Lecture 10

Finding strongly connected components

Announcements

- Midterm is done!
  - Nice job everyone!!!!
  - (And I hope that you are still speaking to me…)
  - If you lost either a pencil case or a totebag with a book in it, let me know (marykw@stanford.edu)
- We hope to have the exam graded soon, and will release grades/solutions this week.

- Give yourselves a hand! 🎉 👏
- Give the CAs a hand for an impressive rally during the printer outage! 🖨️

Announcements

- HW4 out now.
  - Due on FRIDAY, not Wednesday.
- HW5 will be out Wednesday.
  - Due the following Wednesday.
- A few things back to normal:
  - HW party is back in Fujitsu this week!
  - Apoorva's section is back in Hewlett 102!
- Ed etiquette:
  - Please post general (non-spoiler) Ed questions as public, rather than private.
  - Remember that the Ed CAs have to sleep too! Thanks for your patience ☺

Announcements

- It's that time in the quarter...

-   ...take care of yourselves!

- *Your mental and physical health is more important than this class.*

- Links to some resources on the course website:

4

Last time

- Breadth-first and depth-first search
- Plus, applications!
  - Topological sorting
  - In-order traversal of BSTs
  - Finding Bacon numbers
  - Testing for bipartite-ness

- The key was paying attention to the structure of the tree that these search algorithms implicitly build.
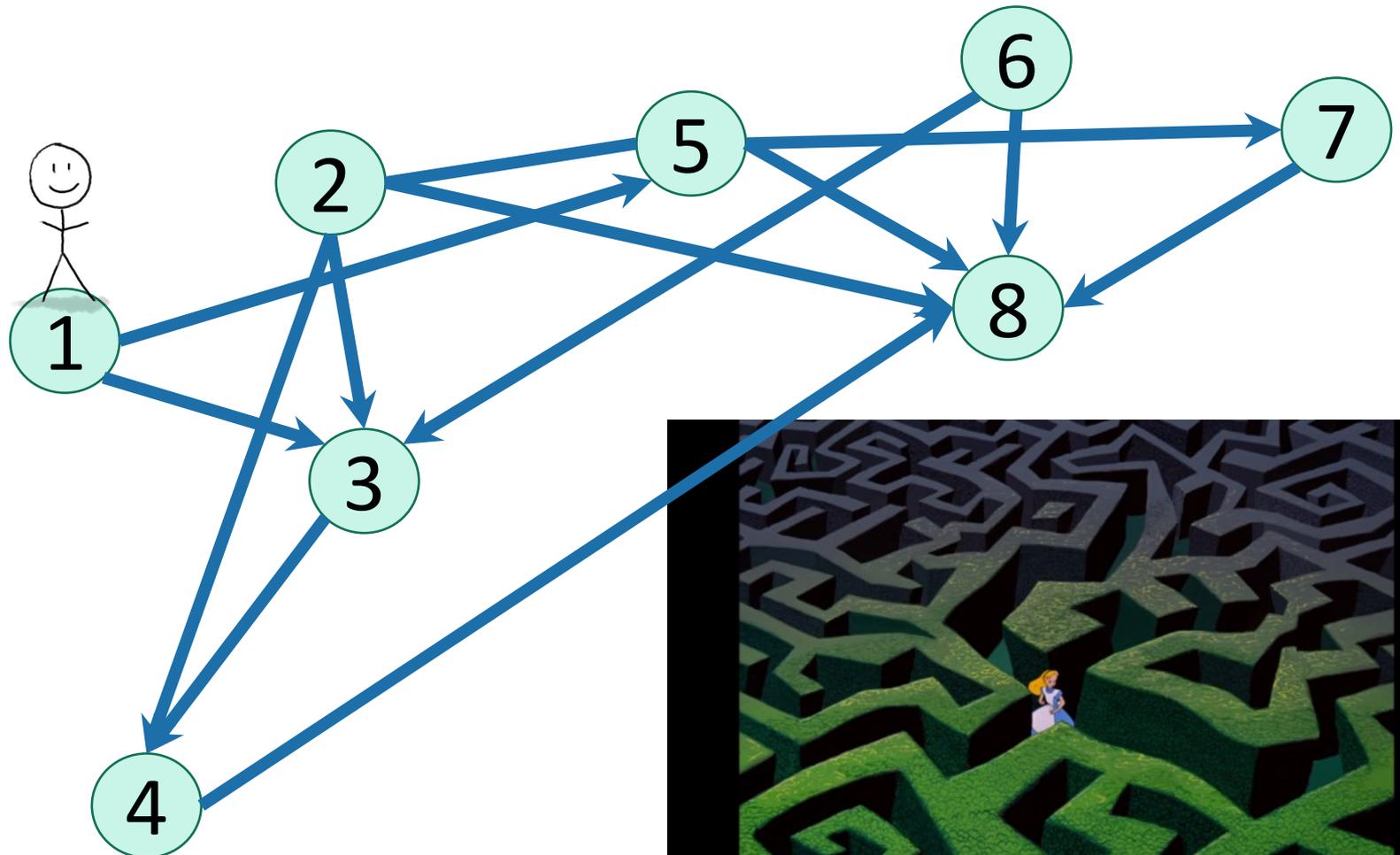
Today

- One more application of DFS:

Finding
Strongly Connected Components

Recall: DFS

It's how you'd explore a labyrinth with chalk and a piece of string.

Today, all graphs are **directed**!
Check that the things we did
last week still all work!

Depth First Search
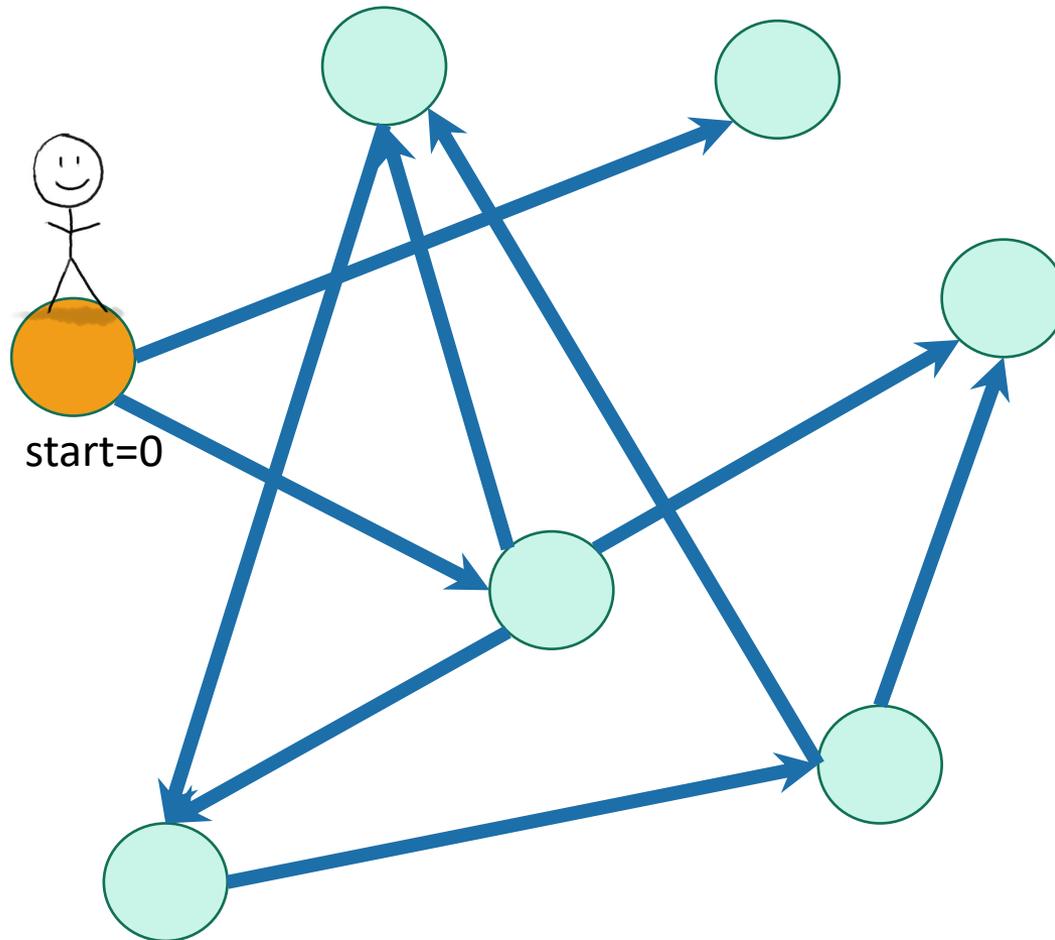## Exploring a labyrinth with chalk and a piece of string

start
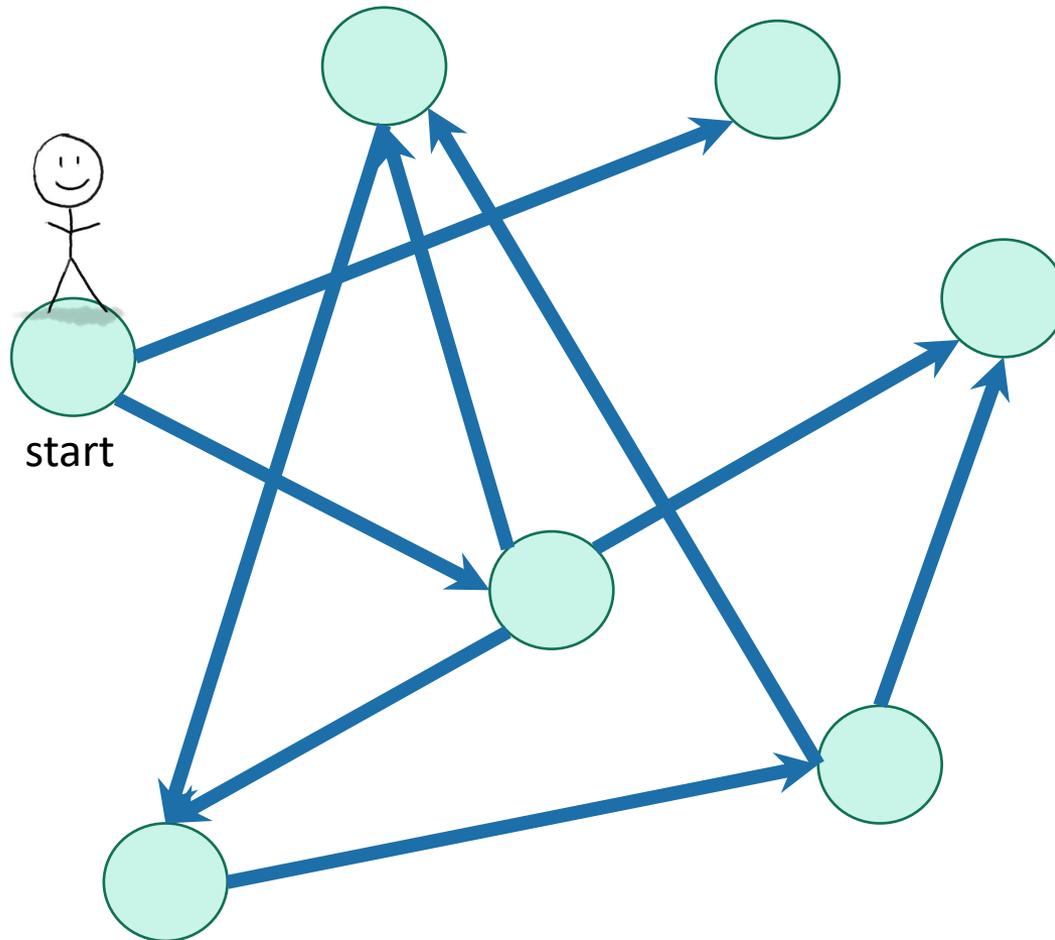
Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

This is the same picture we had Monday, except I've directed all of the edges. Notice that there **ARE** cycles.

Depth First Search
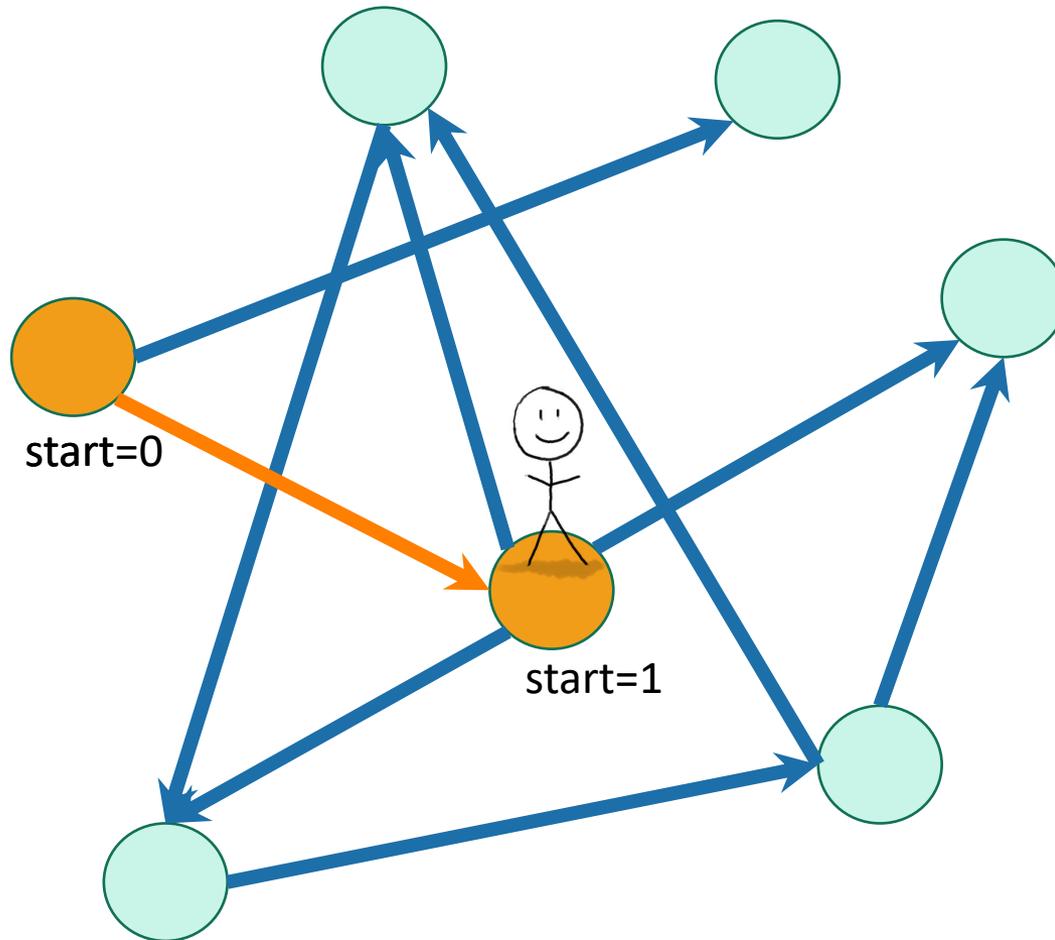Exploring a labyrinth with chalk and a piece of string

start=0

Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
Exploring a labyrinth with chalk and a piece of string

start=0

start=1

Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
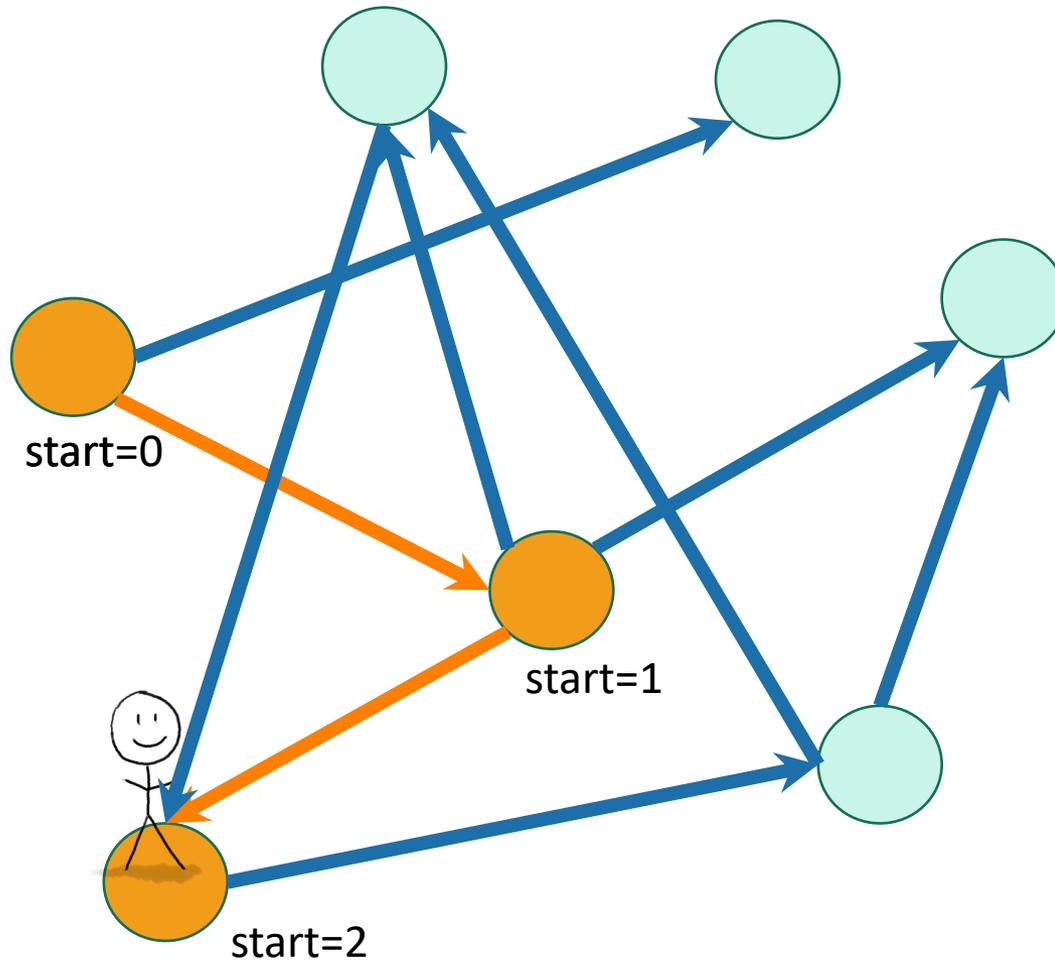Exploring a labyrinth with chalk and a piece of string

