @ acc
*)


fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)

fun reverse (L : int list) : int list = trev(L, [])
```

Reversing a list: can we do better?

```
(* trev : int list * int list -> int list
   REQUIRES: true
   ENSURES:  trev(L, acc) == (rev L) @ acc
*)


fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)

fun reverse (L : int list) : int list = trev(L, [])
```

Before determining work for `trev`, are `rev` and `trev` extensional equivalent?

Are rev and trev extensional equivalent?

Are rev and trev extensional equivalent?

Theorem: For all values `L: int list` and `acc: int list`, `trev(L,acc)` ≅ `(rev L) @ acc`.

Are rev and trev extensional equivalent?

Theorem: For all values `L: int list` and `acc: int list`,
`trev(L,acc)` ≅ `(rev L) @ acc`.

Prove this theorem as an exercise!

Are rev and trev extensional equivalent?

Theorem: For all values `L: int list` and `acc: int list`,
`trev(L,acc)` ≅ `(rev L) @ acc`.

- Prove this theorem as an exercise!
- We provide the solution in the notes (rev.pdf). But try first!

Work analysis for trev

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with n the list length and m the accumulator length.

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{trev}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$W_{\texttt{trev}}(0, m) =$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$W_{\texttt{trev}}(0, m) = c_0$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$W_{\texttt{trev}}(0, m) = c_0$

$W_{\texttt{trev}}(n, m) =$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\mathtt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$W_{\mathtt{trev}}(0, m) = c_0$

$W_{\mathtt{trev}}(n, m) = c_1 + W_{\mathtt{trev}}(n-1, m+1)$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$W_{\texttt{trev}}(0, m) = c_0$

$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n-1, m+1)$

Finding closed form:

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$$W_{\texttt{trev}}(0, m) = c_0$$

$$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n\text{-}1, m+1)$$

Finding closed form:

$$W_{\texttt{trev}}(n, m) \leq c_1 + W_{\texttt{trev}}(n\text{-}1, m+1)$$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with **n** the list length and **m** the accumulator length.

Recurrence relation:

$W_{\texttt{trev}}(0, m) = c_0$

$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n-1, m+1)$

Finding closed form:

$$
\begin{aligned}
W_{\texttt{trev}}(n, m) &\leq c_1 + W_{\texttt{trev}}(n-1, m+1) \\
&\leq c_1 + (c_1 + W_{\texttt{trev}}(n-2, m+2))
\end{aligned}
$$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with **n** the list length and **m** the accumulator length.

Recurrence relation:

$W_{\texttt{trev}}(0, m) = c_0$

$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n-1, m+1)$

Finding closed form:

$$
\begin{aligned}
W_{\texttt{trev}}(n, m) &\le c_1 + W_{\texttt{trev}}(n-1, m+1) \\
&\le c_1 + (c_1 + W_{\texttt{trev}}(n-2, m+2))
\end{aligned}
$$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with **n** the list length and **m** the accumulator length.

Recurrence relation:

$$W_{\texttt{trev}}(0, m) = c_0$$

$$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n\text{-}1, m\text{+}1)$$

Finding closed form:

$$\begin{aligned} W_{\texttt{trev}}(n, m) &\leq c_1 + W_{\texttt{trev}}(n\text{-}1, m\text{+}1) \\ &\leq c_1 + (c_1 + W_{\texttt{trev}}(n\text{-}2, m\text{+}2)) \\ &\vdots \end{aligned}$$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with **n** the list length and **m** the accumulator length.

Recurrence relation:

$$W_{\texttt{trev}}(0, m) = c_0$$

$$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n-1, m+1)$$

Finding closed form:

$$\begin{aligned} W_{\texttt{trev}}(n, m) &\leq c_1 + W_{\texttt{trev}}(n-1, m+1) \\ &\leq c_1 + (c_1 + W_{\texttt{trev}}(n-2, m+2)) \\ &\vdots \end{aligned}$$

$$W_{\texttt{trev}}(n, m)$$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with **n** the list length and **m** the accumulator length.

Recurrence relation:

$$W_{\texttt{trev}}(0, m) = c_0$$

$$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n-1, m+1)$$

Finding closed form:

$$\begin{aligned}W_{\texttt{trev}}(n, m) &\leq c_1 + W_{\texttt{trev}}(n-1, m+1)\\ &\leq c_1 + (c_1 + W_{\texttt{trev}}(n-2, m+2))\\ &\vdots\end{aligned}$$

$$W_{\texttt{trev}}(n, m) = c_0 + n \cdot c_1$$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$W_{\texttt{trev}}(0, m) = c_0$

$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n\text{-}1, m\text{+}1)$

Finding closed form:

$W_{\texttt{trev}}(n, m) = c_0 + n \cdot c_1$

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\mathtt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$$W_{\mathtt{trev}}(0, m) = c_0$$

$$W_{\mathtt{trev}}(n, m) = c_1 + W_{\mathtt{trev}}(n\text{-}1, m\text{+}1)$$

Finding closed form:

$$W_{\mathtt{trev}}(n, m) = c_0 + n \cdot c_1$$

Consequently: $W_{\mathtt{trev}}(n)$ is O(n).

Work analysis for trev

```
fun trev ([] : int list, acc : int list) : int list = acc
  | trev (x::xs, acc) = trev(xs, x::acc)
```

Work: $W_{\texttt{trev}}(n, m)$ with n the list length and m the accumulator length.

Recurrence relation:

$$W_{\texttt{trev}}(0, m) = c_0$$

$$W_{\texttt{trev}}(n, m) = c_1 + W_{\texttt{trev}}(n-1, m+1)$$

Finding closed form:

$$W_{\texttt{trev}}(n, m) = c_0 + n \cdot c_1$$

Consequently: $W_{\texttt{trev}}(n)$ is $O(n)$.

Note: Using the recurrence relation we can prove the closed form by induction on n.

Work and span analysis: tree sum

Summing integers in a tree

Summing integers in a tree

```
datatype tree = Empty | Node of tree * int * tree
```

Summing integers in a tree

```
datatype tree = Empty | Node of tree * int * tree

(*
   sum : tree -> int
   REQUIRES: true
   ENSURES: sum(T) evaluates to the sum of
            all the integers in T.
*)
```

Summing integers in a tree

```
datatype tree = Empty | Node of tree * int * tree

(*
   sum : tree -> int
   REQUIRES: true
   ENSURES: sum(T) evaluates to the sum of
            all the integers in T.
*)

fun sum (Empty : tree) : int =
  |
```

Summing integers in a tree

```
datatype tree = Empty | Node of tree * int * tree

(*
   sum : tree -> int
   REQUIRES: true
   ENSURES: sum(T) evaluates to the sum of
            all the integers in T.
*)

fun sum (Empty : tree) : int = 0
  |
```

Summing integers in a tree

```
datatype tree = Empty | Node of tree * int * tree

(*
   sum : tree -> int
   REQUIRES: true
   ENSURES: sum(T) evaluates to the sum of
            all the integers in T.
*)

fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) =
```

Summing integers in a tree

```
datatype tree = Empty | Node of tree * int * tree

(*
   sum : tree -> int
   REQUIRES: true
   ENSURES: sum(T) evaluates to the sum of
            all the integers in T.
*)

fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work analysis for sum

Work analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work: $W_{sum}(n)$ with n the number of integers in a tree t.

Work analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work: $W_{sum}(n)$ with n the number of integers in a tree t.

Recurrence relation:

Work analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work: $W_{sum}(n)$ with n the number of integers in a tree t.

Recurrence relation:

$W_{sum}(0) = c_0$

Work analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work: $W_{sum}(n)$ with n the number of integers in a tree t.

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) =$

Work analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work: $W_{sum}(n)$ with n the number of integers in a tree t.

Recurrence relation:

$$W_{sum}(0) = c_0$$

$$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$$

Work analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work: $W_{sum}(n)$ with n the number of integers in a tree t.

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

number of integers in left subtree

Work analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Work: $W_{sum}(n)$ with n the number of integers in a tree t.

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

number of integers in left subtree

number of integers in right subtree

Work analysis for sum

Recurrence relation:

$$W_{sum}(0) = c_0$$

$$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$$

Work analysis for sum

Recurrence relation:

$$W_{sum}(0) = c_0$$

$$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$$

To find a closed form, let's employ the "**tree method**".

Work analysis for sum

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

To find a closed form, let's employ the "**tree method**".

visualize work at each node/leaf:

Work analysis for sum

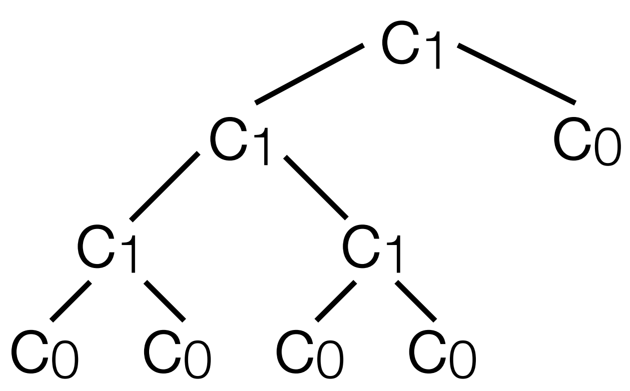
Recurrence relation:

$$W_{sum}(0) = c_0$$

$$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$$

To find a closed form, let's employ the "**tree method**".

visualize work at each node/leaf:



Work analysis for sum

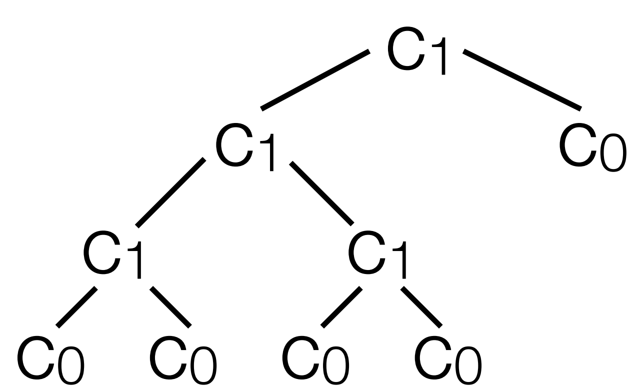
Recurrence relation:

$$W_{sum}(0) = c_0$$

$$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$$

To find a closed form, let's employ the "**tree method**".

visualize work at each node/leaf:



Closed form:

Work analysis for sum

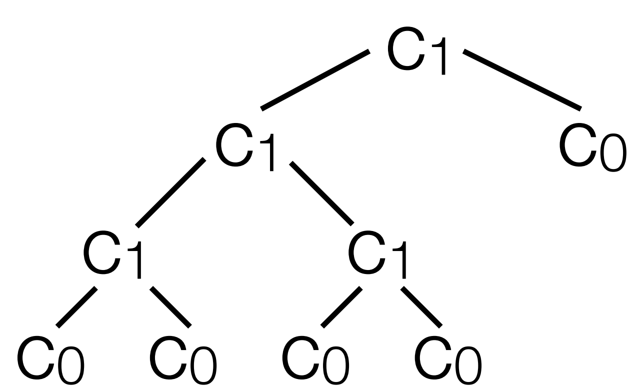
Recurrence relation:

$$W_{sum}(0) = c_0$$

$$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$$

To find a closed form, let's employ the "**tree method**".

visualize work at each node/leaf:



Closed form: $W_{sum}(n) = n \cdot c1 + (n+1) \cdot c_0$

Work analysis for sum

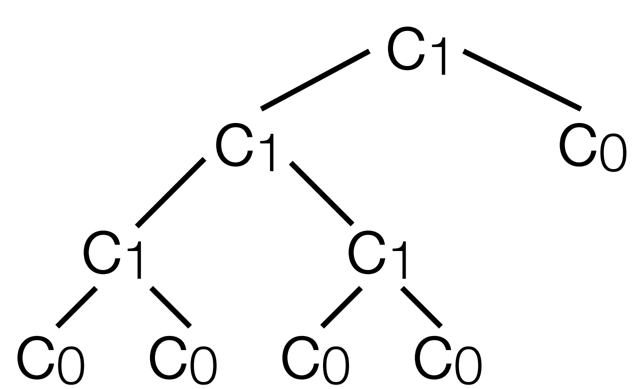
Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

To find a closed form, let's employ the "**tree method**".

visualize work at each node/leaf:



Closed form: $W_{sum}(n) = n \cdot c1 + (n+1) \cdot c_0$

Work analysis for sum

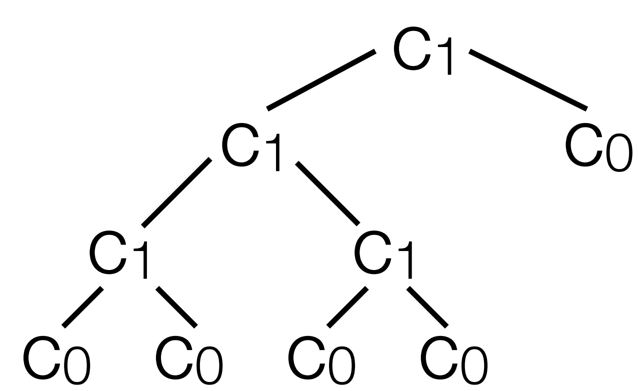
Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

To find a closed form, let's employ the "**tree method**".

visualize work at each node/leaf:



Closed form: $W_{sum}(n) = n \cdot c1 + (n+1) \cdot c_0$

Work analysis for sum

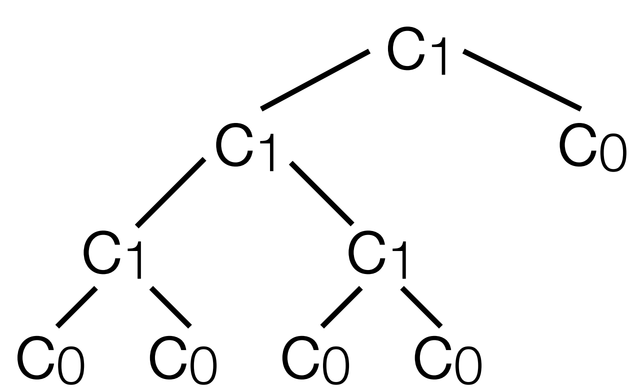
Recurrence relation:

$$W_{sum}(0) = c_0$$

$$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$$

To find a closed form, let's employ the "**tree method**".

visualize work at each node/leaf:



A binary tree has n nodes and n+1 leaves

Closed form: $W_{sum}(n) = n \cdot c1 + (n+1) \cdot c_0$

Work analysis for sum

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

Closed form: $W_{sum}(n) = n \cdot c1 + (n+1) \cdot c_0$

Work analysis for sum

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

Closed form: $W_{sum}(n) = n \cdot c1 + (n+1) \cdot c_0$

Consequently: $W_{sum}(n)$ is O(n).

Is there an opportunity for parallelism?

Is there an opportunity for parallelism?

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Is there an opportunity for parallelism?

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

Is there an opportunity for parallelism?

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

number of integers in left subtree

number of integers in right subtree

Is there an opportunity for parallelism?

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

number of integers in left subtree

number of integers in right subtree

Let's evaluate the two recursive calls in parallel!

Is there an opportunity for parallelism?

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$W_{sum}(0) = c_0$

$W_{sum}(n) = c_1 + W_{sum}(n_l) + W_{sum}(n_r)$

number of integers in left subtree

number of integers in right subtree

Let's evaluate the two recursive calls in parallel!

Valid b/c no data dependencies between `l` and `r` (thanks to FP).

Is there an opportunity for parallelism?

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

number of integers in left subtree

number of integers in right subtree

Let's evaluate the two recursive calls in parallel!

Valid b/c no data dependencies between `l` and `r` (thanks to FP).

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with no balance assumption:

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with no balance assumption:

$S_{sum}(n) \leq$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with no balance assumption:

$S_{sum}(n) \leq c_1 + \max(S_{sum}(n-1), S_{sum}(0))$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with no balance assumption:

$S_{sum}(n) \leq c_1 + \max(S_{sum}(n\text{-}1), S_{sum}(0))$

$\leq c_1 + S_{sum}(n\text{-}1)$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with no balance assumption:

$$
\begin{aligned}
S_{sum}(n) &\leq c_1 + \max(S_{sum}(n\text{-}1), S_{sum}(0)) \\
&\leq c_1 + S_{sum}(n\text{-}1)
\end{aligned}
$$

Consequently: $S_{sum}(n)$ is O(n), w/o balance.

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with balance assumption:

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with balance assumption:

$S_{sum}(n) \approx$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with balance assumption:

$S_{sum}(n) \approx c_1 + \max(S_{sum}(n/2), S_{sum}(n/2))$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with balance assumption:

$$\begin{aligned} S_{sum}(n) &\approx c_1 + \max(S_{sum}(n/2), S_{sum}(n/2)) \\ &= c_1 + S_{sum}(n/2) \end{aligned}$$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with balance assumption:

$$
\begin{aligned}
S_{sum}(n) &\approx c_1 + \max(S_{sum}(n/2), S_{sum}(n/2)) \\
&= c_1 + S_{sum}(n/2) \\
&= c_1 + c_1 + S_{sum}(n/4)
\end{aligned}
$$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with balance assumption:

$$
\begin{aligned}
S_{sum}(n) &\approx c_1 + \max(S_{sum}(n/2), S_{sum}(n/2)) \\
&= c_1 + S_{sum}(n/2) \\
&= c_1 + c_1 + S_{sum}(n/4) \\
&\vdots
\end{aligned}
$$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with balance assumption:

$$
\begin{aligned}
S_{sum}(n) &\approx c_1 + \max(S_{sum}(n/2), S_{sum}(n/2)) \\
&= c_1 + S_{sum}(n/2) \\
&= c_1 + c_1 + S_{sum}(n/4) \\
&\vdots \\
S_{sum}(n) &= c_0 + (\lfloor \log n \rfloor) \cdot c_1
\end{aligned}
$$

Span analysis for sum

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation:

$S_{sum}(0) = c_0$

$S_{sum}(n) = c_1 + \max(S_{sum}(n_l), S_{sum}(n_r))$

Finding closed form with balance assumption:

$$
\begin{aligned}
S_{sum}(n) &\approx c_1 + \max(S_{sum}(n/2), S_{sum}(n/2)) \\
&= c_1 + S_{sum}(n/2) \\
&= c_1 + c_1 + S_{sum}(n/4) \\
&\vdots \\
S_{sum}(n) &= c_0 + (\lfloor \log n \rfloor) \cdot c_1
\end{aligned}
$$

Consequently: $S_{sum}(n)$ is O(log n), with balance.