start=0

start=1

start=2
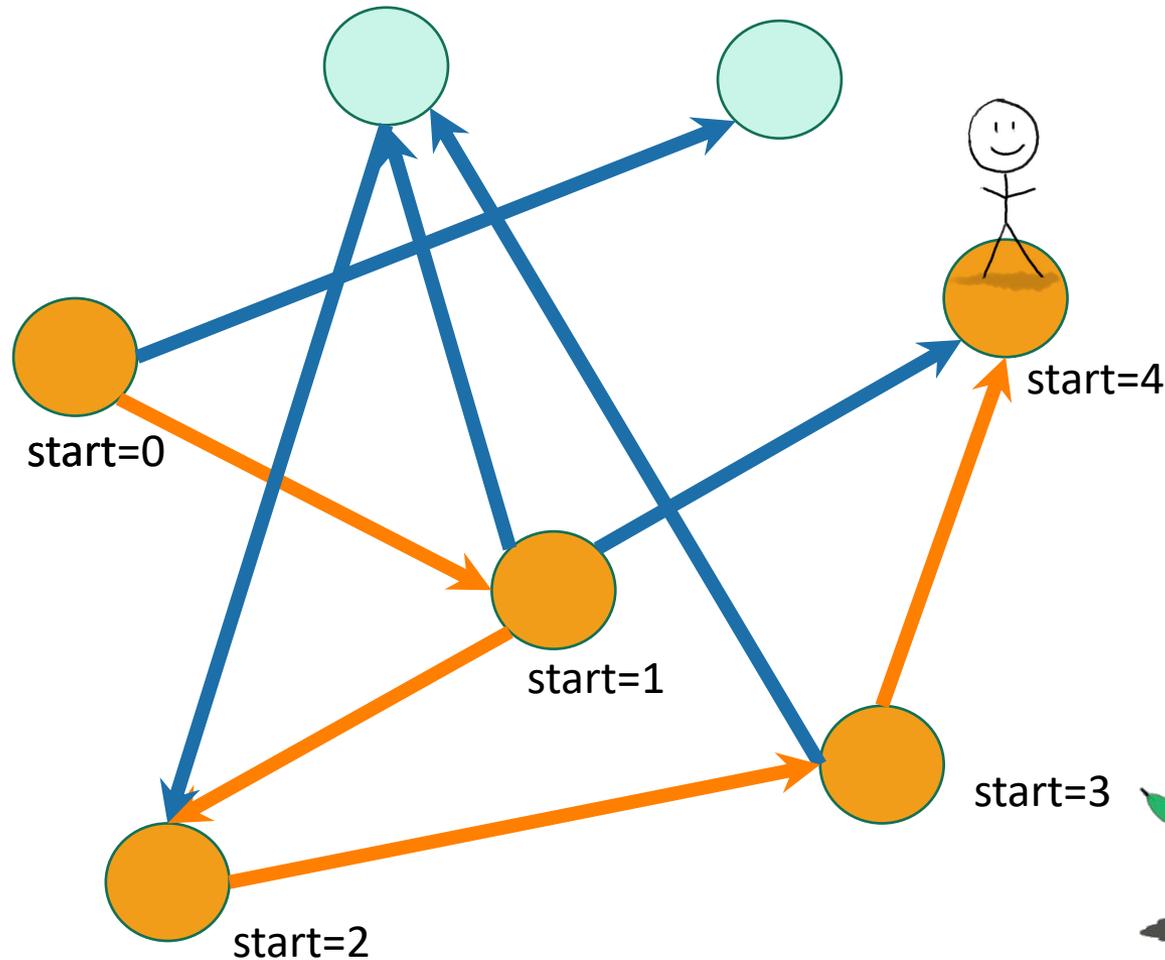
Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
Exploring a labyrinth with chalk and a piece of string

Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

start=0

start=1

start=2

start=3

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
Exploring a labyrinth with chalk and a piece of string

Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

start=0

start=1

start=2

start=3

start=4

Recall we also keep track of **start** and **finish** times for every node.

13

Depth First Search
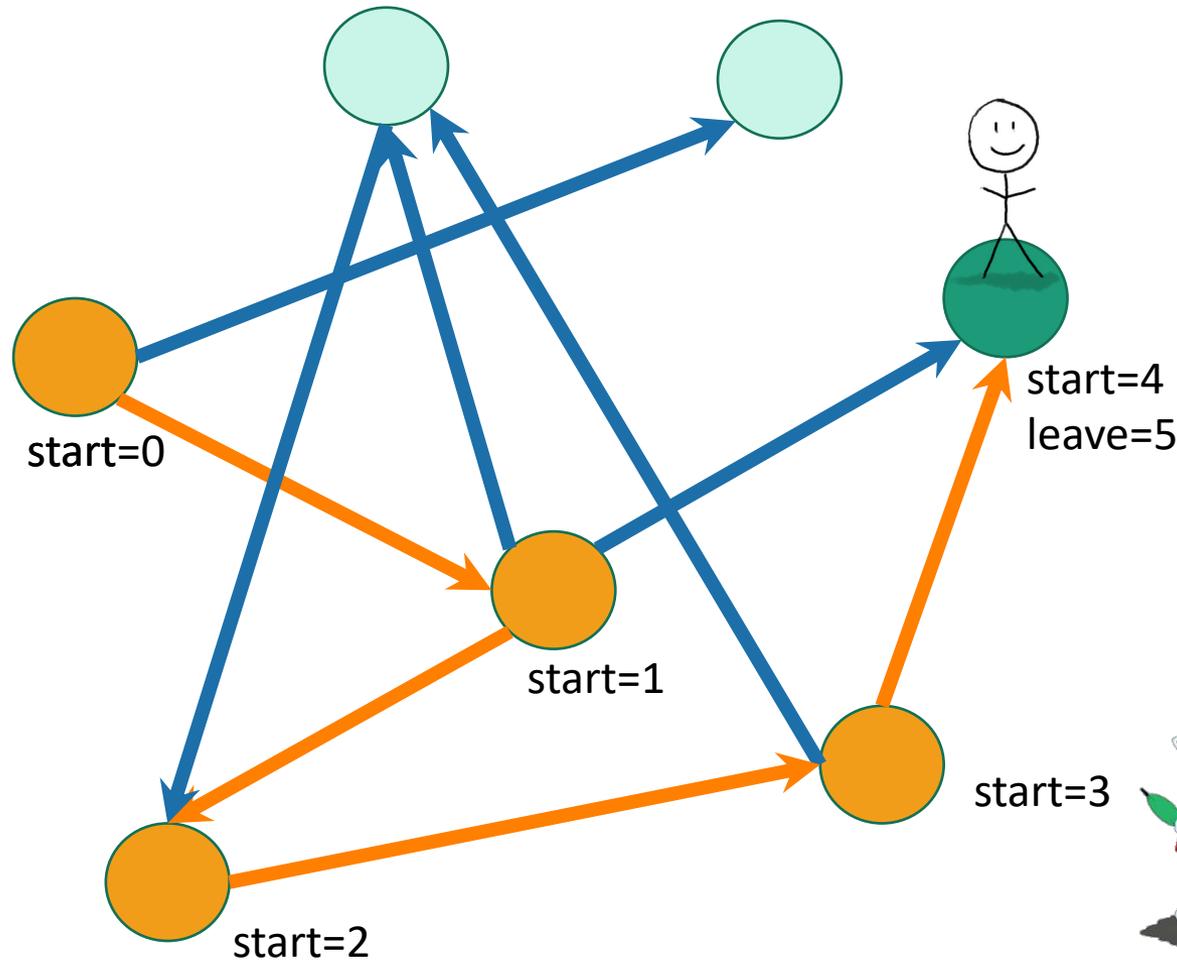## Exploring a labyrinth with chalk and a piece of string

Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

start=0

start=1

start=2

start=3

start=4
leave=5

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
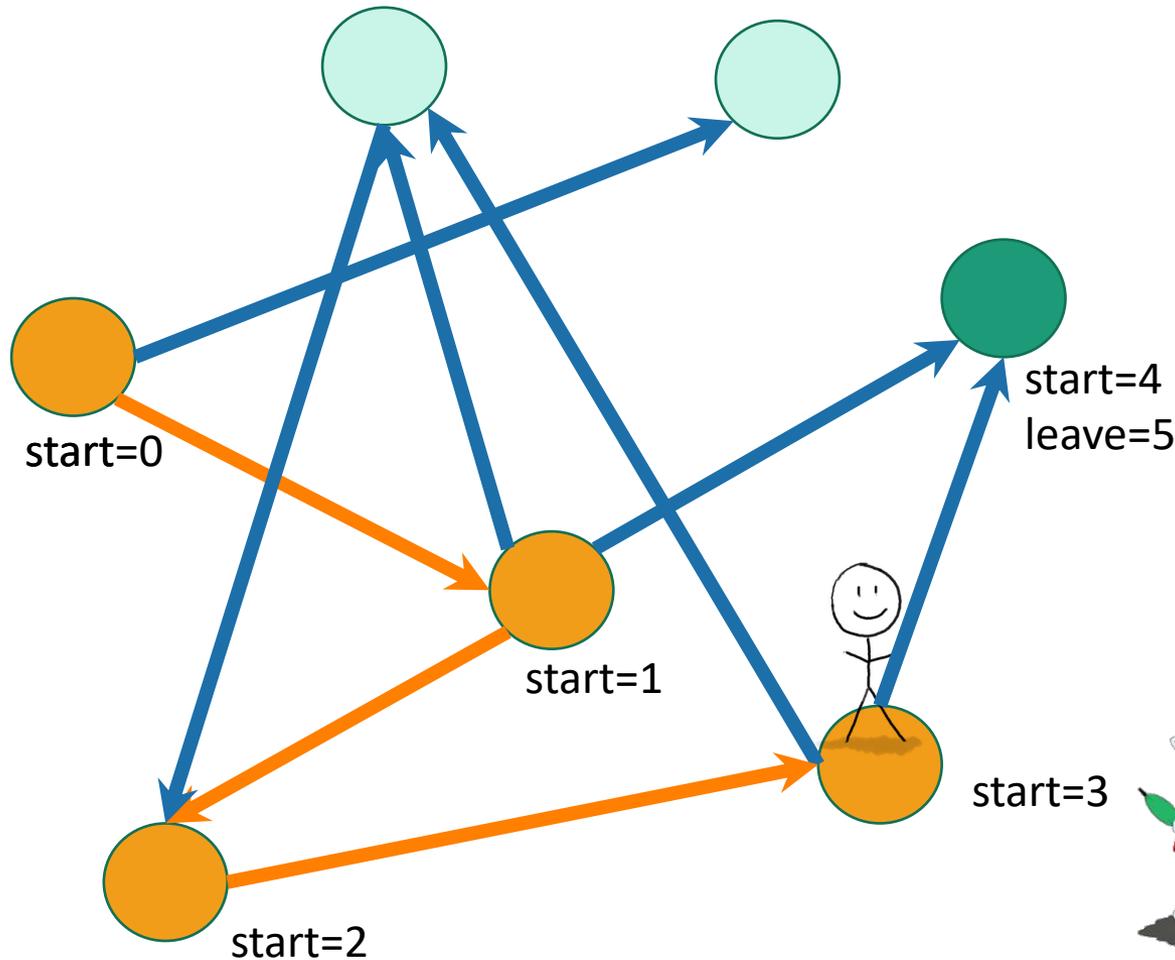## Exploring a labyrinth with chalk and a piece of string

Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

start=0

start=1

start=2

start=3

start=4
leave=5

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
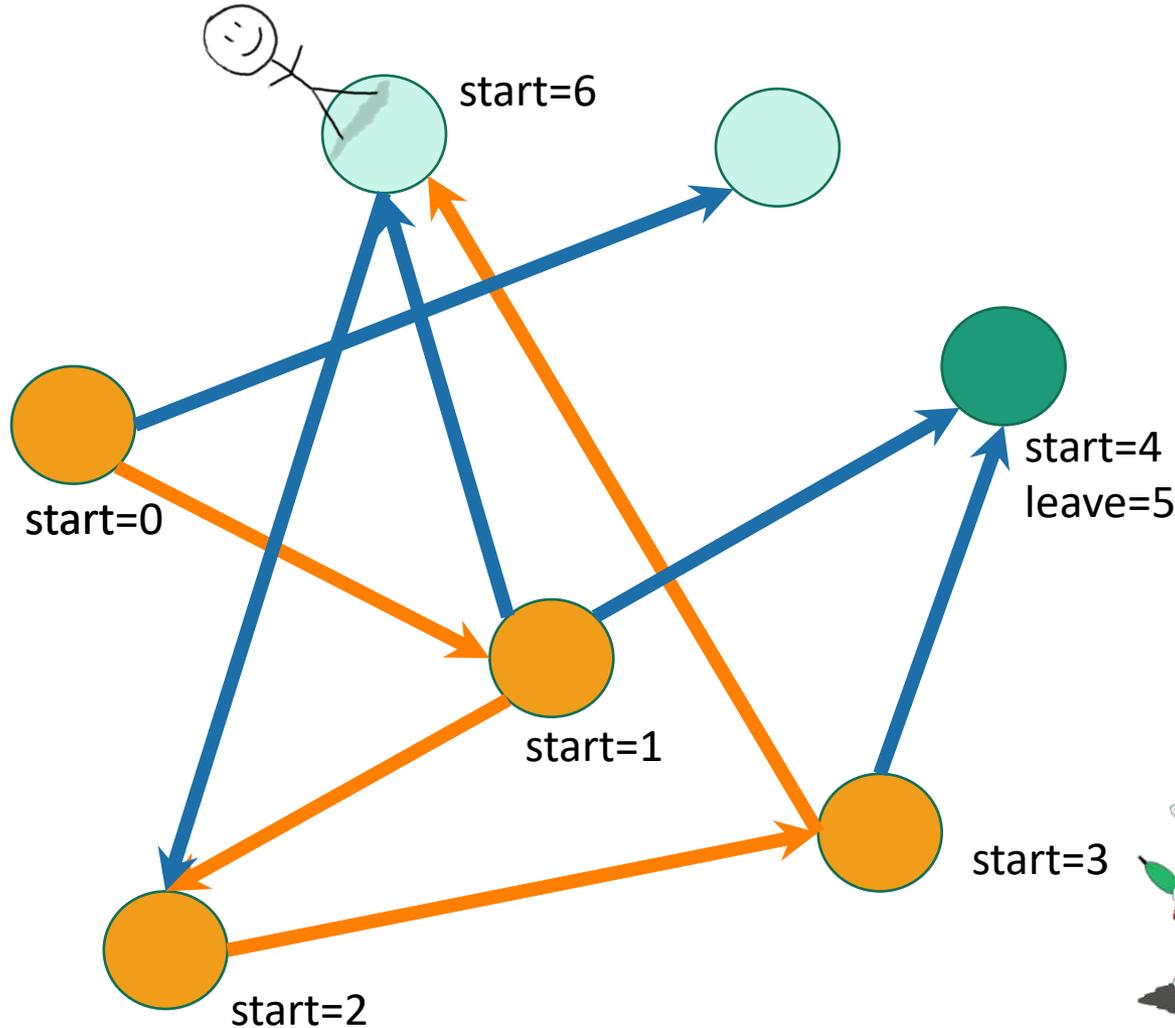Exploring a labyrinth with chalk and a piece of string

start=6

start=0

start=1

start=2

start=3

start=4
leave=5
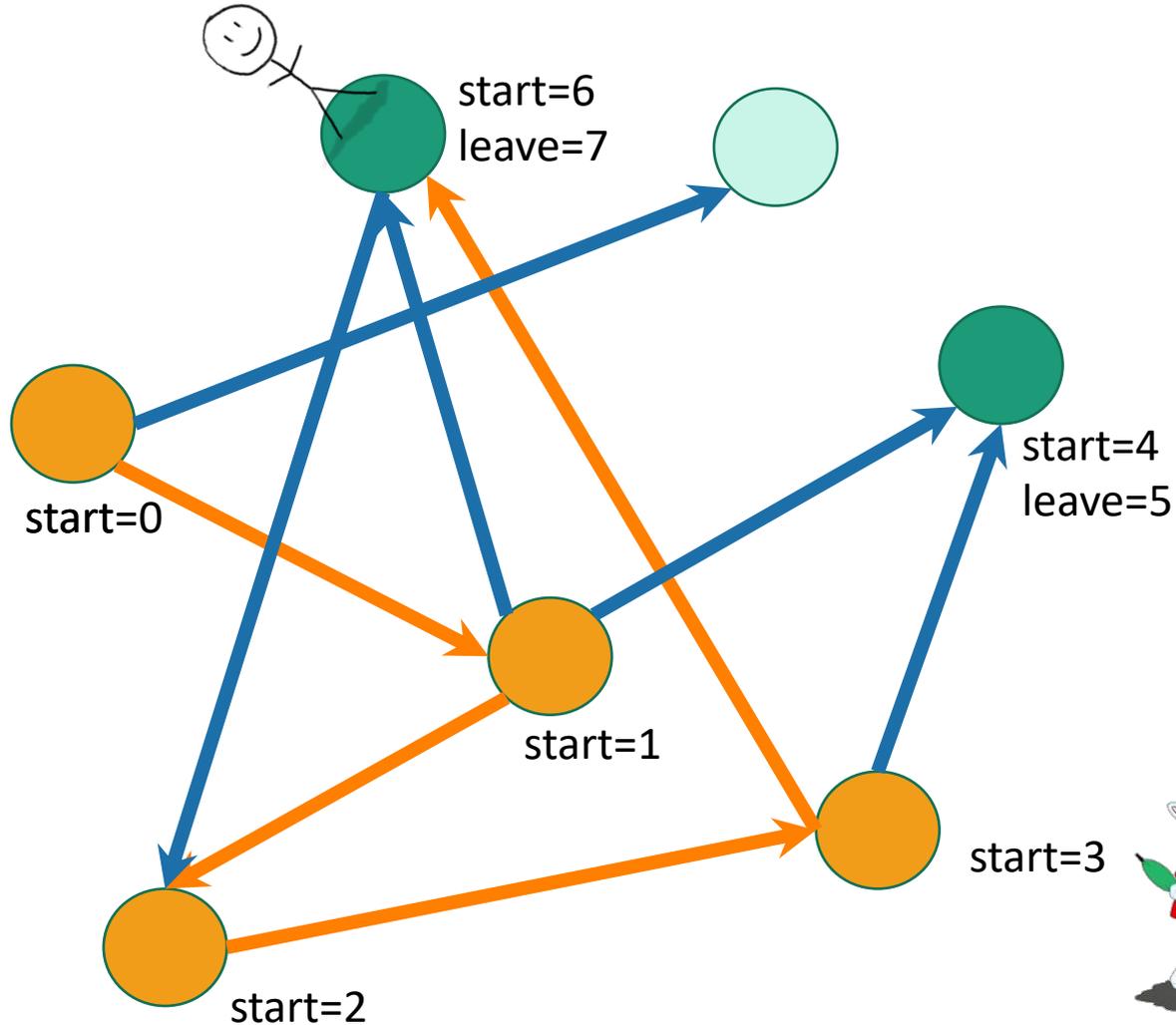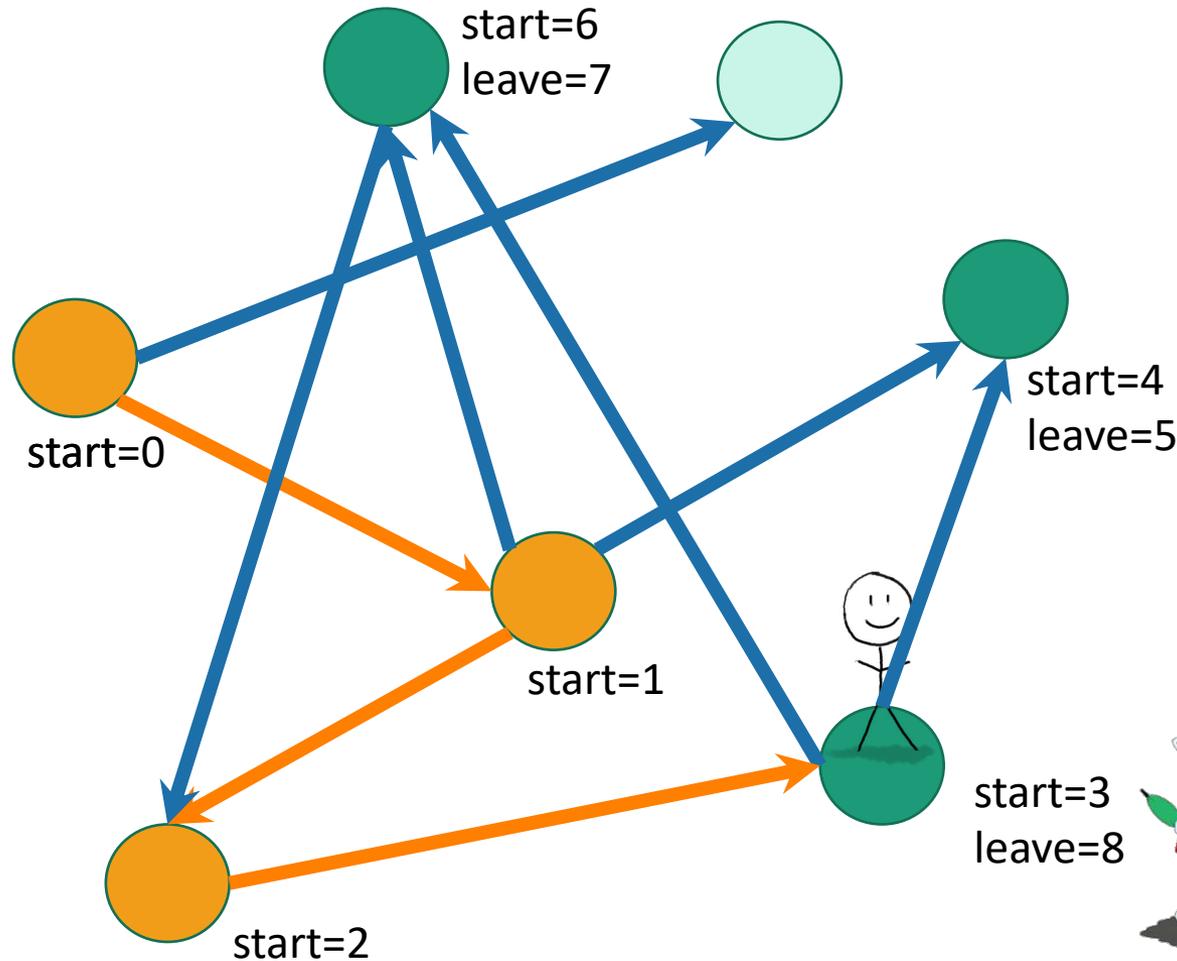
Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
Exploring a labyrinth with chalk and a piece of string

start=6
leave=7

Not been there yet

Been there, haven't explored all the paths out.

start=4
leave=5

Been there, have explored all the paths out.

start=0

start=1

start=3

start=2

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
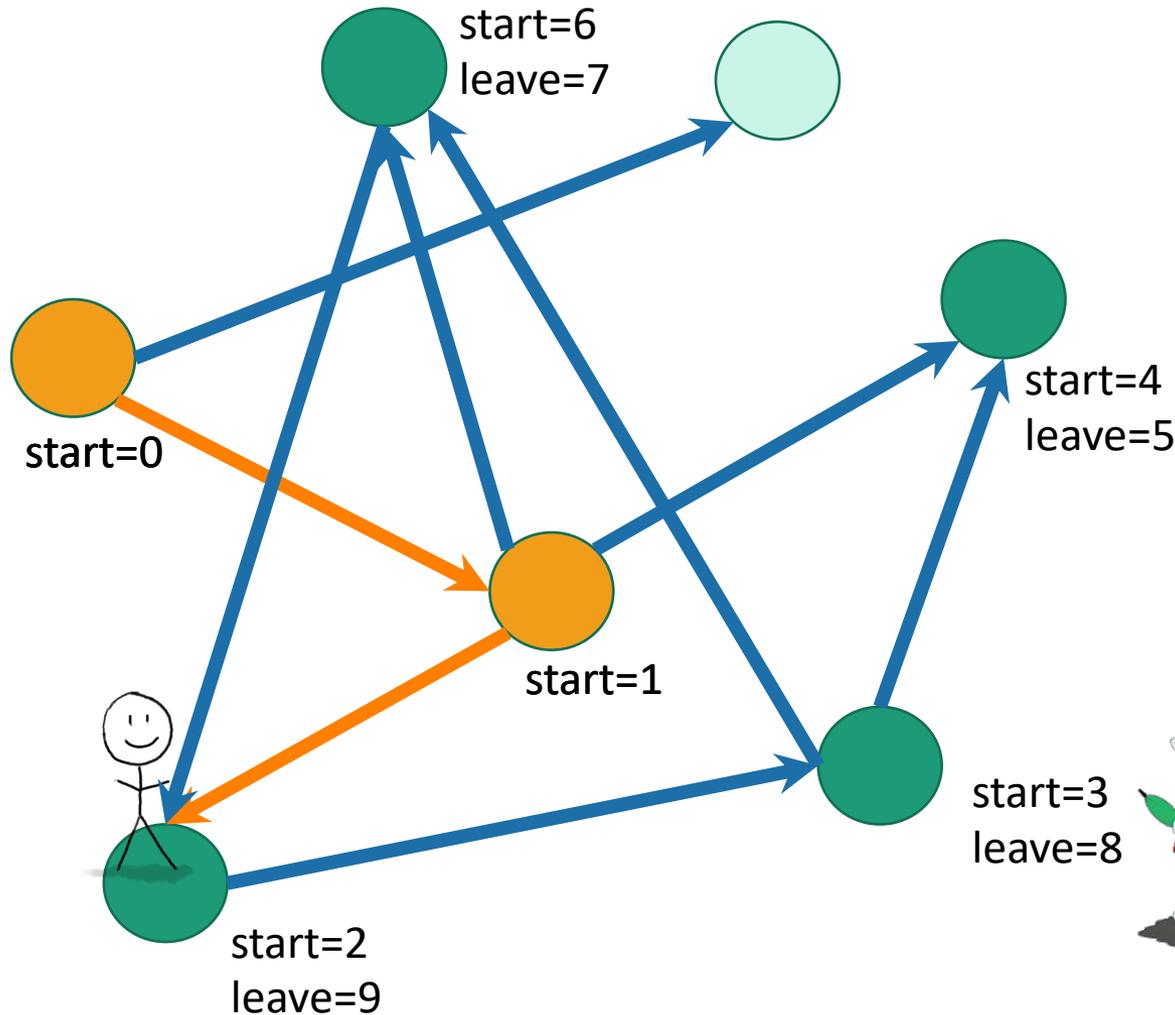Exploring a labyrinth with chalk and a piece of string

start=6
leave=7

Not been there yet

Been there, haven't explored all the paths out.

start=4
leave=5

start=0

Been there, have explored all the paths out.

start=1

start=3
leave=8

start=2

Recall we also keep track of **start** and **finish** times for every node.

18

Depth First Search
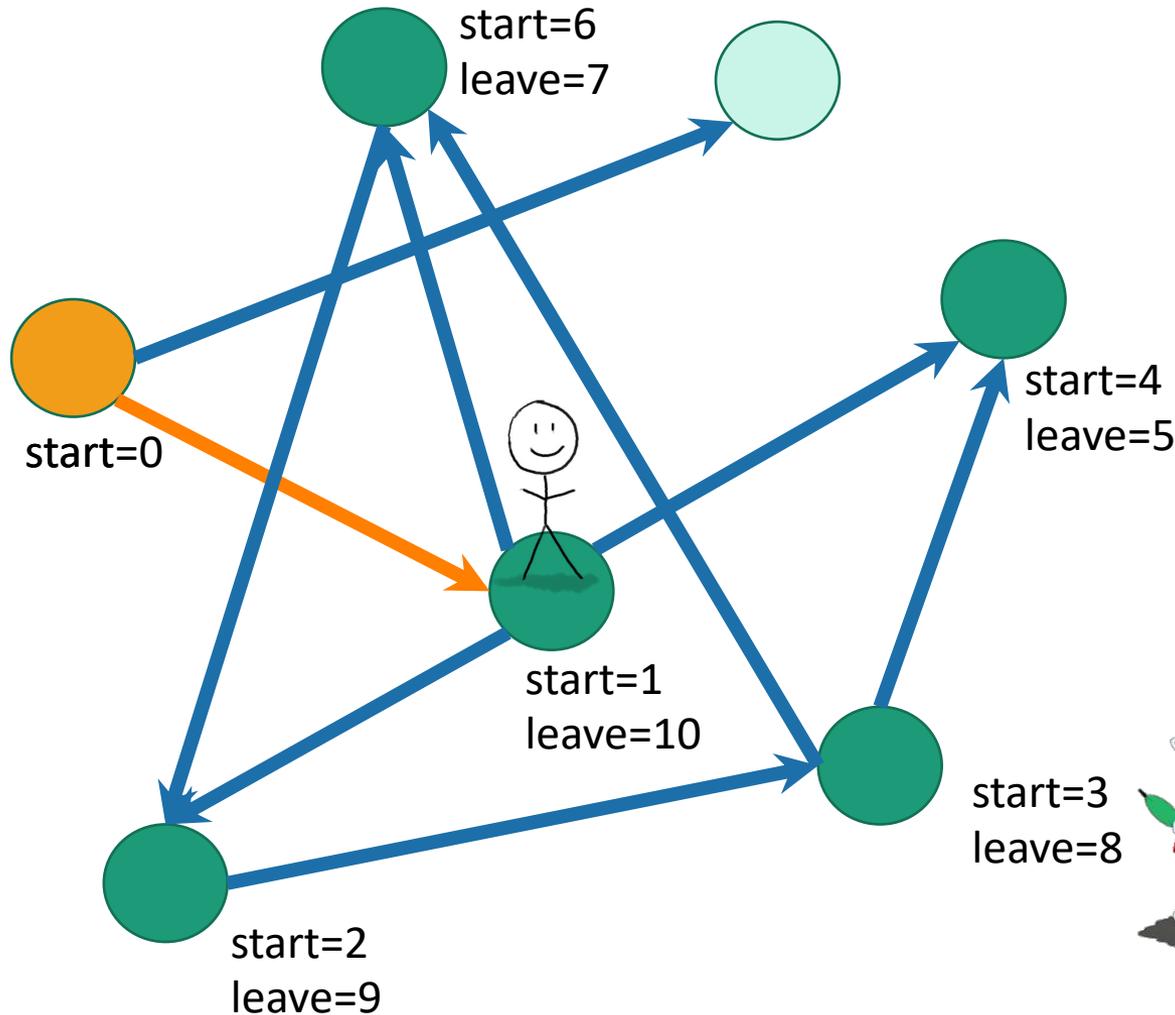Exploring a labyrinth with chalk and a piece of string

start=6
leave=7

Not been there yet

Been there, haven't explored all the paths out.

start=0

start=4
leave=5

Been there, have explored all the paths out.

start=1

start=3
leave=8

start=2
leave=9

Recall we also keep track of **start** and **finish** times for every node.