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

$S_{sum}(d) =$

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + \max(S_{sum}(d-1), S_{sum}(d'))$

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + \max(S_{sum}(d-1), S_{sum}(d'))$

d' ≤ d-1

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + \max(S_{sum}(d-1), S_{sum}(d'))$

d' ≤ d-1

Finding closed form:

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + \max(S_{sum}(d\text{-}1), S_{sum}(d'))$

d' ≤ d-1

Finding closed form:

$S_{sum}(d) \approx$

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + \max(S_{sum}(d-1), S_{sum}(d'))$

$d' \leq d-1$

Finding closed form:

$S_{sum}(d) \approx c_1 + S_{sum}(d-1)$

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{\mathtt{sum}}(0) = c_0$

$S_{\mathtt{sum}}(d) = c_1 + \max(S_{\mathtt{sum}}(d\text{-}1), S_{\mathtt{sum}}(d'))$

$d' \leq d\text{-}1$

Finding closed form:

$S_{\mathtt{sum}}(d) \approx c_1 + S_{\mathtt{sum}}(d\text{-}1)$

$\vdots$

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + \max(S_{sum}(d-1), S_{sum}(d'))$

d' ≤ d-1

Finding closed form:

$$
\begin{aligned}
S_{sum}(d) &\approx c_1 + S_{sum}(d-1) \\
&\vdots \\
S_{sum}(n) &= c_0 + d \cdot c_1
\end{aligned}
$$

Span analysis for sum (with depth)

```
fun sum (Empty : tree) : int = 0
  | sum (Node(l,x,r)) = (sum l) + (sum r) + x
```

Recurrence relation, now with depth **d** rather than number of integers:

$S_{sum}(0) = c_0$

$S_{sum}(d) = c_1 + \max(S_{sum}(d\text{-}1), S_{sum}(d'))$

d' ≤ d-1

Finding closed form:

$S_{sum}(d) \approx c_1 + S_{sum}(d\text{-}1)$

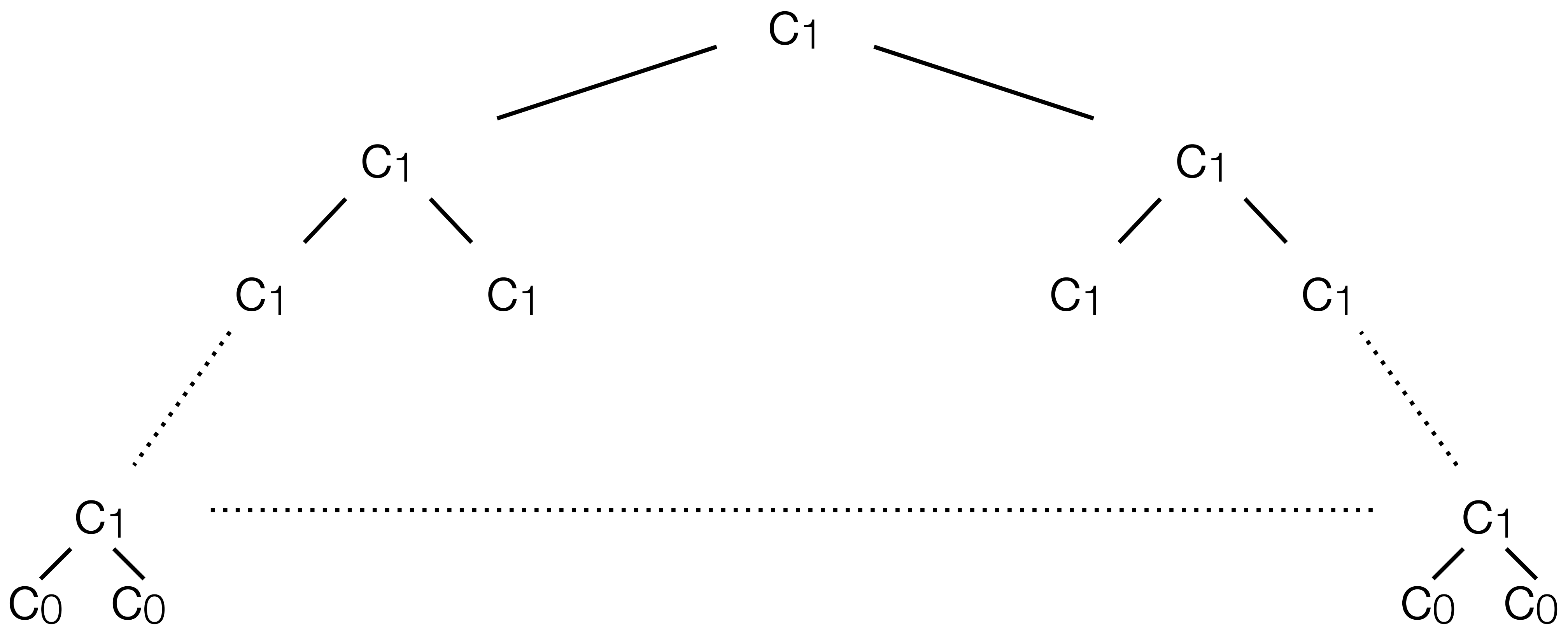
$\vdots$

$S_{sum}(n) = c_0 + d \cdot c_1$

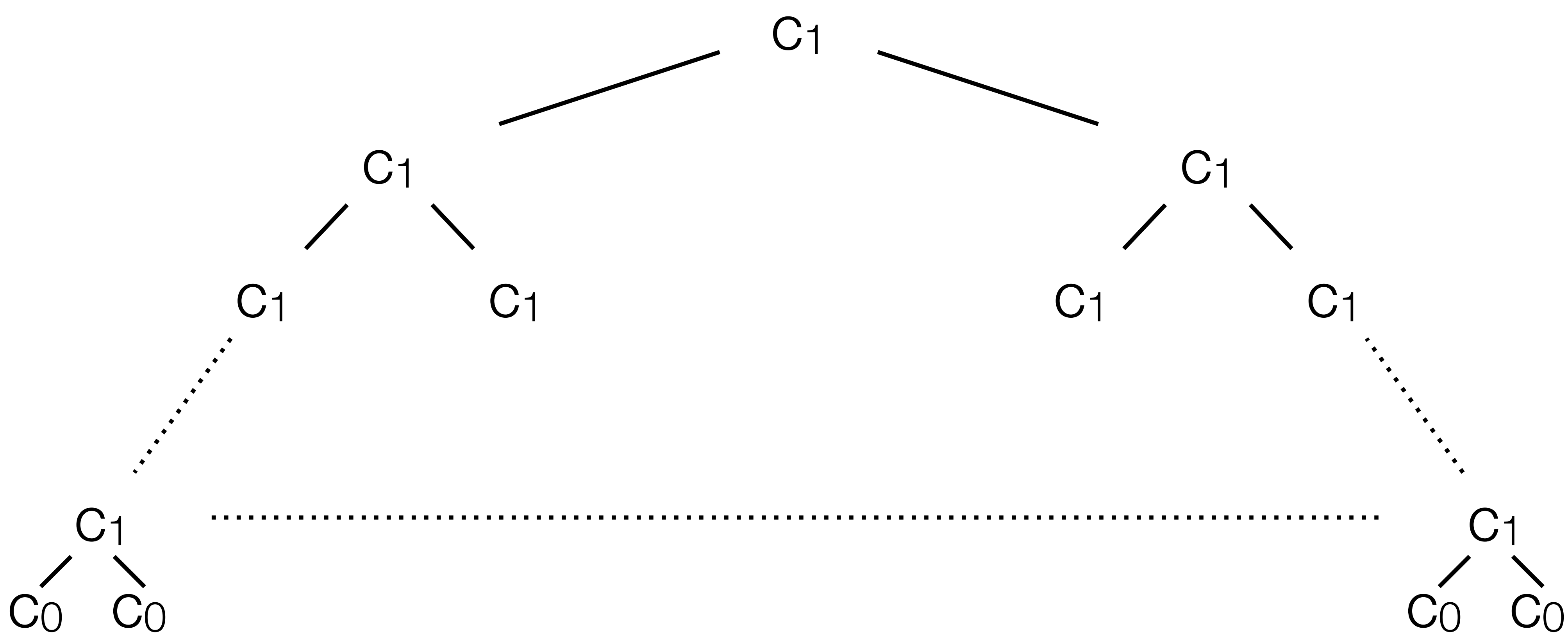
Consequently: $S_{sum}(d)$ is O(**d**), where **d** = log(**n**), if balanced.