19

Depth First Search
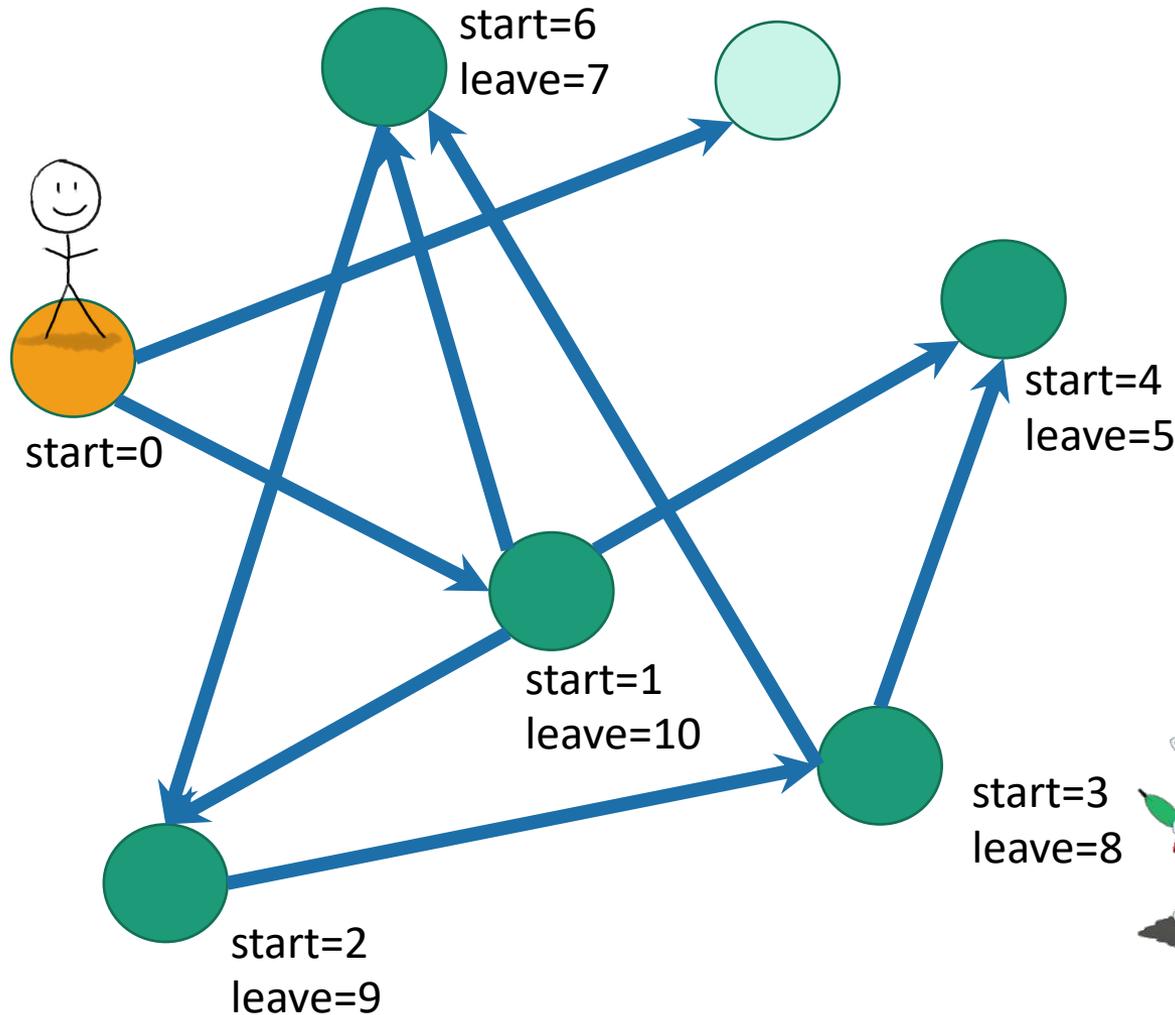## Exploring a labyrinth with chalk and a piece of string

start=6
leave=7

Not been there yet

Been there, haven't explored all the paths out.

start=0

start=4
leave=5

Been there, have explored all the paths out.

start=1
leave=10

start=3
leave=8

Recall we also keep track of **start** and **finish** times for every node.

start=2
leave=9

Depth First Search
Exploring a labyrinth with chalk and a piece of string

start=6
leave=7

Not been there yet

Been there, haven't explored all the paths out.

start=4
leave=5

start=0

Been there, have explored all the paths out.

start=1
leave=10

start=3
leave=8

start=2
leave=9

Recall we also keep track of **start** and **finish** times for every node.

Depth First Search
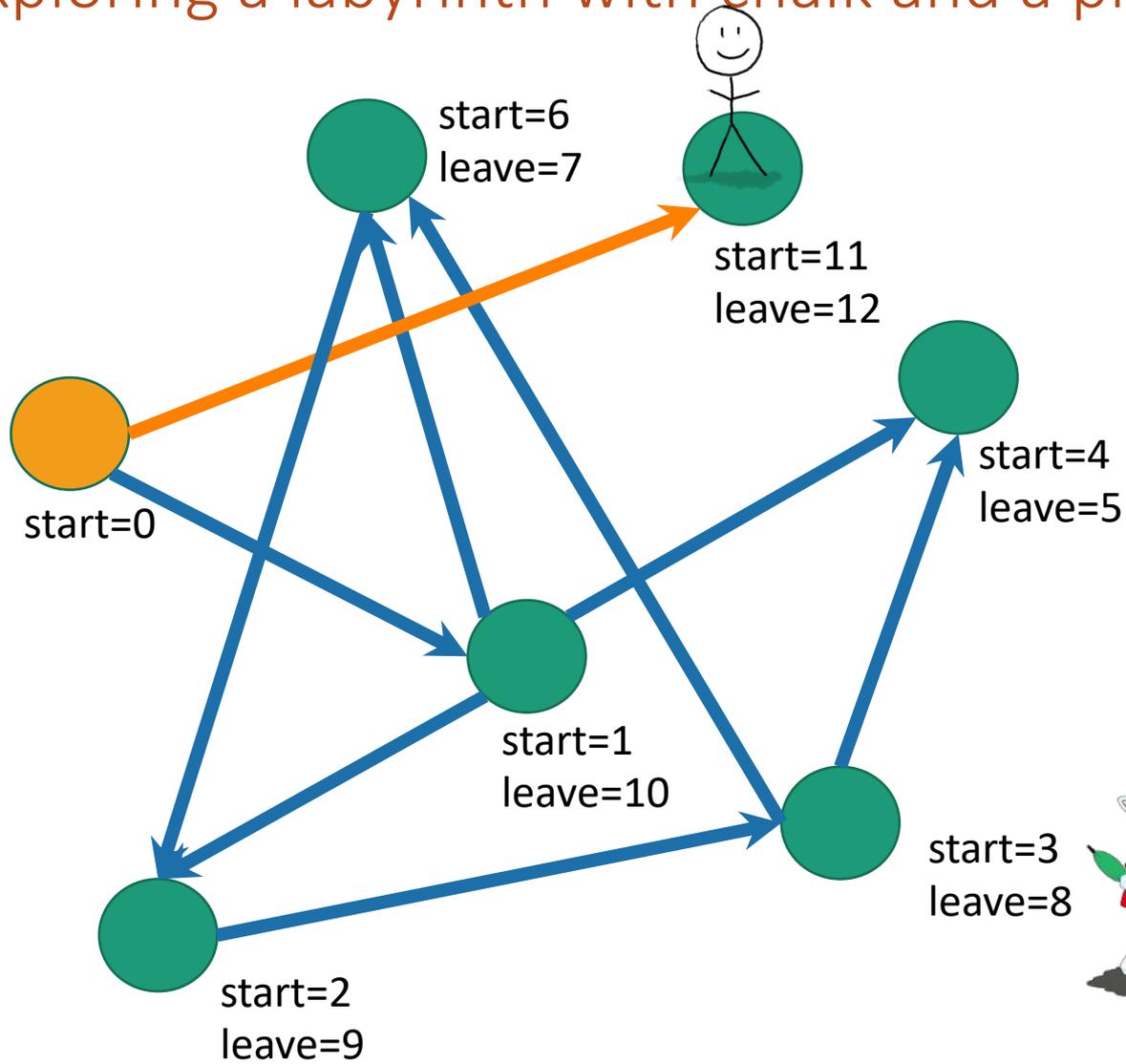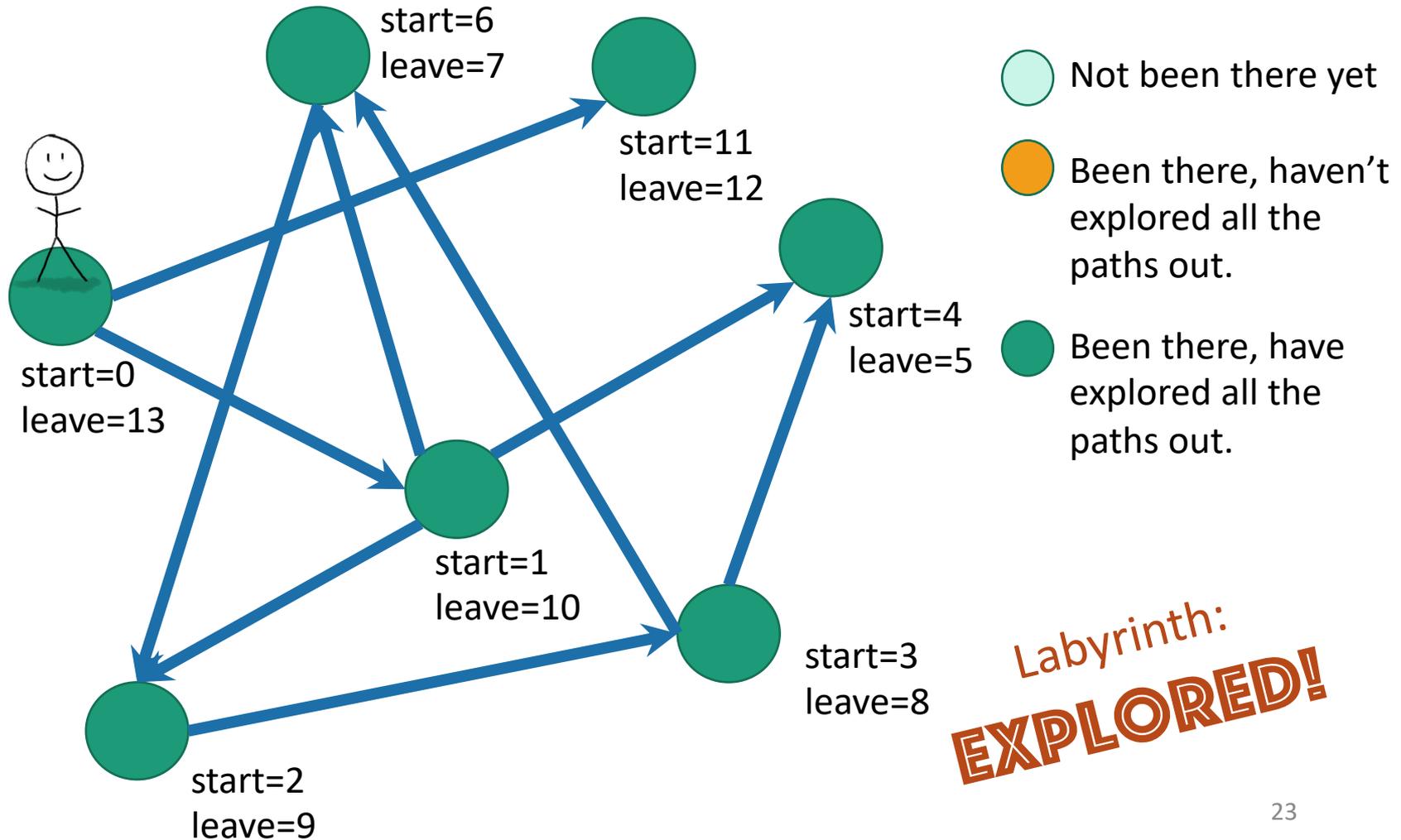## Exploring a labyrinth with chalk and a piece of string

start=6
leave=7

start=11
leave=12

Not been there yet

Been there, haven't explored all the paths out.

start=0

start=4
leave=5

Been there, have explored all the paths out.

start=1
leave=10

start=3
leave=8

Recall we also keep track of **start** and **finish** times for every node.

start=2
leave=9
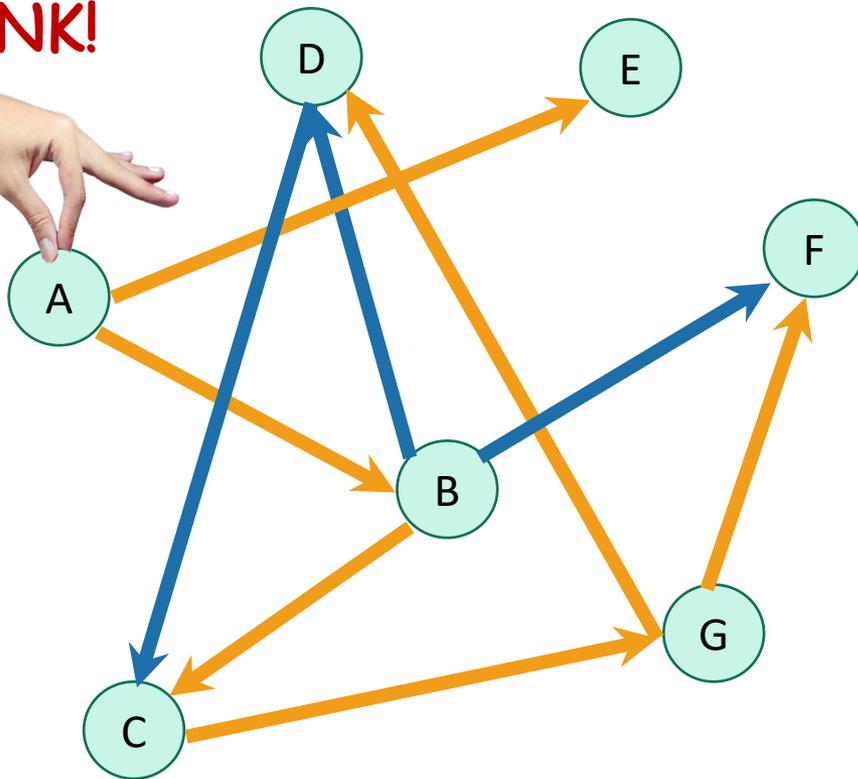
Depth First Search
Exploring a labyrinth with chalk and a piece of string
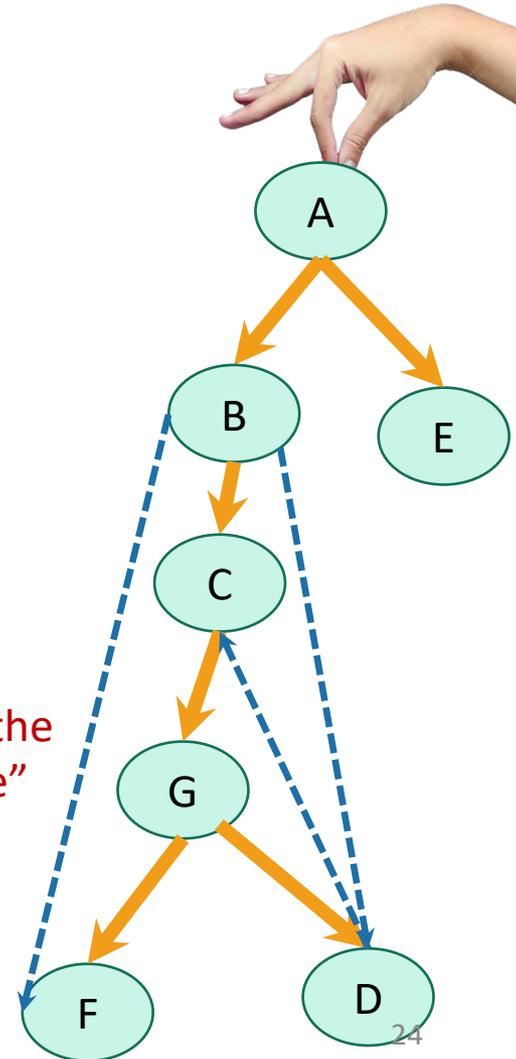

23

Depth first search
implicitly creates a tree on everything you can reach
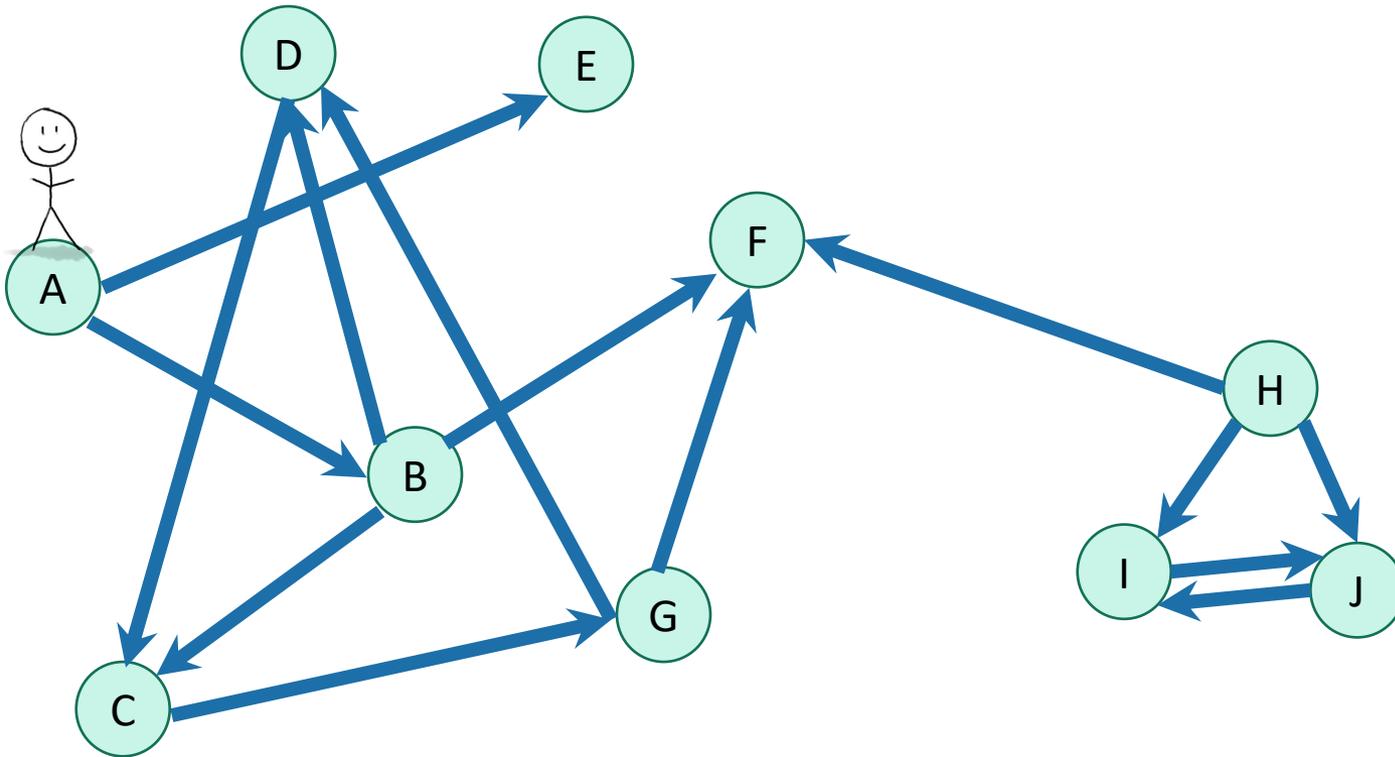
YOINK!

Call this the "DFS tree"

When you can't reach everything

- Run DFS repeatedly to get a depth-first forest

When you can't reach everything

- Run DFS repeatedly to get a depth-first forest

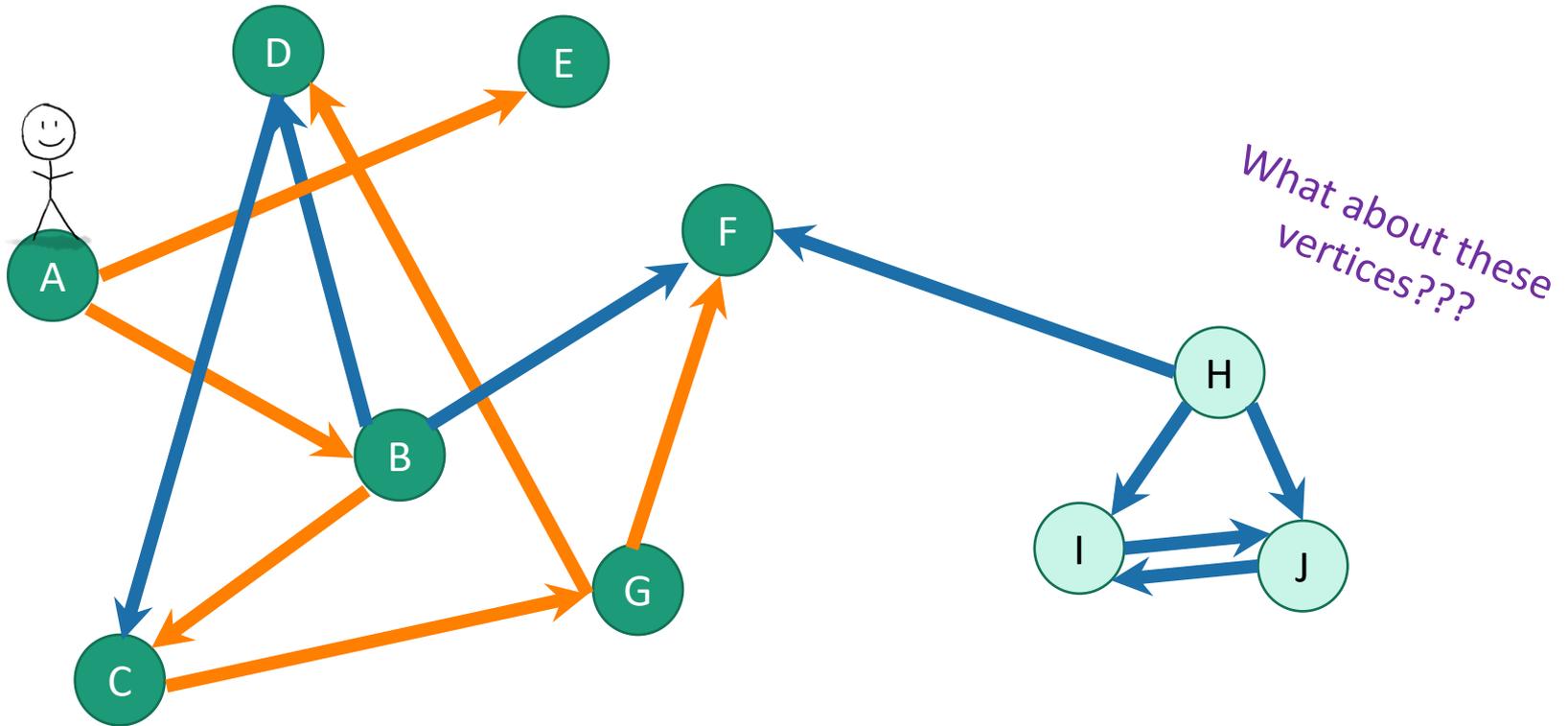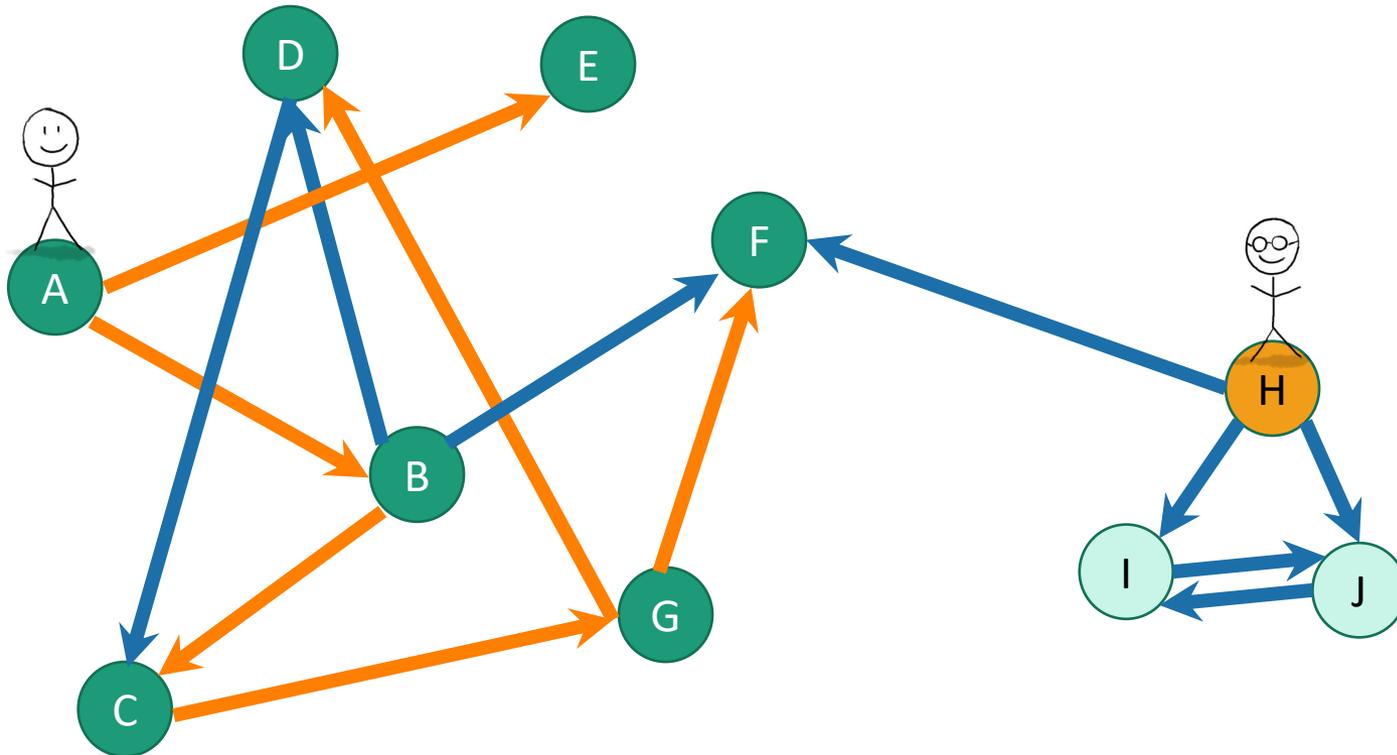# When you can't reach everything

- Run DFS repeatedly to get a depth-first forest

When you can't reach everything
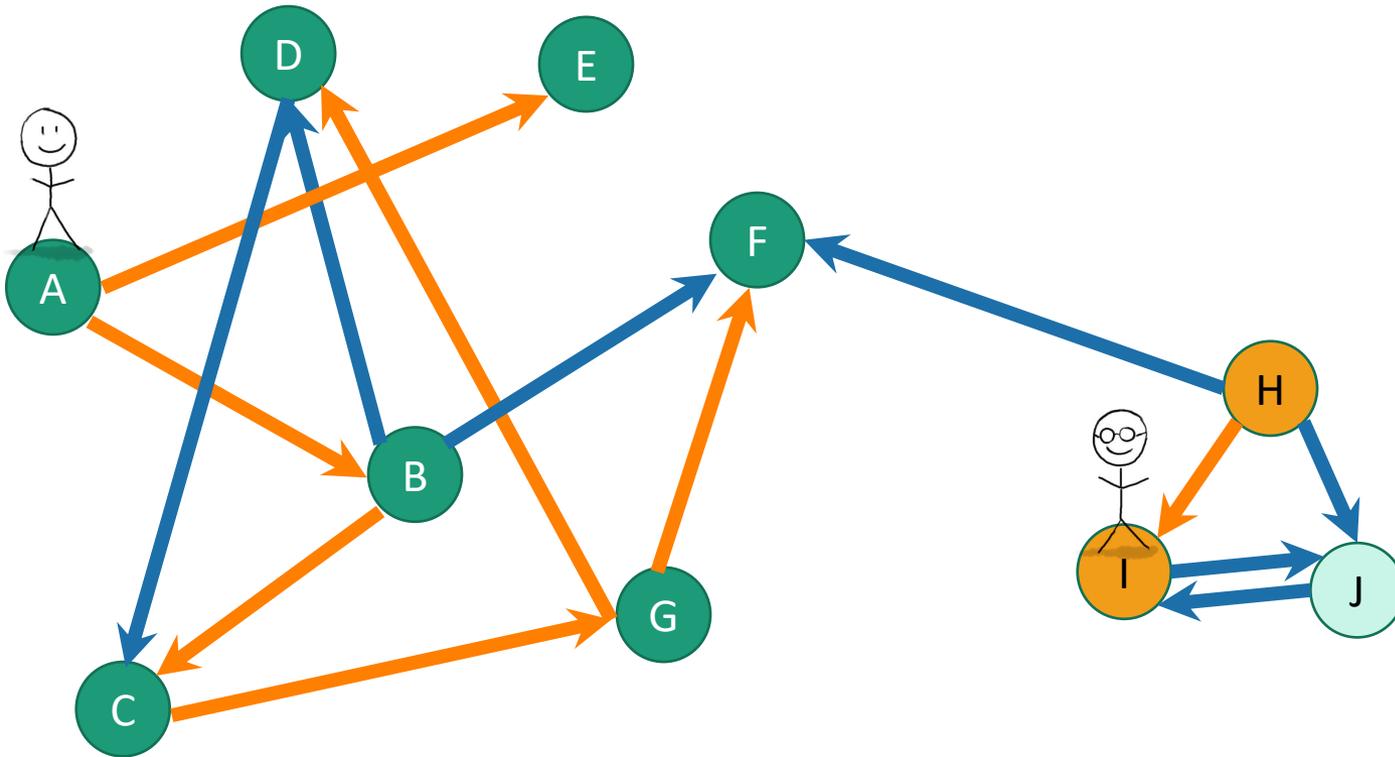
- Run DFS repeatedly to get a depth-first forest

When you can't reach everything

- Run DFS repeatedly to get a depth-first forest

When you can't reach everything
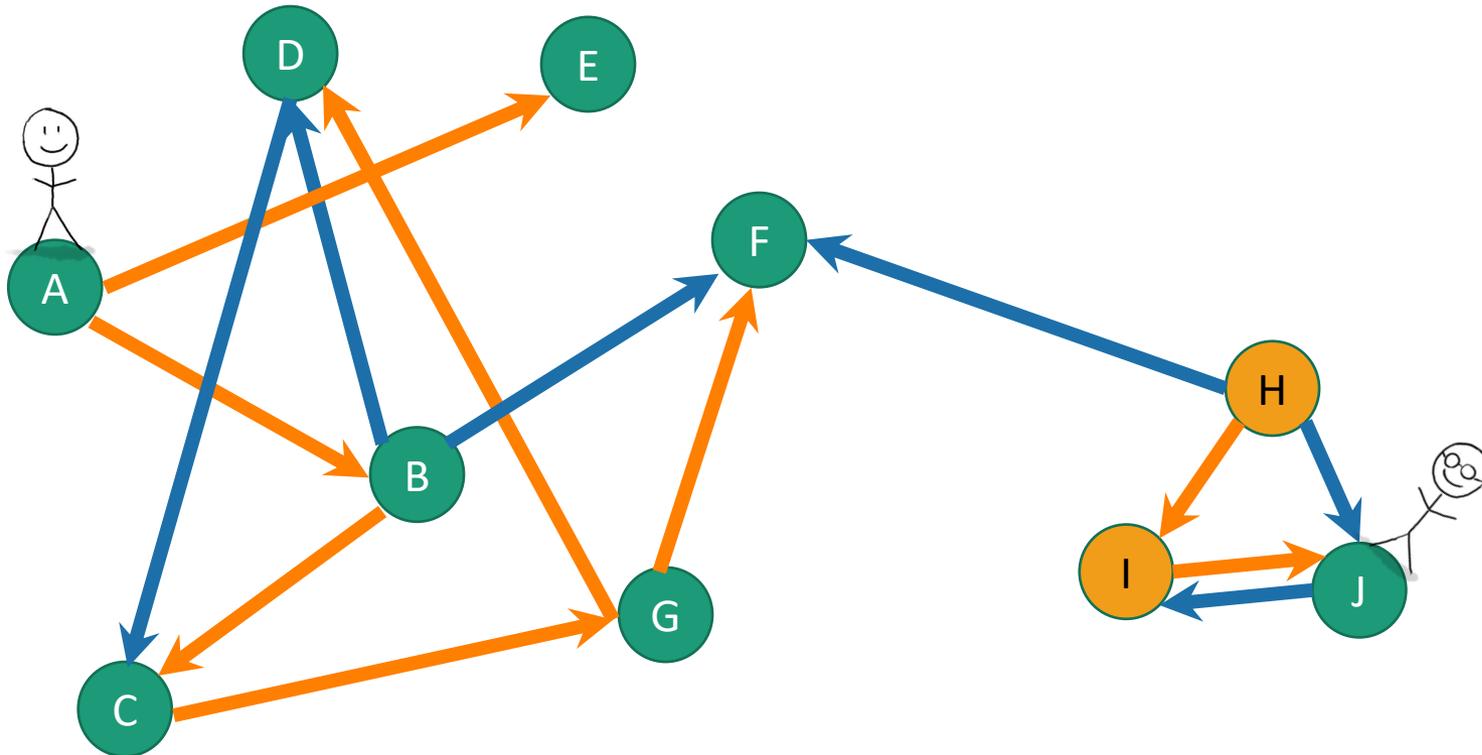
- Run DFS repeatedly to get a depth-first forest