Visualize work/span for sum (with depth)

Visualize work/span for sum (with depth)

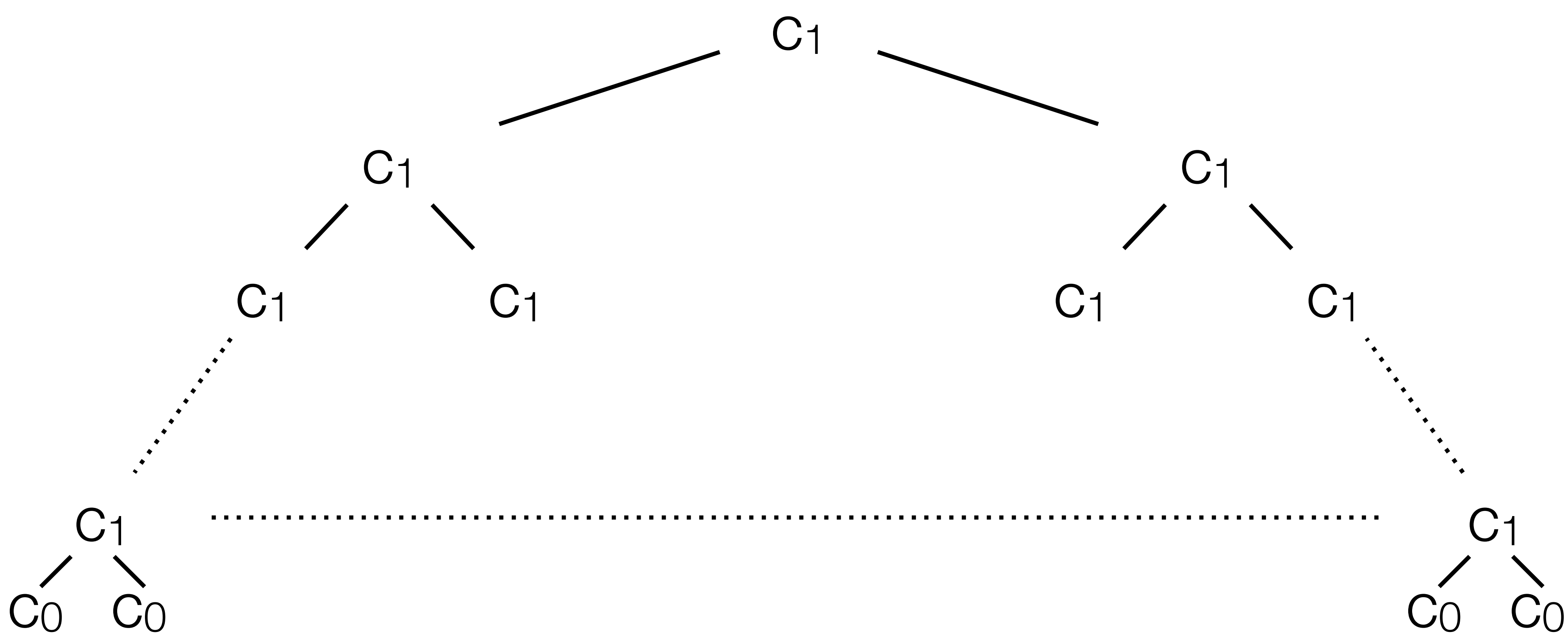


Visualize work/span for sum (with depth)



$1c_1$

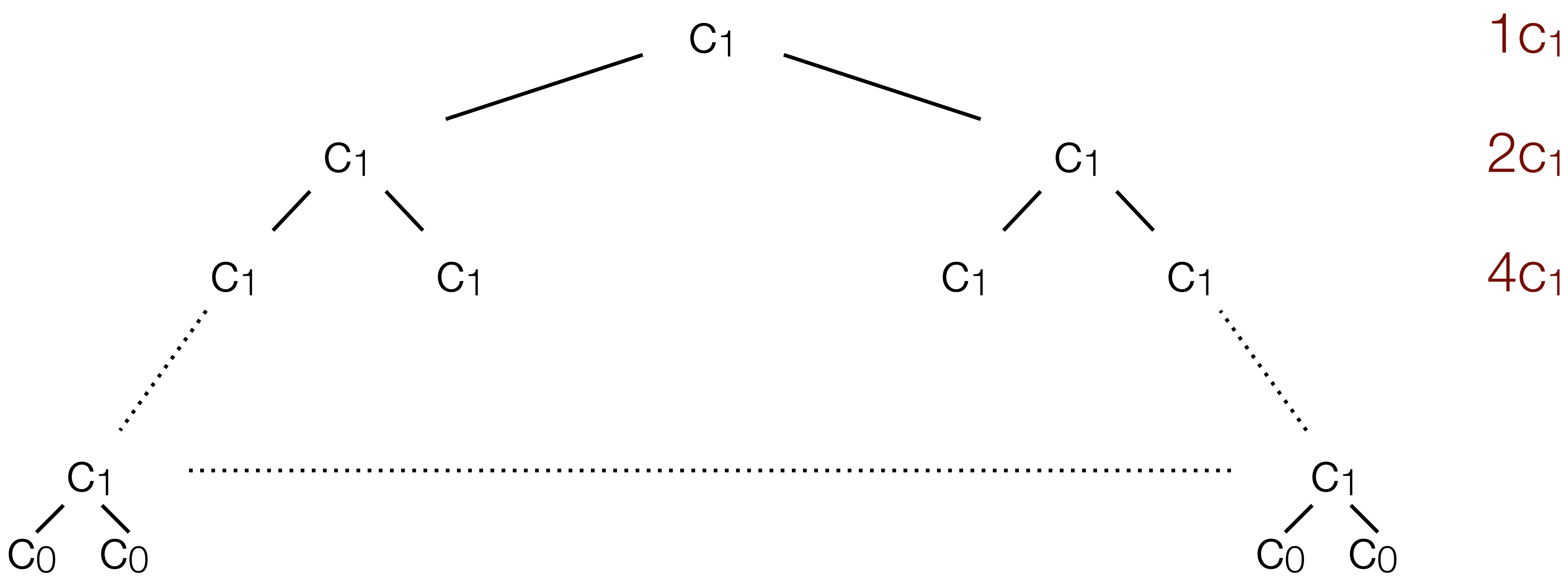
Visualize work/span for sum (with depth)