When you can't reach everything
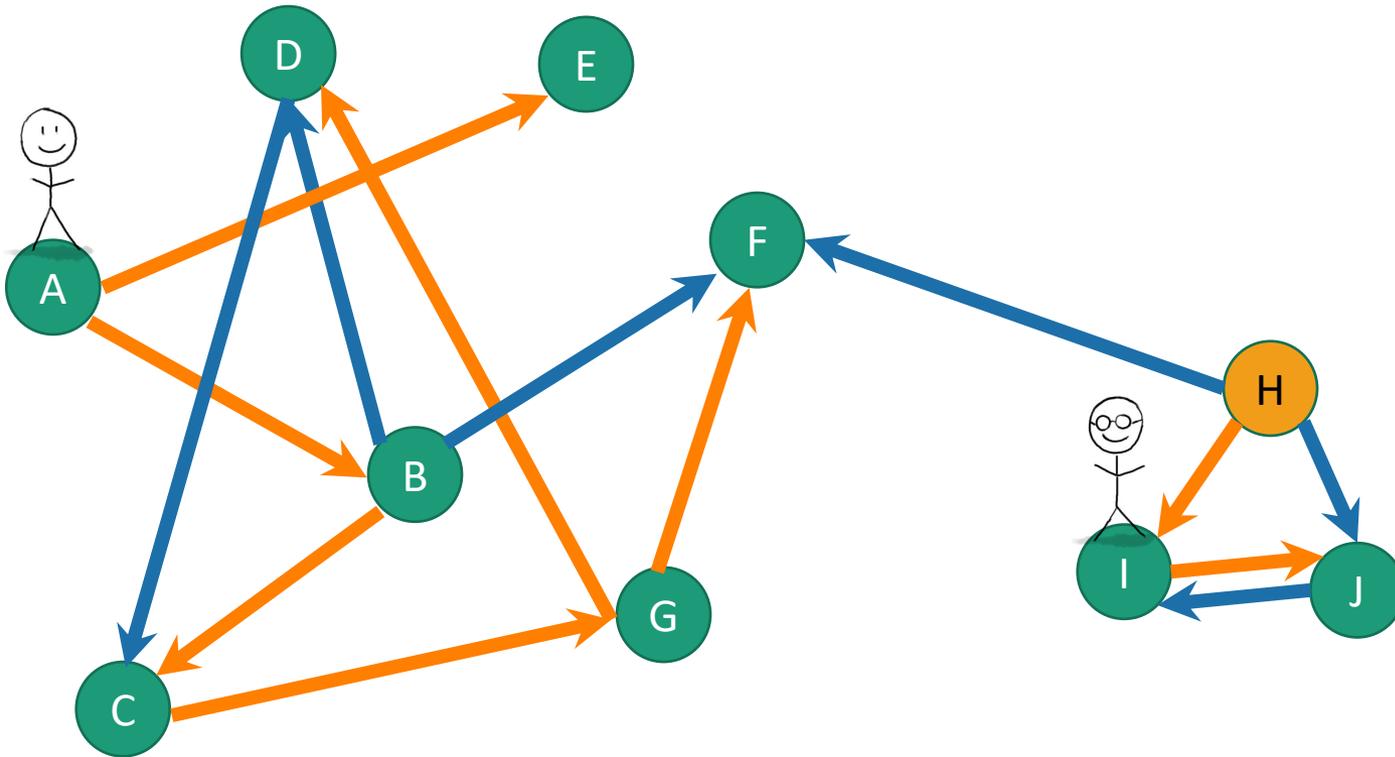
- Run DFS repeatedly to get a depth-first forest

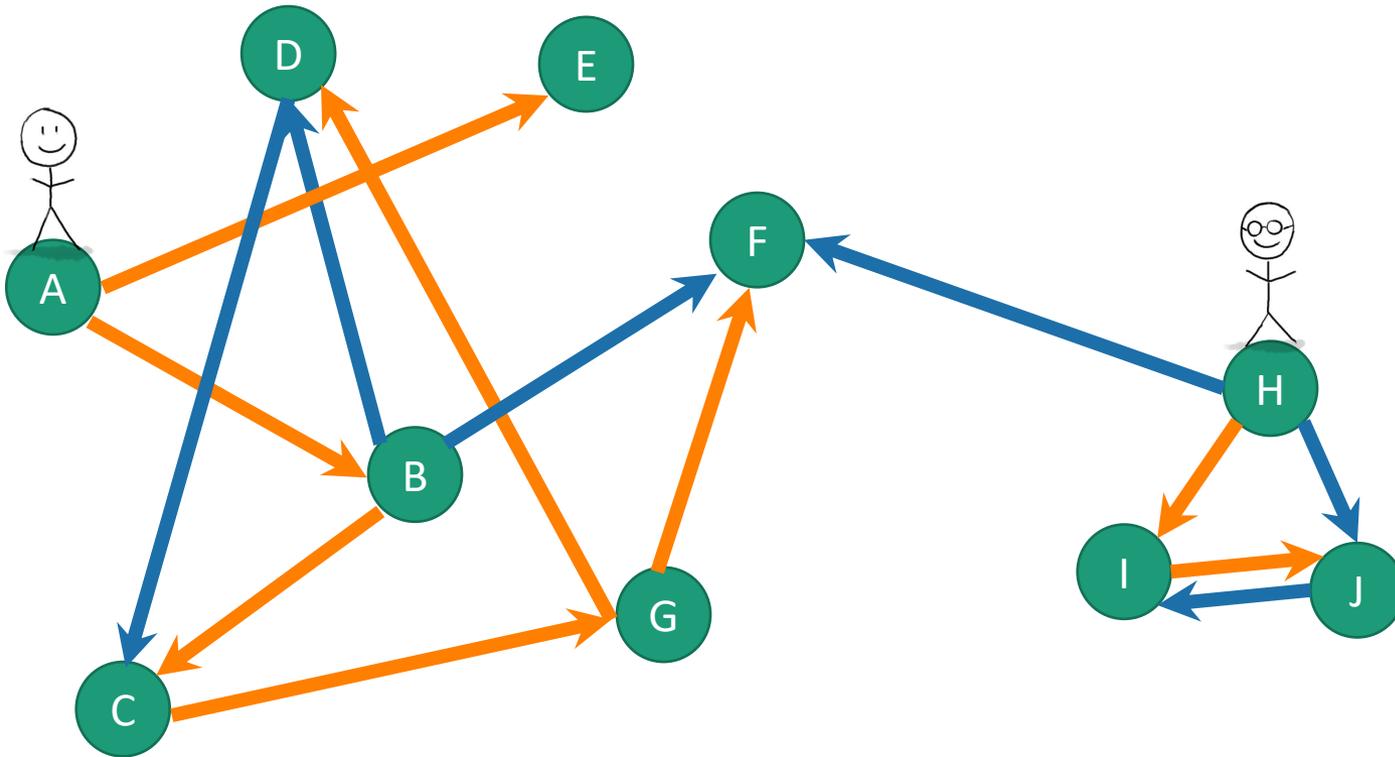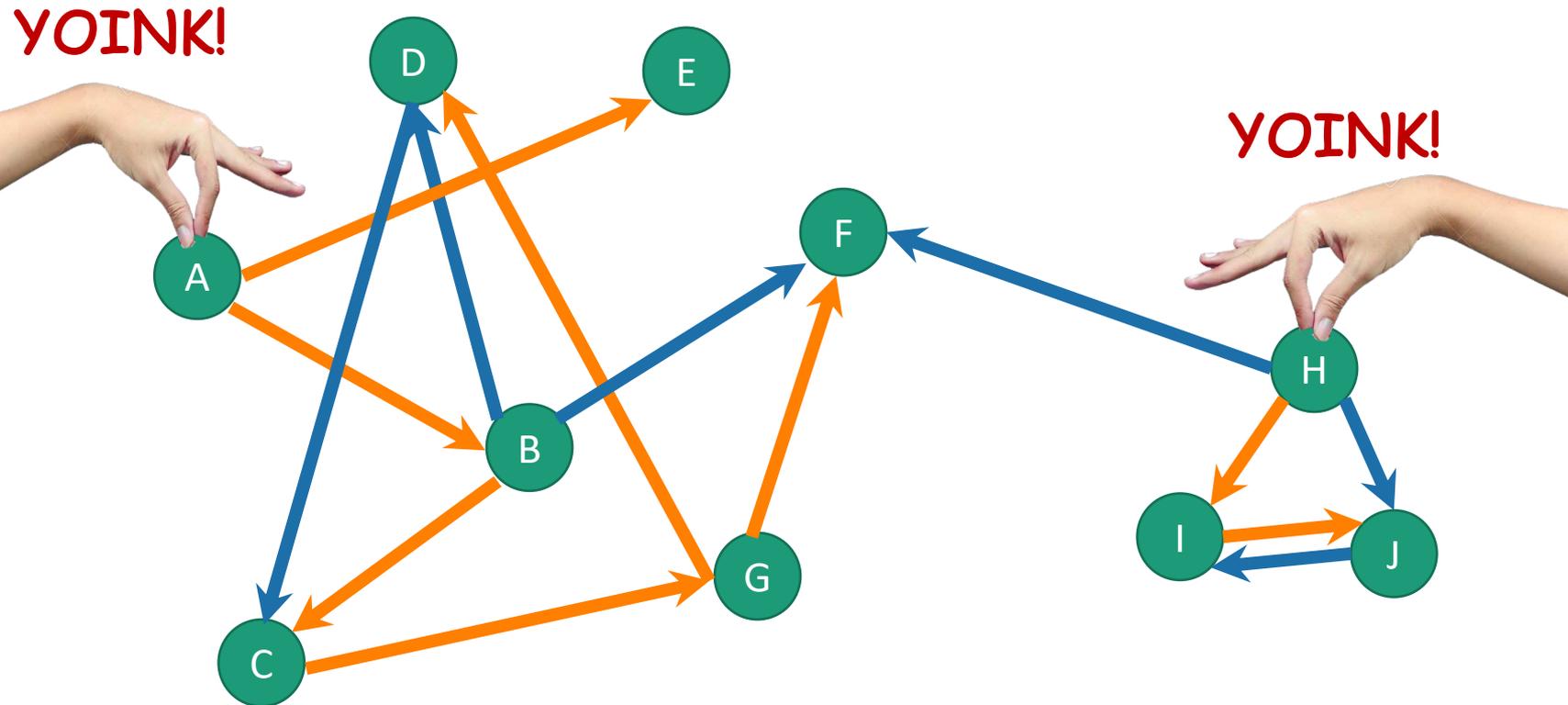# When you can't reach everything

- Run DFS repeatedly to get a depth-first forest

When you can't reach everything

- Run DFS repeatedly to get a depth-first search forest

The DFS forest is made up of DFS trees
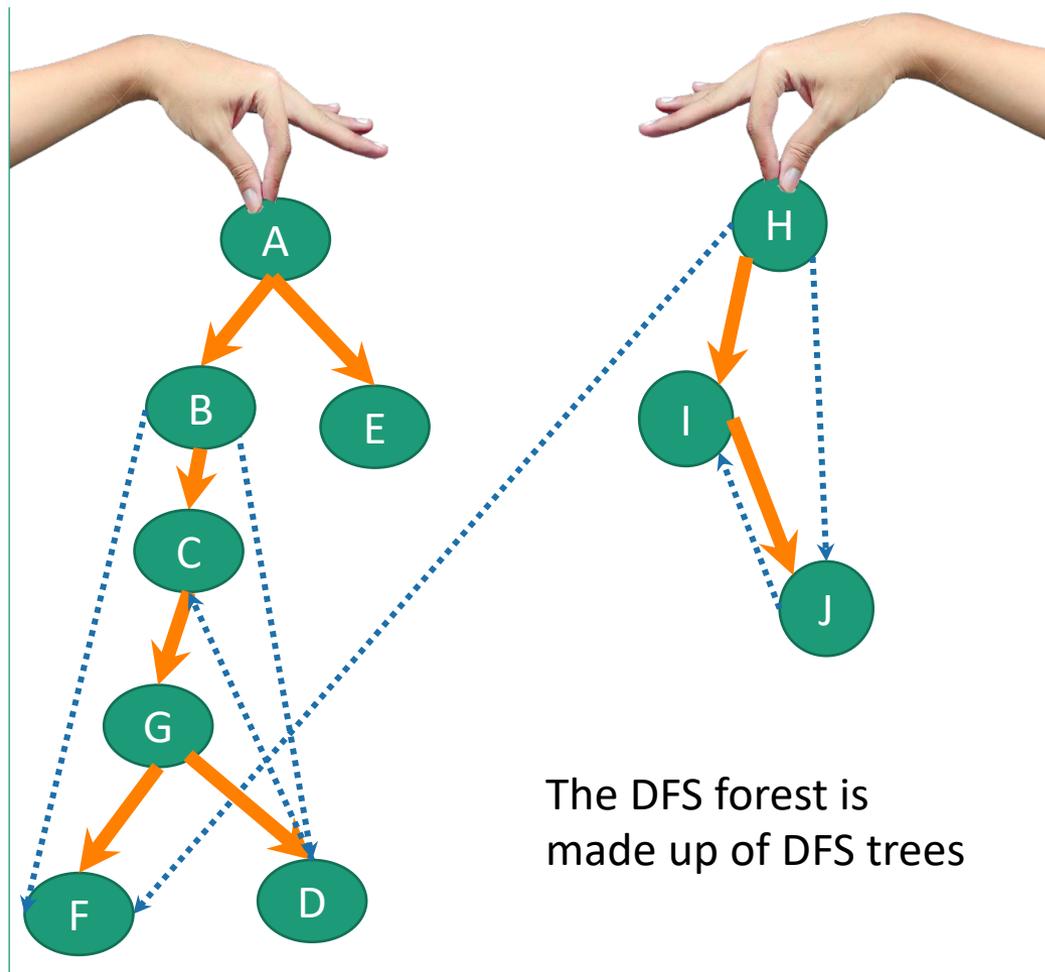
Recall:

(Works the same with DFS forests)

DFS tree

- If v is a descendent of w in this tree:

  w.start   v.start   v.finish   w.finish

  timeline

- If w is a descendent of v in this tree:

  v.start   w.start   w.finish   v.finish

- If neither are descendants of each other:

  v.start   v.finish   w.start   w.finish

  (or the other way around)
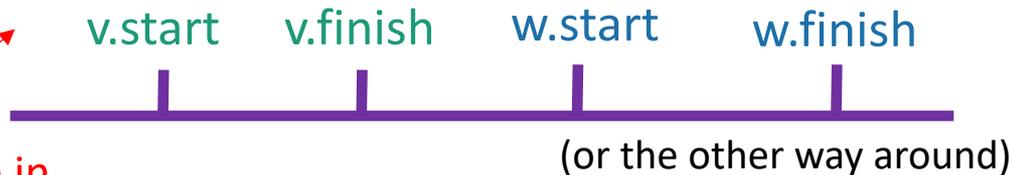
If v and w are in different trees, it's always this last one.

34
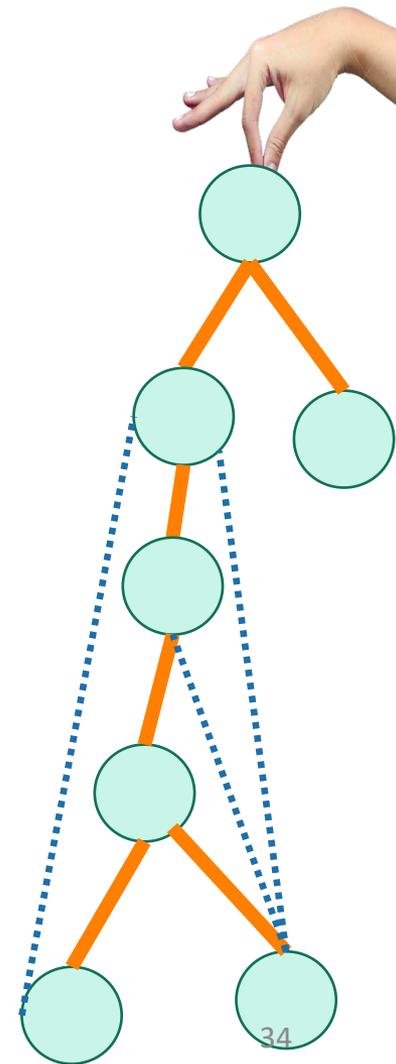
Enough of review

Strongly connected components

Strongly connected components

- A directed graph G = (V,E) is **strongly connected** if:
- for all v,w in V:
    - there is a path from v to w and
    - there is a path from w to v.
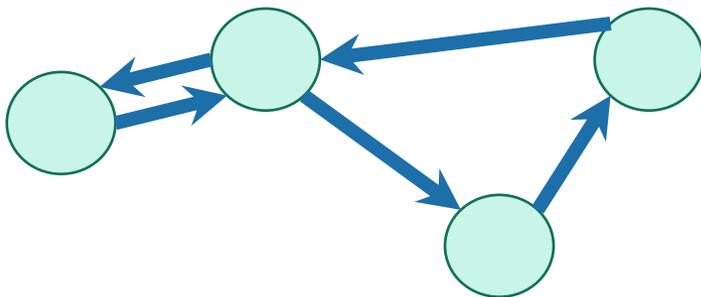
strongly connected
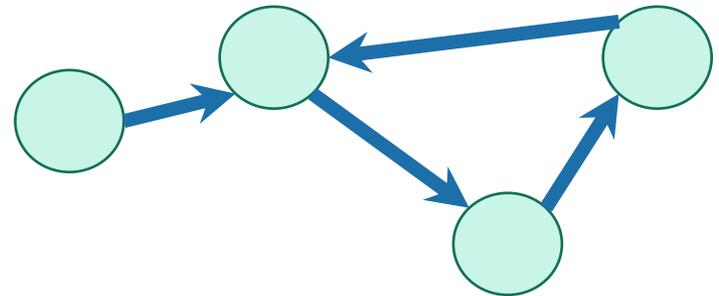
not strongly connected

We can decompose a graph into strongly connected components (SCCs)