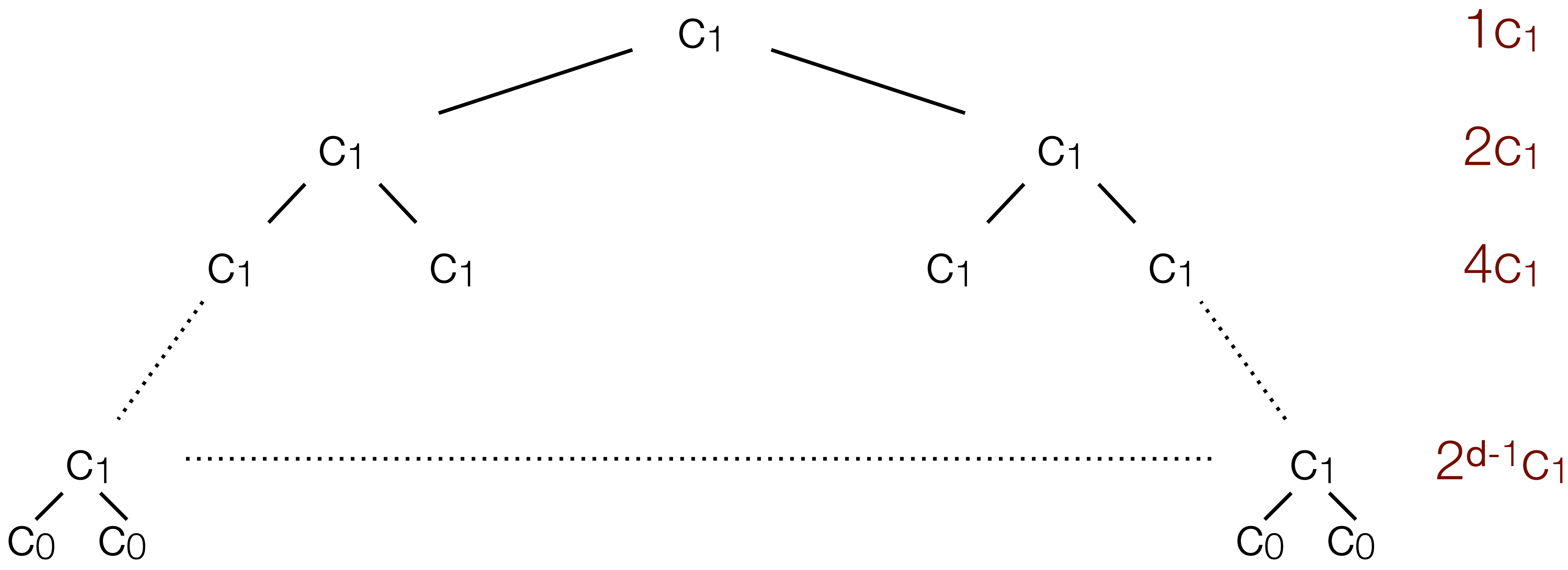
$1c_1$

$2c_1$

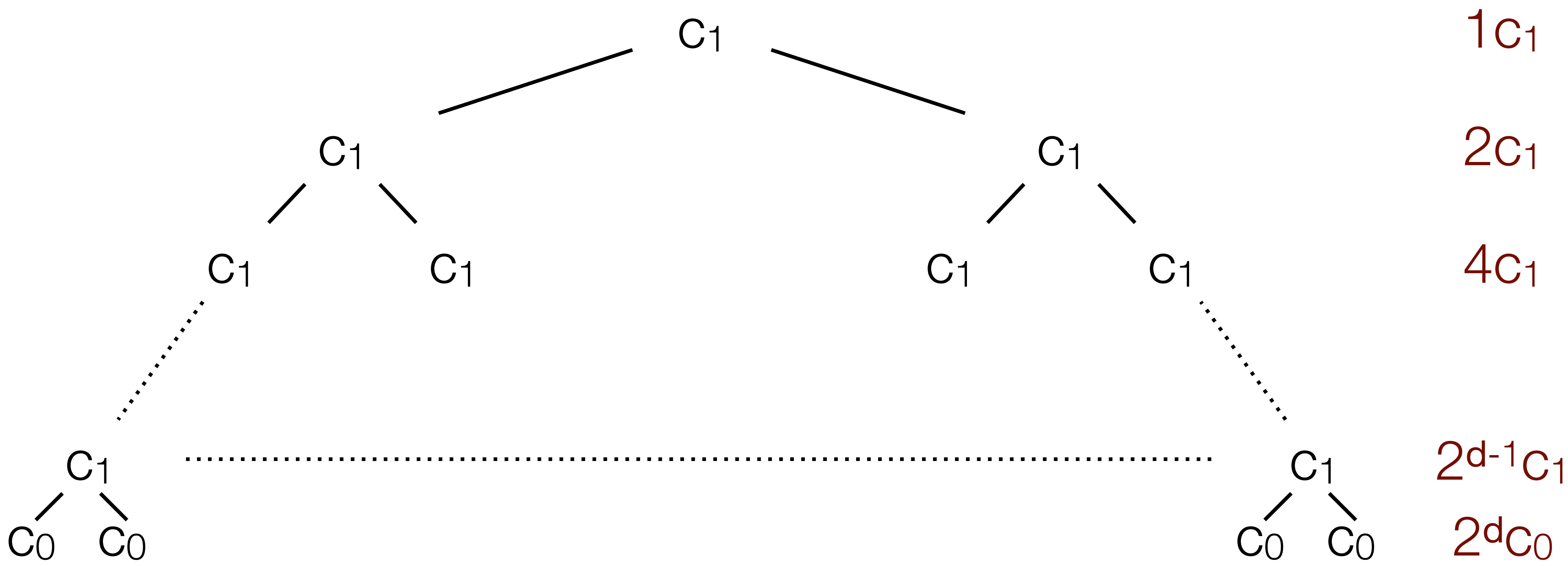
Visualize work/span for sum (with depth)



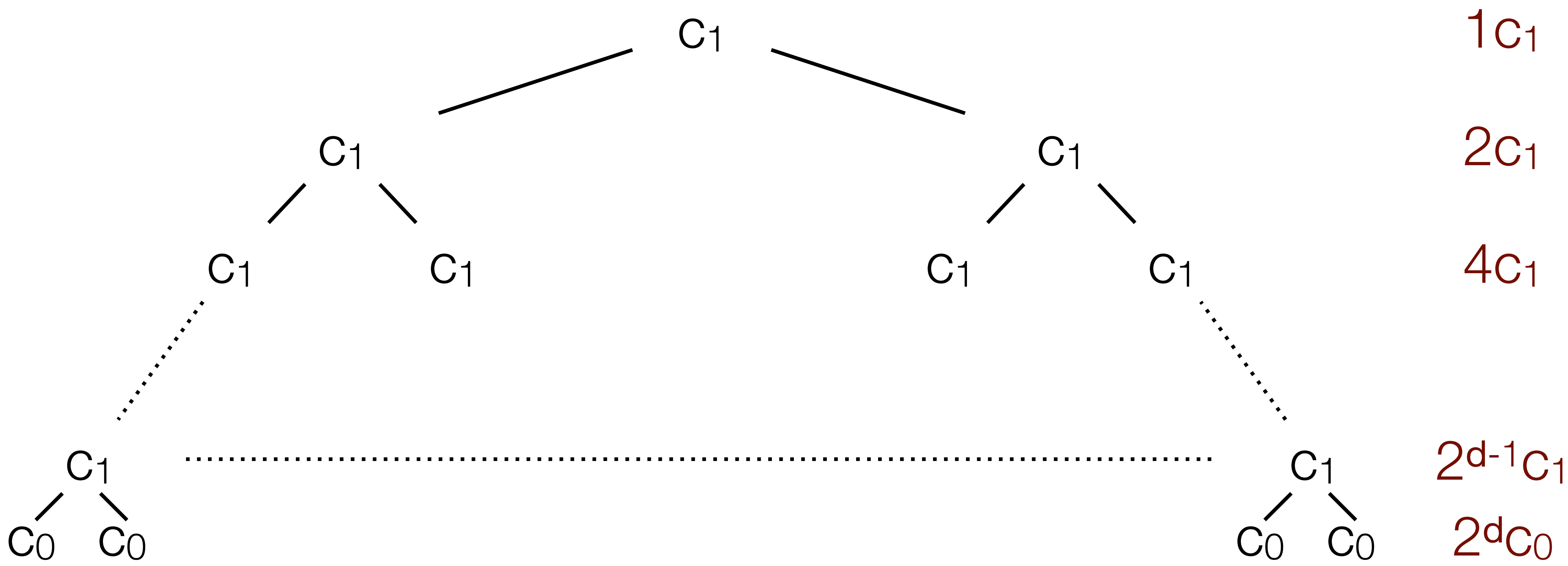
Visualize work/span for sum (with depth)



Visualize work/span for sum (with depth)

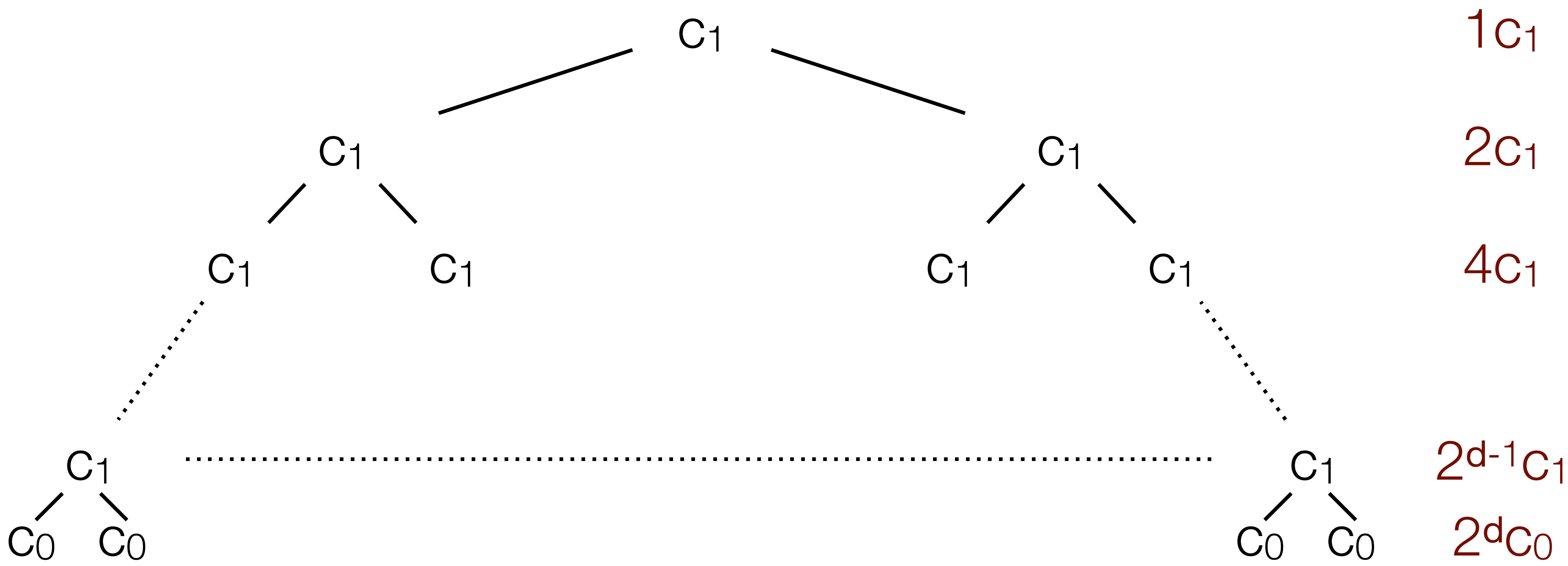


Visualize work/span for sum (with depth)



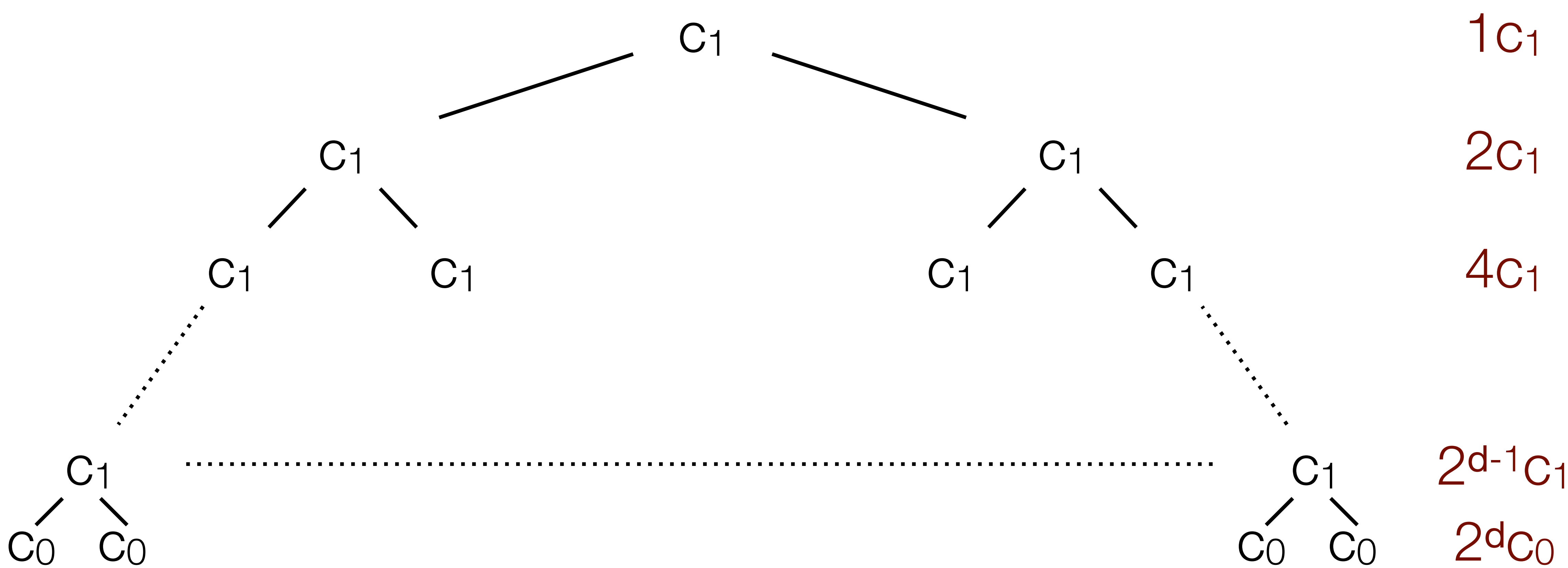
$\mathtt{W_{sum}(d)} = (2^0 + 2^1 + \ldots + 2^{d-1})\cdot \mathbf{c_1} + 2^d\cdot \mathbf{c_0} \leq 2^{d+1}\cdot \mathbf{c}$