(Definition by example)
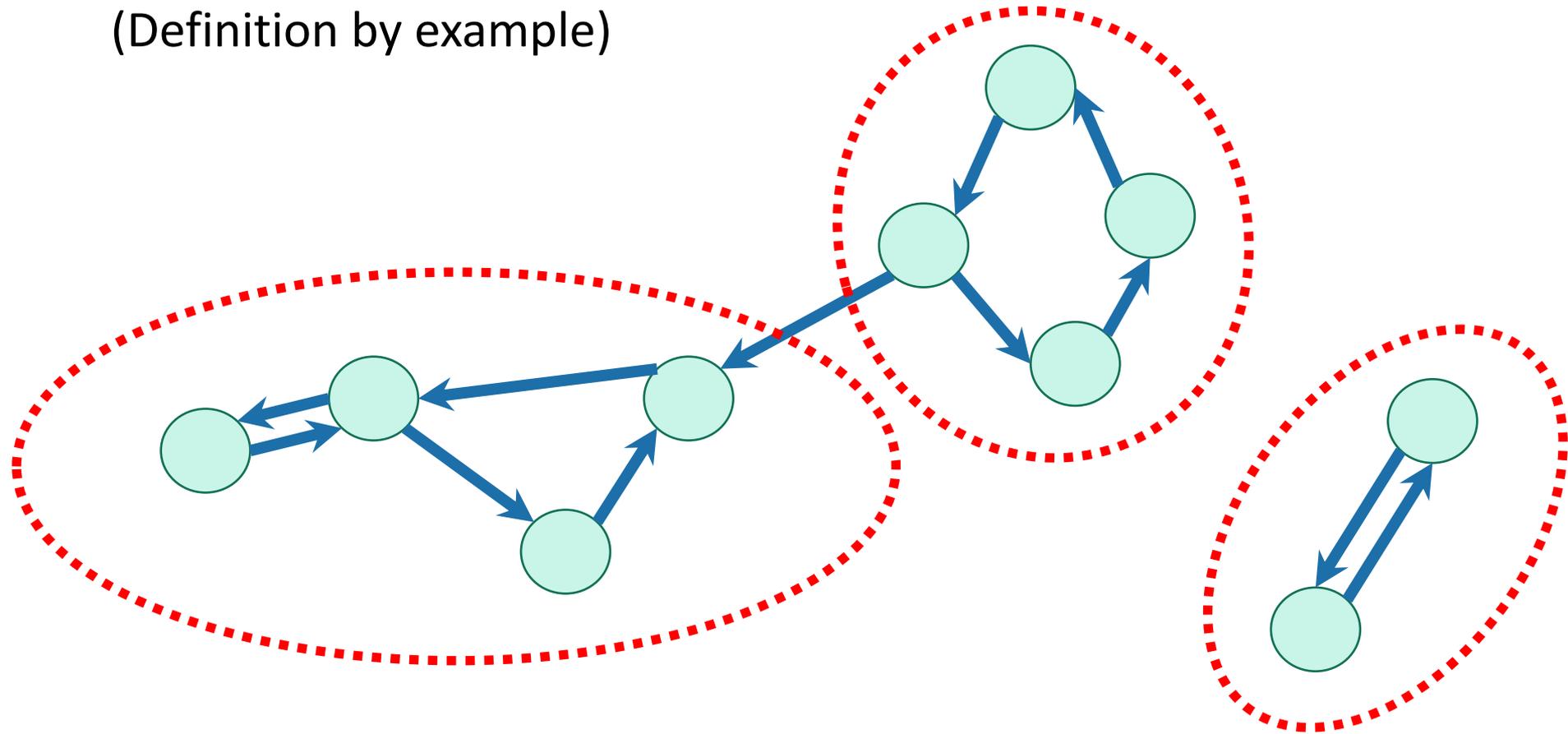
Definition by definition: The SCCs are the equivalence classes under the "are mutually reachable" equivalence relation.

Why do we care about SCCs?

Consider the internet:

stanford.edu

wikipedia.org

nytimes.com

berkeley.edu

4chan.org

reddit.com

google image search for "puppies"

Google terms and conditions
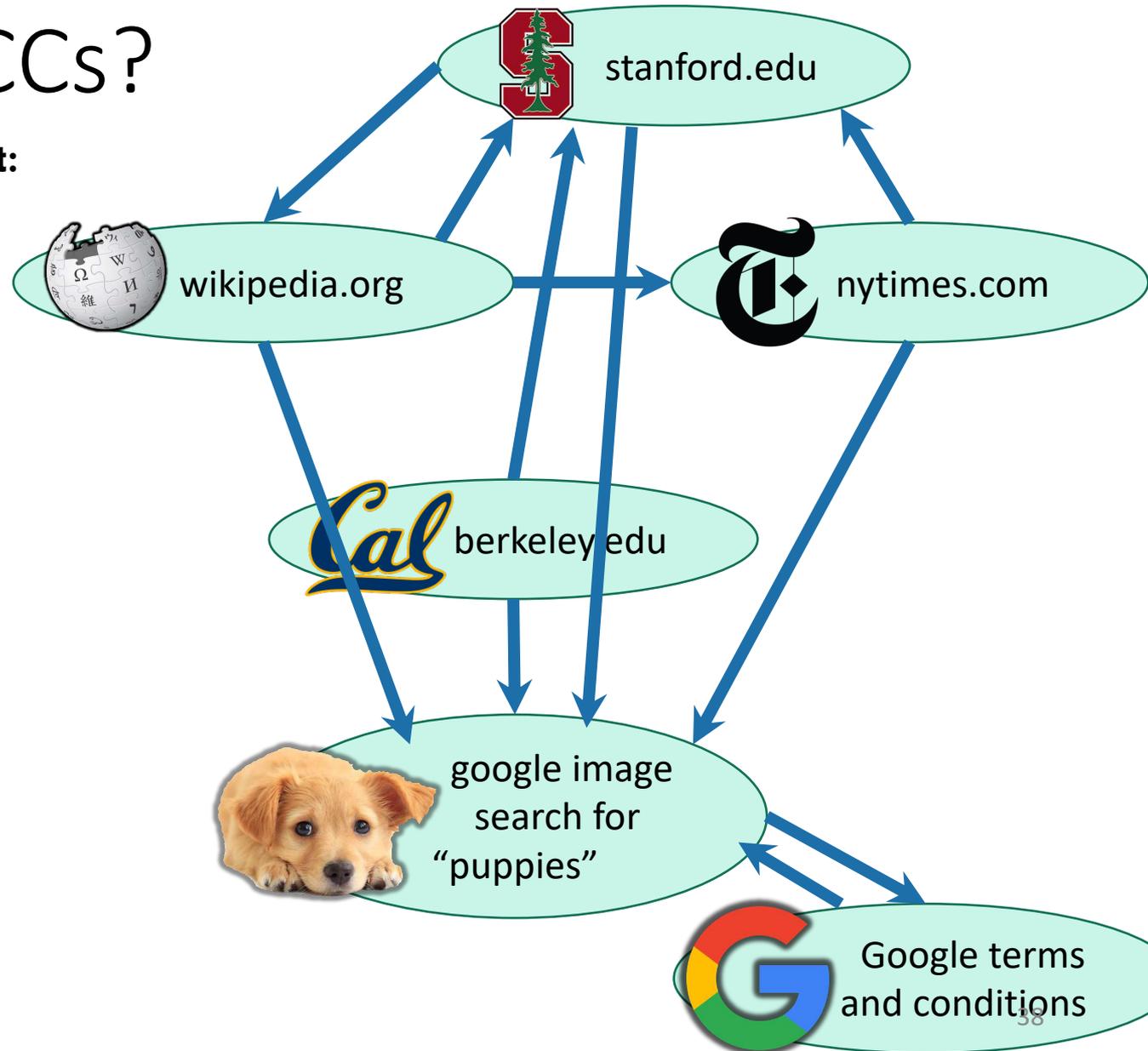
Let's ignore this corner of the internet for now...but everything today works fine if the graph is disconnected.

Why do we care about SCCs?

Consider the internet:

stanford.edu

wikipedia.org

nytimes.com

berkeley.edu

google image search for "puppies"

Google terms and conditions
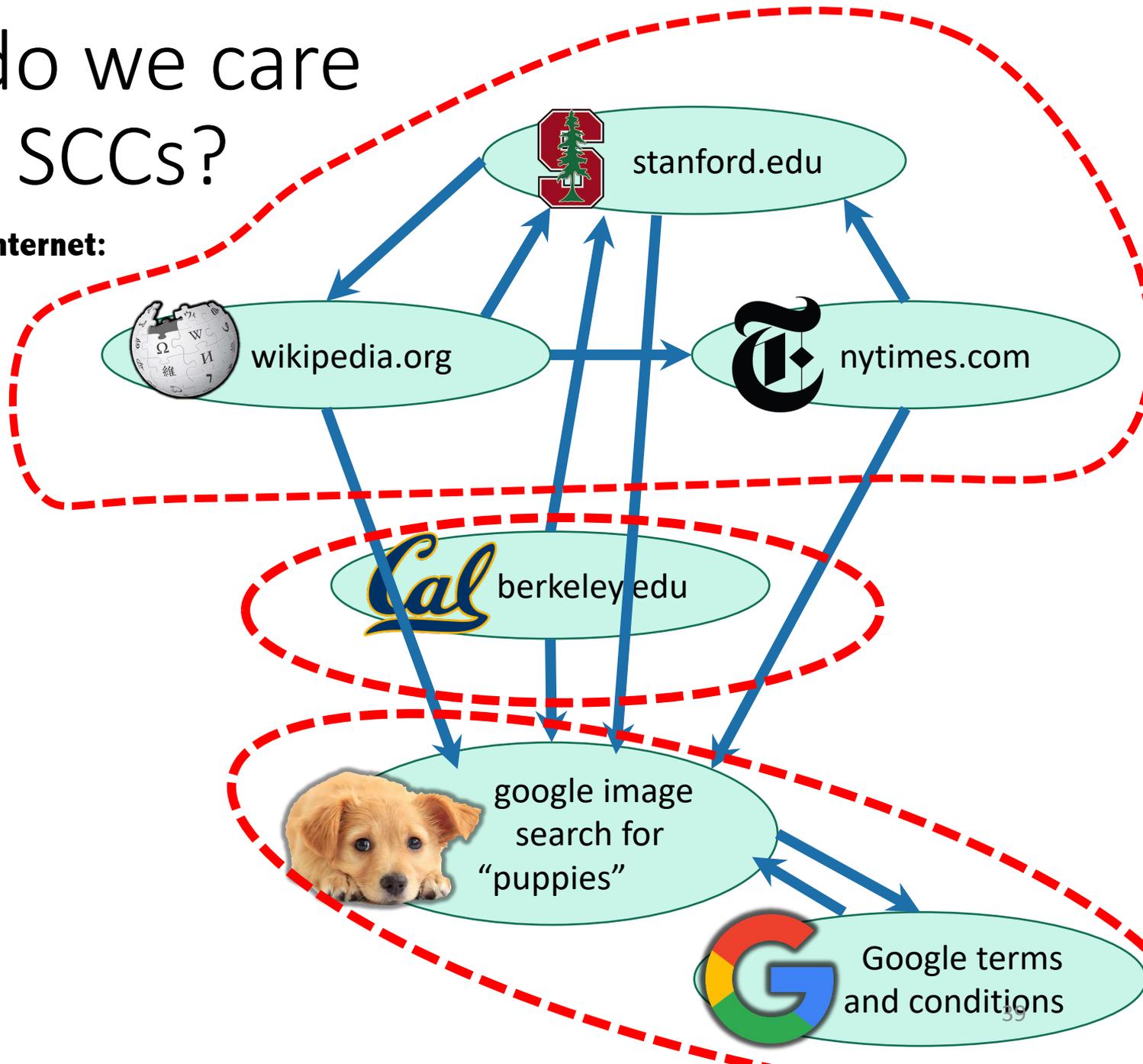
Aside: What the internet graph actually looks like
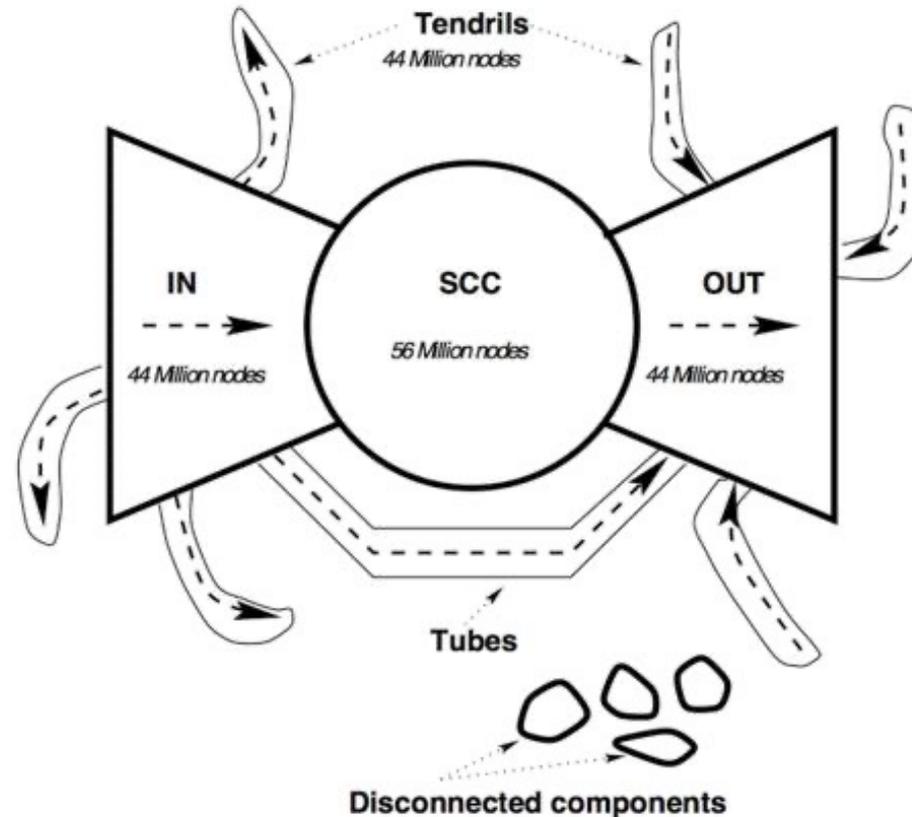
- Circa 2000[1]:

[1] "Graph Structure in the Web." Broder et al. *Computer Networks*. 2000.

[2] "Local Bow-tie Structure of the Web." Fujita et al. *Applied Network Science*. 2019.

Aside: What the internet graph actually looks like
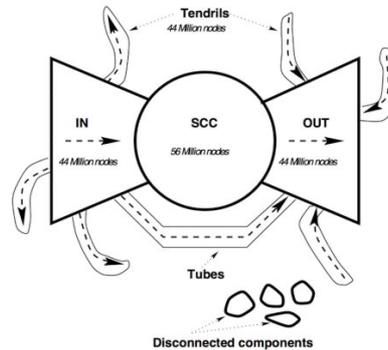
- Circa 2000[1]:

- Circa 2019[2]:
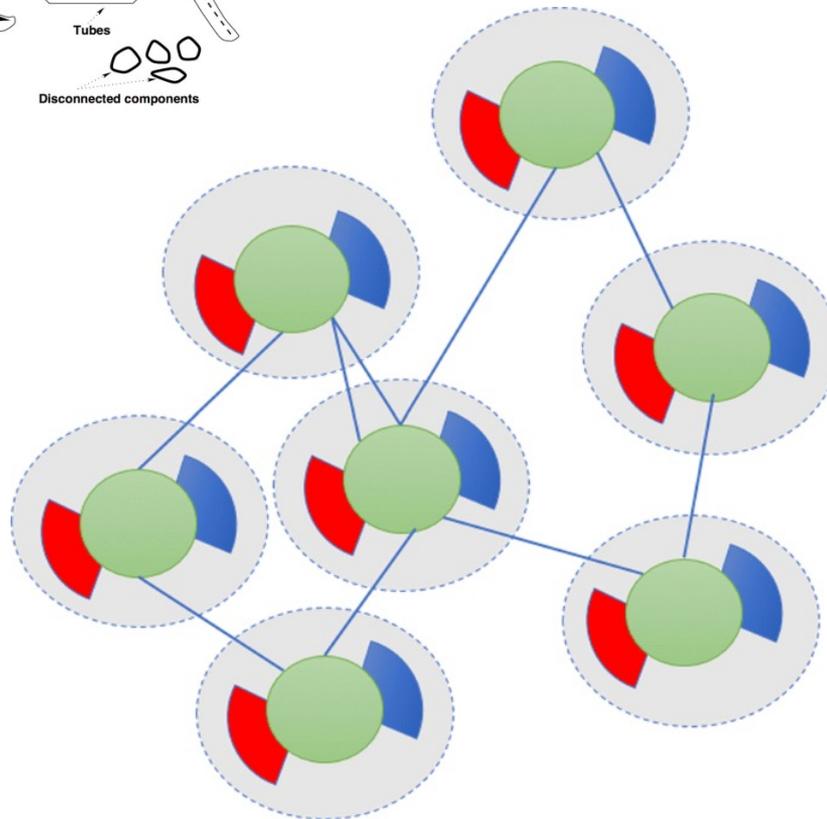
[1] "Graph Structure in the Web." Broder et al. *Computer Networks*. 2000.