Visualize work/span for sum (with depth)



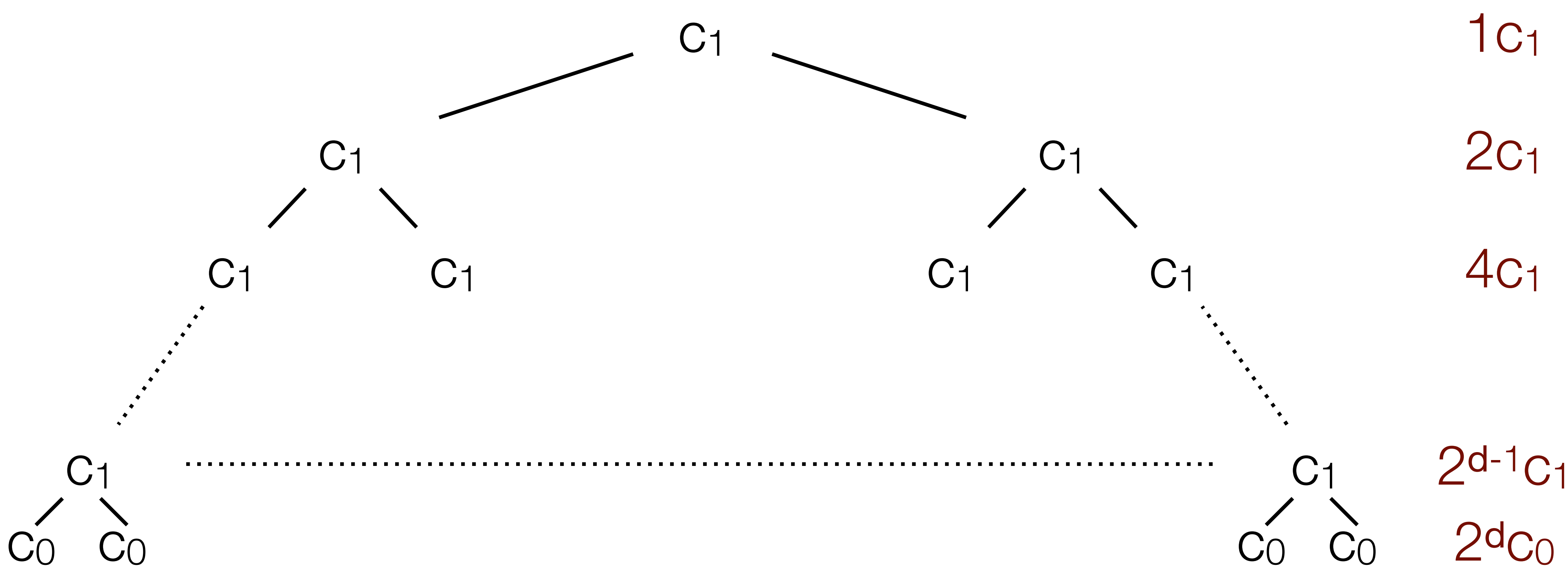
$$\mathtt{W_{sum}(d)} = (2^0 + 2^1 + \ldots + 2^{d-1})\cdot c_1 + 2^d\cdot c_0 \leq 2^{d+1}\cdot c$$

Visualize work/span for sum (with depth)



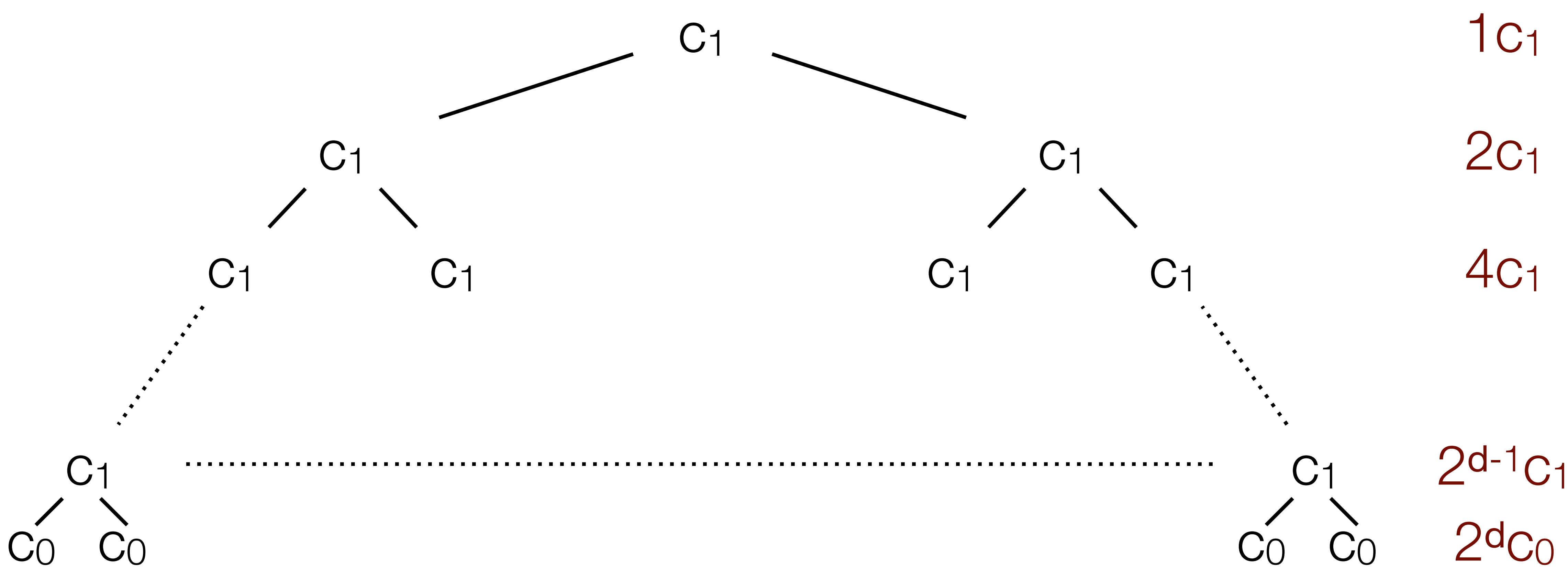
$$W_{sum}(d) = \underbrace{(2^0 + 2^1 + \ldots + 2^{d-1})}_{< 2^d} \cdot c_1 + 2^d \cdot c_0 \le 2^{d+1} \cdot c$$

Visualize work/span for sum (with depth)



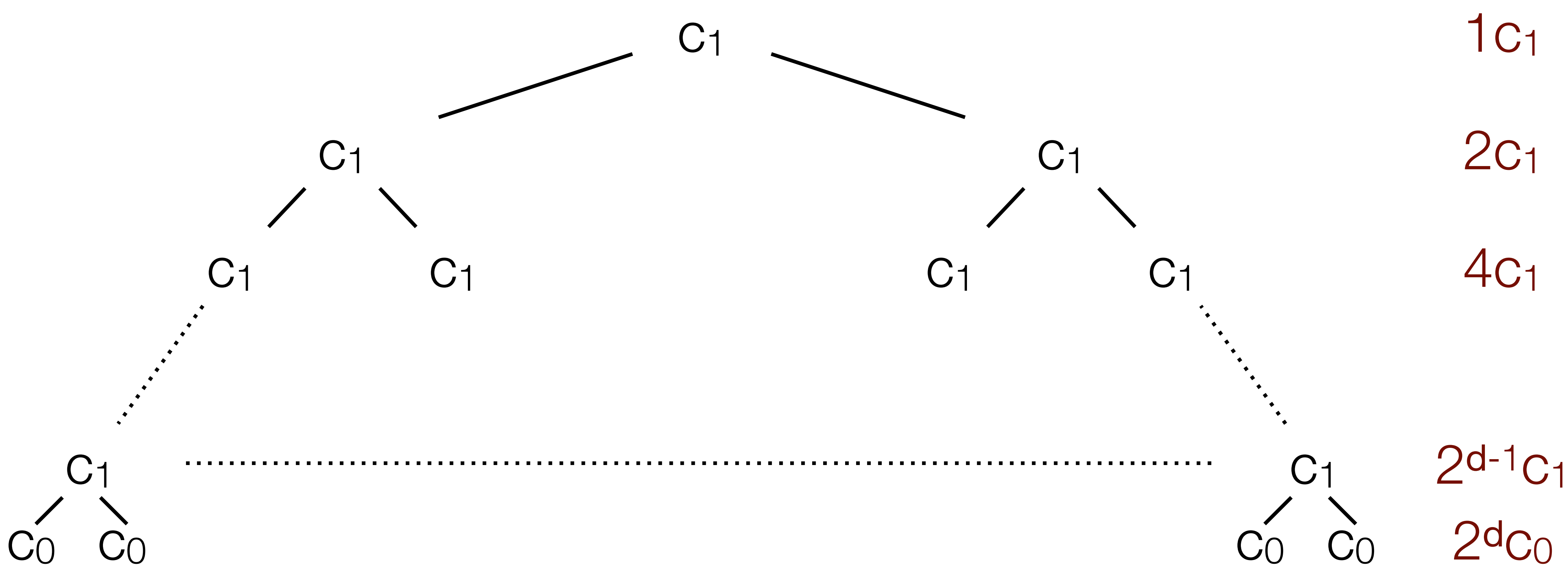
$$W_{sum}(d) = (\underbrace{2^0 + 2^1 + \ldots + 2^{d-1}}_{< 2^d})\cdot c_1 + 2^d\cdot c_0 \leq 2^{d+1}\cdot c$$

Visualize work/span for sum (with depth)



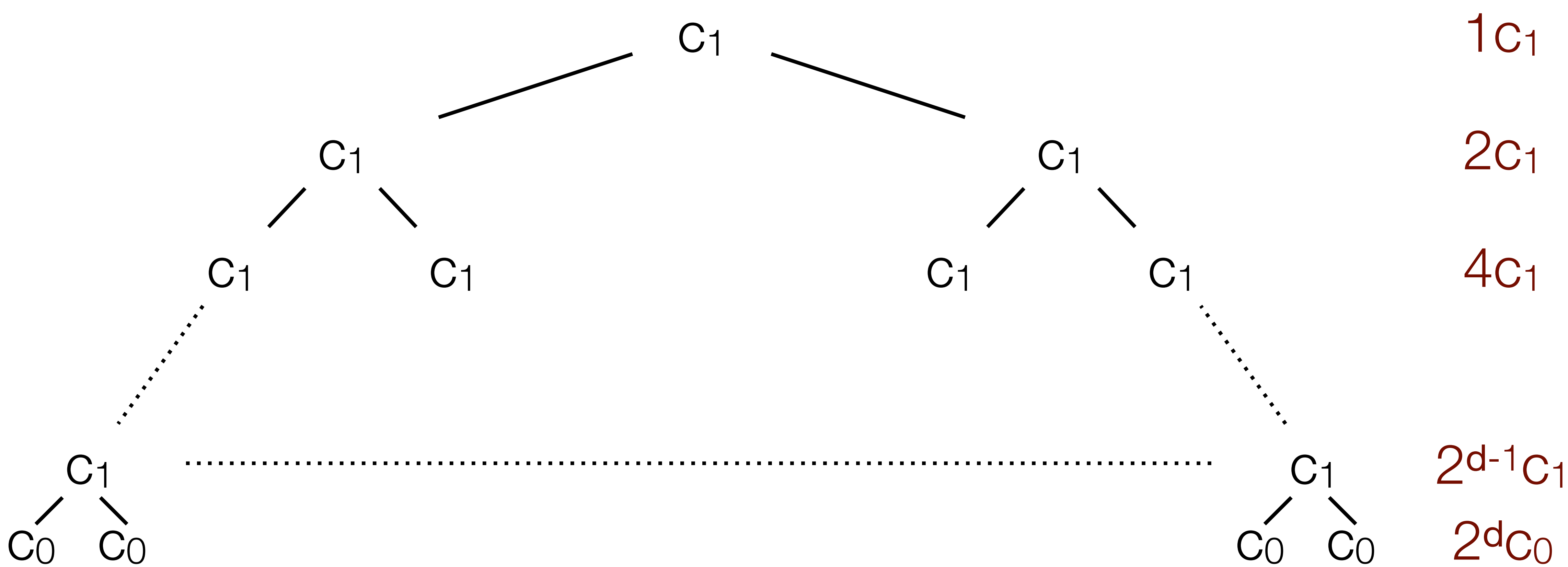
$$W_{sum}(d) = (\underbrace{2^0 + 2^1 + \ldots + 2^{d-1}}_{< 2^d})\cdot c_1 + 2^d \cdot c_0 \leq 2^{d+1} \cdot \underbrace{c}_{\max(c_0,\, c_1)}$$

Visualize work/span for sum (with depth)



$$W_{sum}(d) = (2^0 + 2^1 + \ldots + 2^{d-1})\cdot c_1 + 2^d\cdot c_0 \leq 2^{d+1}\cdot c$$

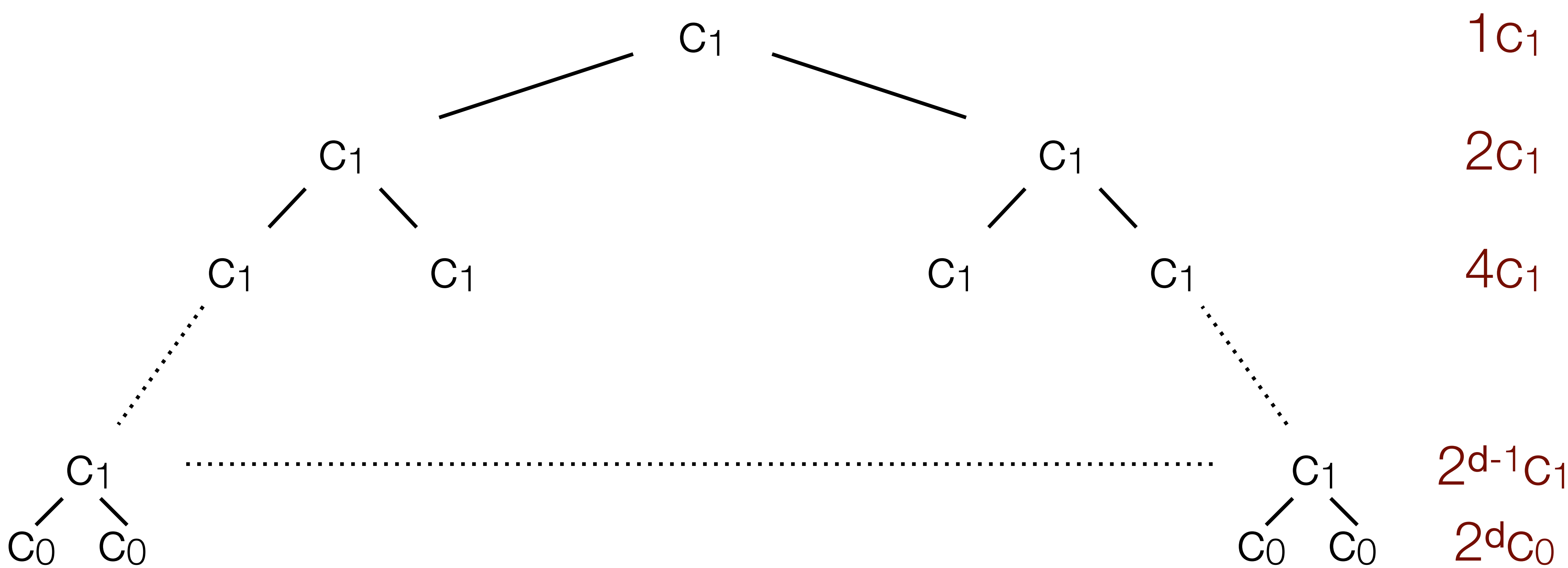
Visualize work/span for sum (with depth)



$$W_{sum}(d) = (2^0 + 2^1 + \ldots + 2^{d-1})\cdot c_1 + 2^d\cdot c_0 \leq 2^{d+1}\cdot c$$

$$S_{sum}(d) = (1 + 1 + \ldots + 1)\cdot c_1 + c_0 \leq d\cdot c$$

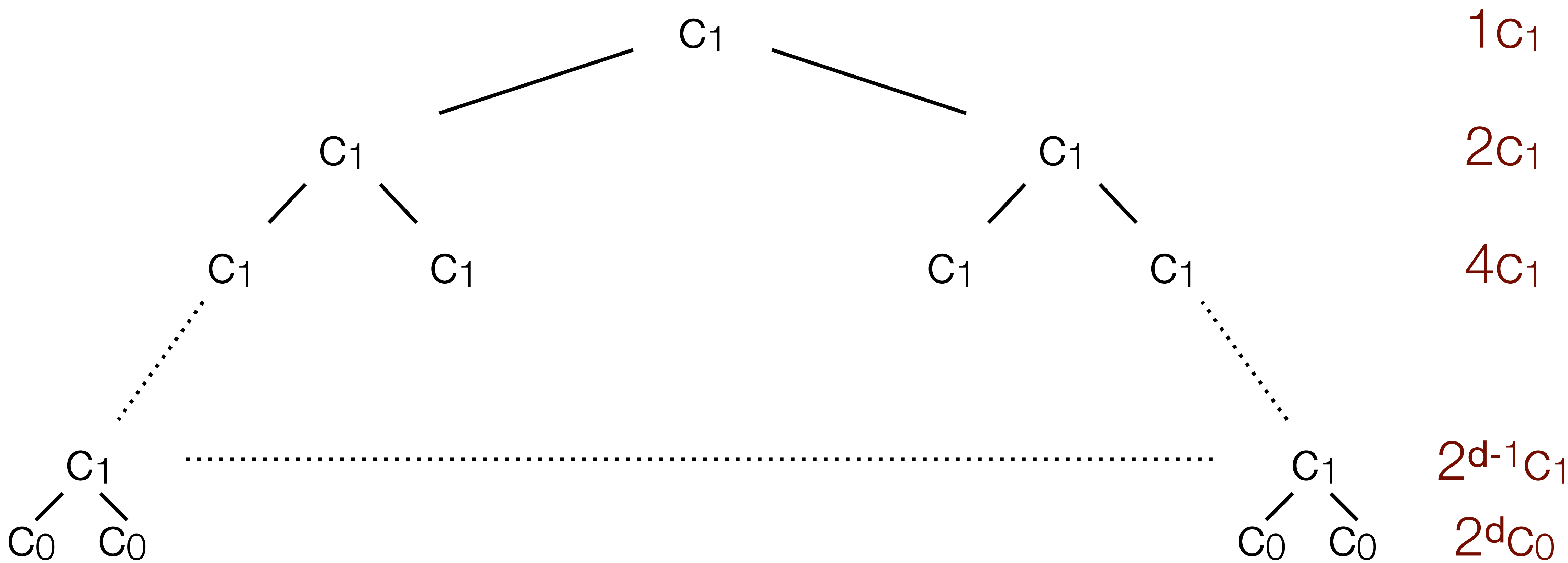
Visualize work/span for sum (with depth)



$$\texttt{W}_{\texttt{sum}}\texttt{(d)} = (2^0 + 2^1 + \ldots + 2^{d-1})\cdot c_1 + 2^d\cdot c_0 \leq 2^{d+1}\cdot c$$

$$\texttt{S}_{\texttt{sum}}\texttt{(d)} = (1 + 1 + \ldots + 1)\cdot c_1 + c_0 \leq d\cdot c$$

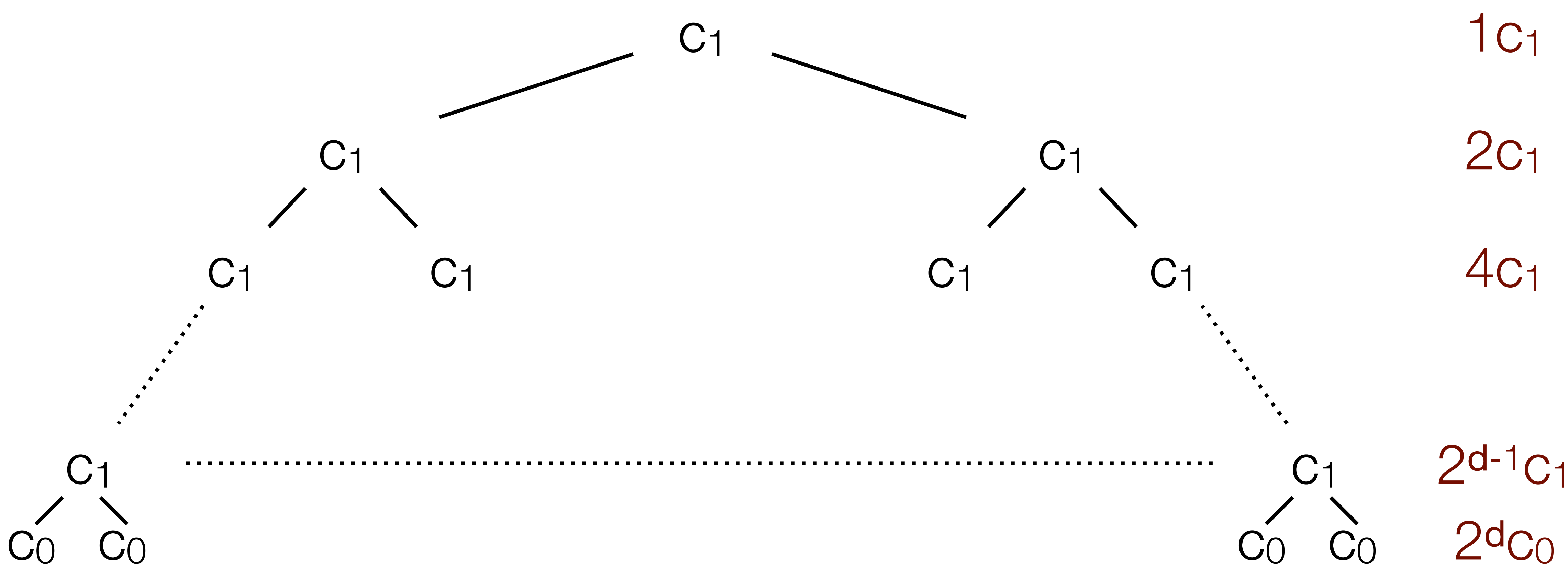
Visualize work/span for sum (with depth)



$$\mathtt{W}_{\mathtt{sum}}\mathtt{(d)} = (2^0 + 2^1 + \ldots + 2^{d-1})\cdot c_1 + 2^d\cdot c_0 \leq 2^{d+1}\cdot c$$

$$\mathtt{S}_{\mathtt{sum}}\mathtt{(d)} = (\underbrace{1 + 1 + \ldots + 1}_{\text{d-1 times}})\cdot c_1 + c_0 \leq d\cdot c$$

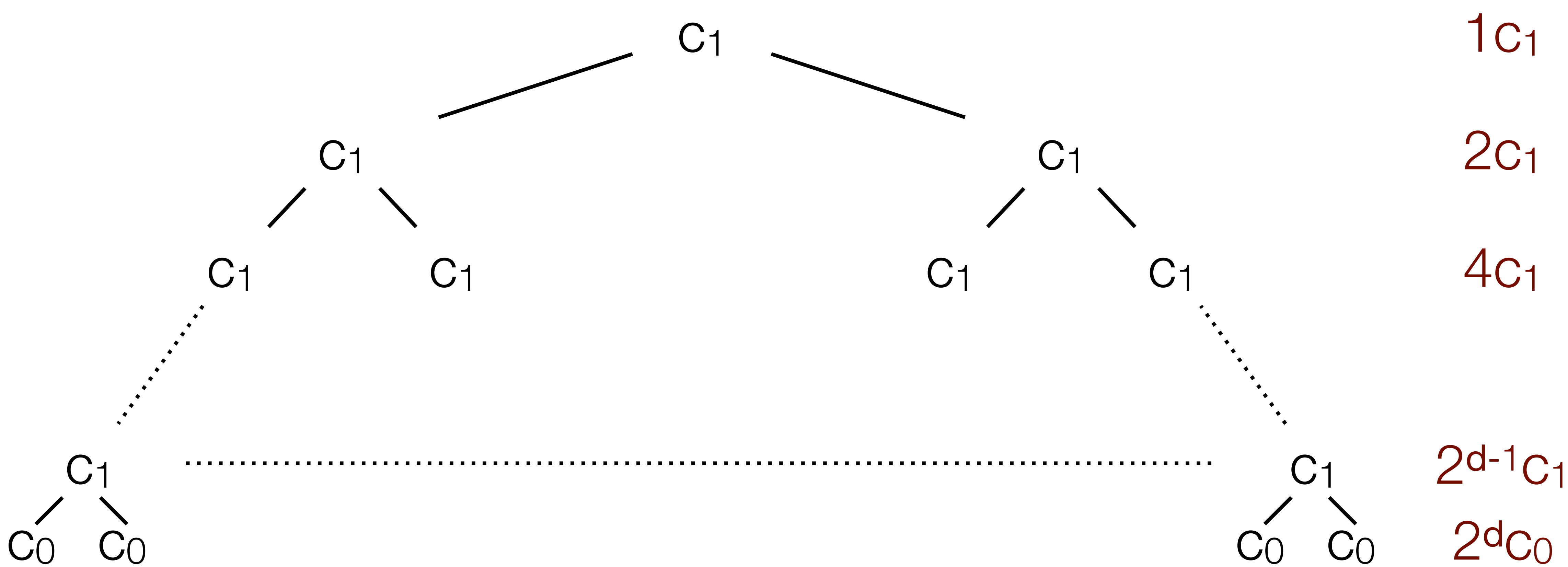
Visualize work/span for sum (with depth)



$$W_{sum}(d) = (2^0 + 2^1 + \dots + 2^{d-1})\cdot c_1 + 2^d\cdot c_0 \leq 2^{d+1}\cdot c$$

$$S_{sum}(d) = (\underbrace{1 + 1 + \dots + 1}_{\text{d-1 times}})\cdot c_1 + c_0 \leq d\cdot c$$

Visualize work/span for sum (with depth)



$$W_{sum}(d) = (2^0 + 2^1 + \ldots + 2^{d-1})\cdot c_1 + 2^d \cdot c_0 \leq 2^{d+1}\cdot c$$

$$S_{sum}(d) = (\underbrace{1 + 1 + \ldots + 1}_{\text{d-1 times}})\cdot c_1 + c_0 \leq d\cdot \underbrace{c}_{\max(c_0,\, c_1)}$$

That's all for today. See you on Thursday!