[2] "Local Bow-tie Structure of the Web." Fujita et al. *Applied Network Science*. 2019.

Why do we care about SCCs?

Consider the internet:

stanford.edu

wikipedia.org

nytimes.com

berkeley.edu

google image search for "puppies"

Google terms and conditions

42
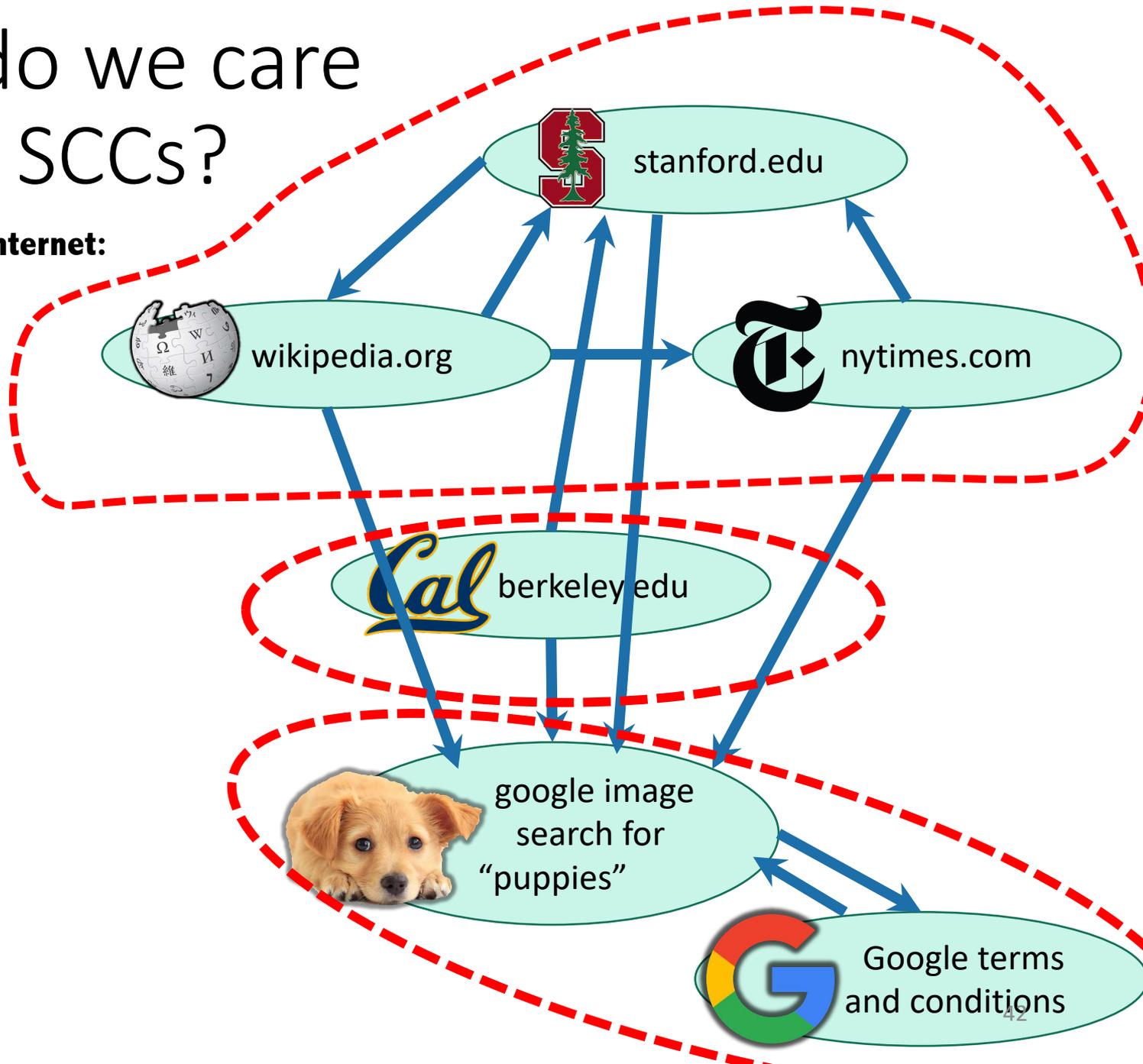
Why do we care about SCCs?

- Strongly connected components tell you about **communities.**

- Lots of graph algorithms only make sense on SCCs.
  - (So sometimes we want to find the SCCs as a first step)
  - Eg: I know an economics prof. who has to first break up their labor market data into SCCs in order to make sense of it.

How to find SCCs?

- Consider all possible decompositions and check.

- Something like…
  - Run DFS a bunch to find out which u's and v's belong in the same SCC
  - Aggregate that information to figure out the SCCs

Come up with a straightforward way to use DFS
to find SCCs. What's the running time?
More than n^2 or less than n^2?

Think: 2 minutes.
Pair+Share:1 minute

44
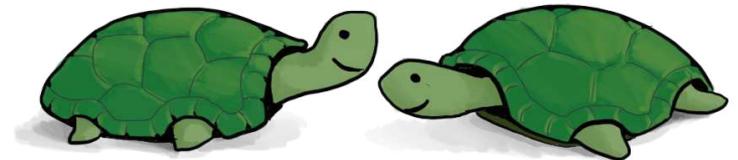
One straightforward solution

This will not be our final solution so don't worry too much about it…

- SCCs = [ ]
- For each u:
  - Run DFS from u
  - For each vertex v that u can reach:
    - If v is in an SCC we've already found:
      - Run DFS from v to see if you can reach u
      - If so, add u to v's SCC
      - Break
  - If we didn't break, create a new SCC which just contains u.

Running time AT LEAST $\Omega(n^2)$, no matter how smart you are about implementing the rest of it…

Today

- We will see how to find strongly connected components in time O(n+m)

- !!!!!

- This is called Kosaraju's algorithm.
  - Heads up: the textbook has a slightly different presentation.

Pre-Lecture exercise

• What do you get when you run DFS from A?

• What about from D?
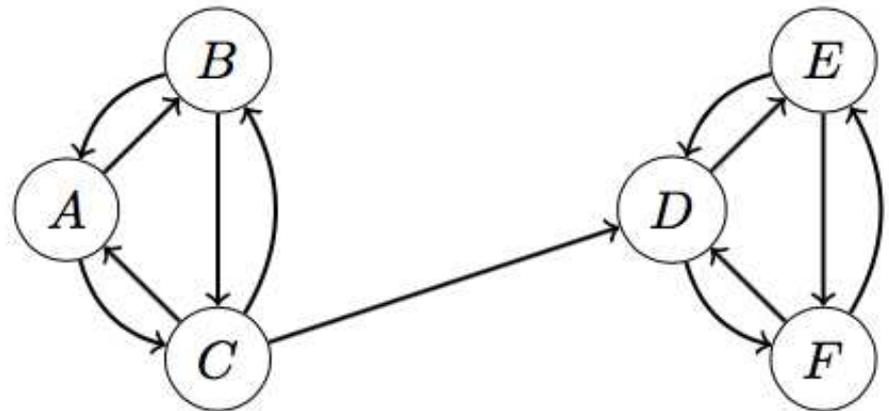
• Suggested algorithm: run DFS from the "right" place to identify SCCs

• Issue: what is the "right" place?

Algorithm

Running time: O(n + m)

- Do DFS to create a DFS forest.
  - Choose starting vertices in any order.
  - Keep track of finishing times.
- Reverse all the edges in the graph.
- Do DFS again to create **another DFS forest**.
  - This time, order the nodes in the reverse order of the finishing times that they had from the first DFS run.
- The SCCs are the different trees in the **second DFS forest.**
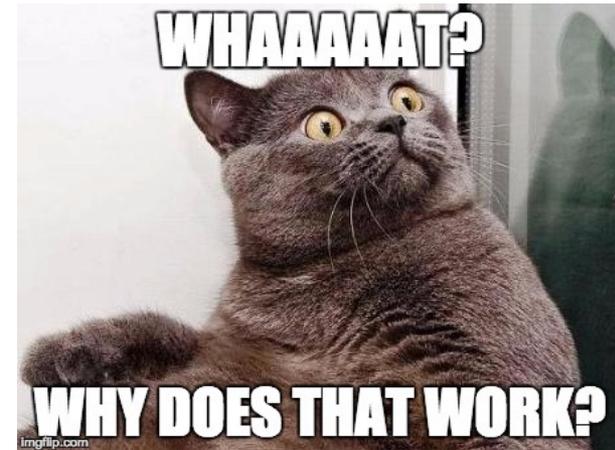
Look, it works!

- (See IPython notebook)

```
In [4]: print(G)

        CS161Graph with:
                Vertices:
                Stanford,Wikipedia,NYTimes,Berkeley,Puppies,Google,
                 Edges:
                (Stanford,Wikipedia) (Stanford,Puppies) (Wikipedia,Stanford) (Wik
        ipedia,NYTimes) (Wikipedia,Puppies) (NYTimes,Stanford) (NYTimes,Puppies)
        (Berkeley,Stanford) (Berkeley,Puppies) (Puppies,Google) (Google,Puppies)
```

```
In [5]: SCCs = SCC(G, False)
        for X in SCCs:
            print ([str(x) for x in X])

        ['Berkeley']
        ['Stanford', 'NYTimes', 'Wikipedia']
        ['Puppies', 'Google']
```

But let's break that down a bit...

Example
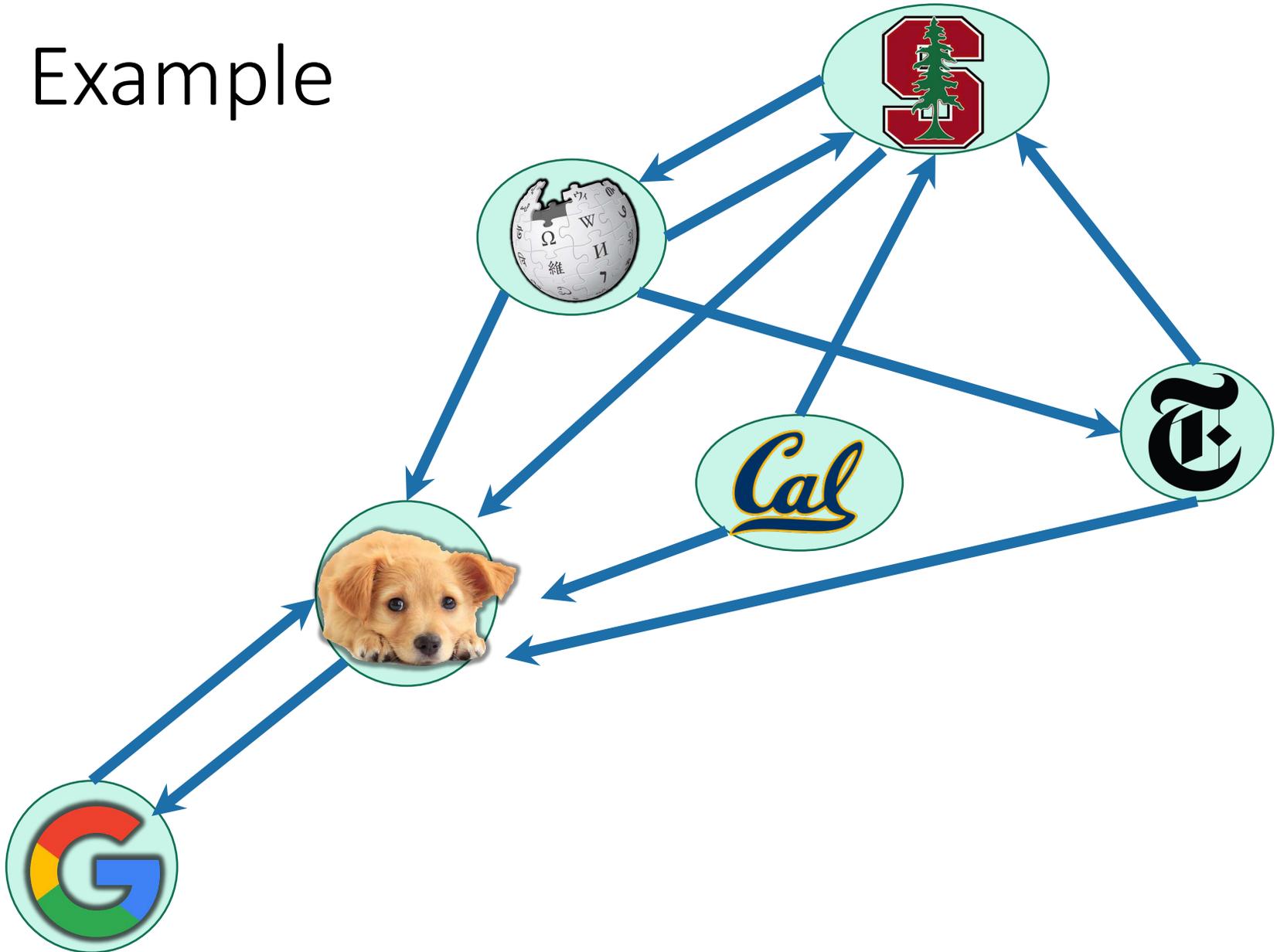
Example
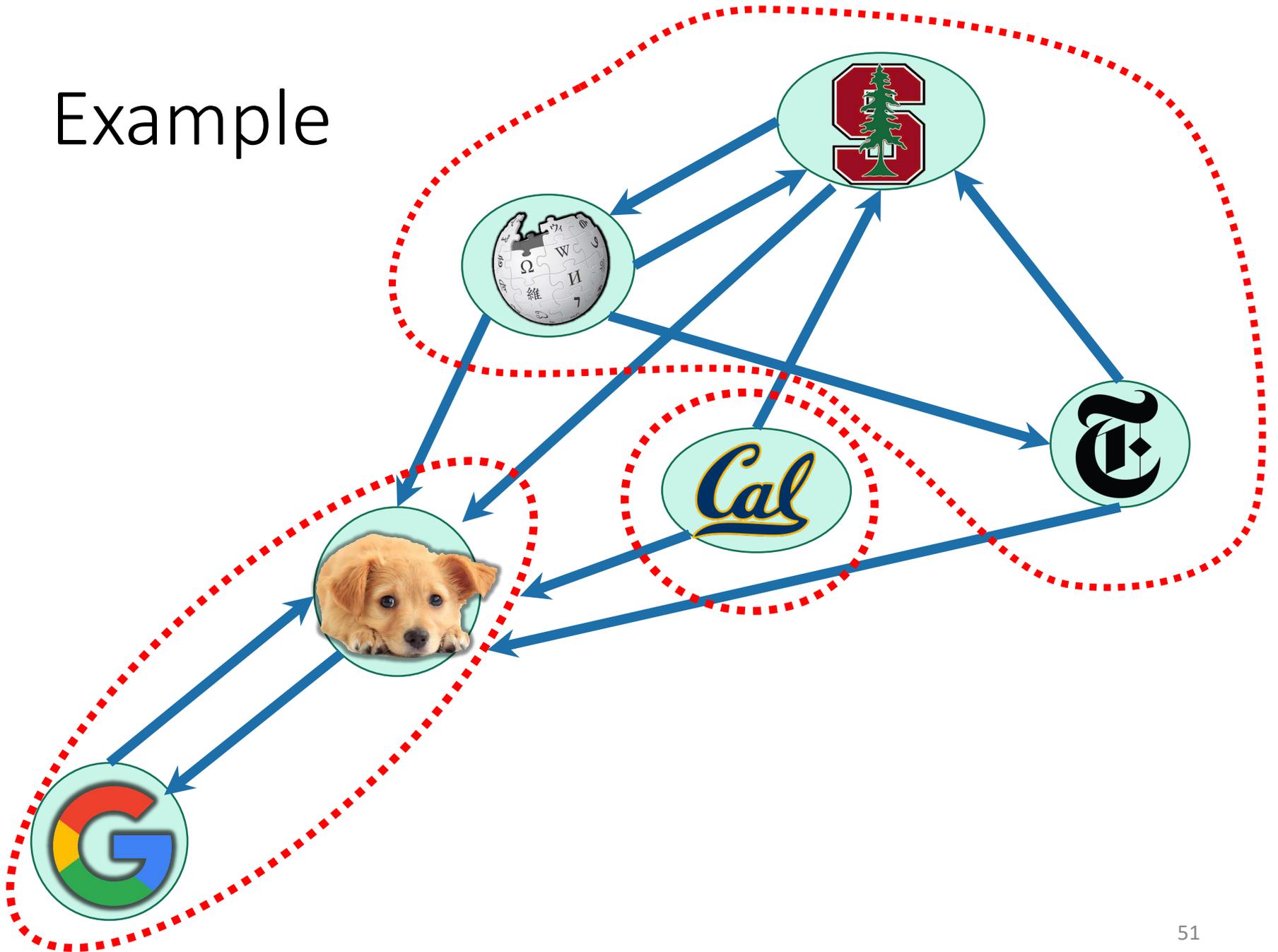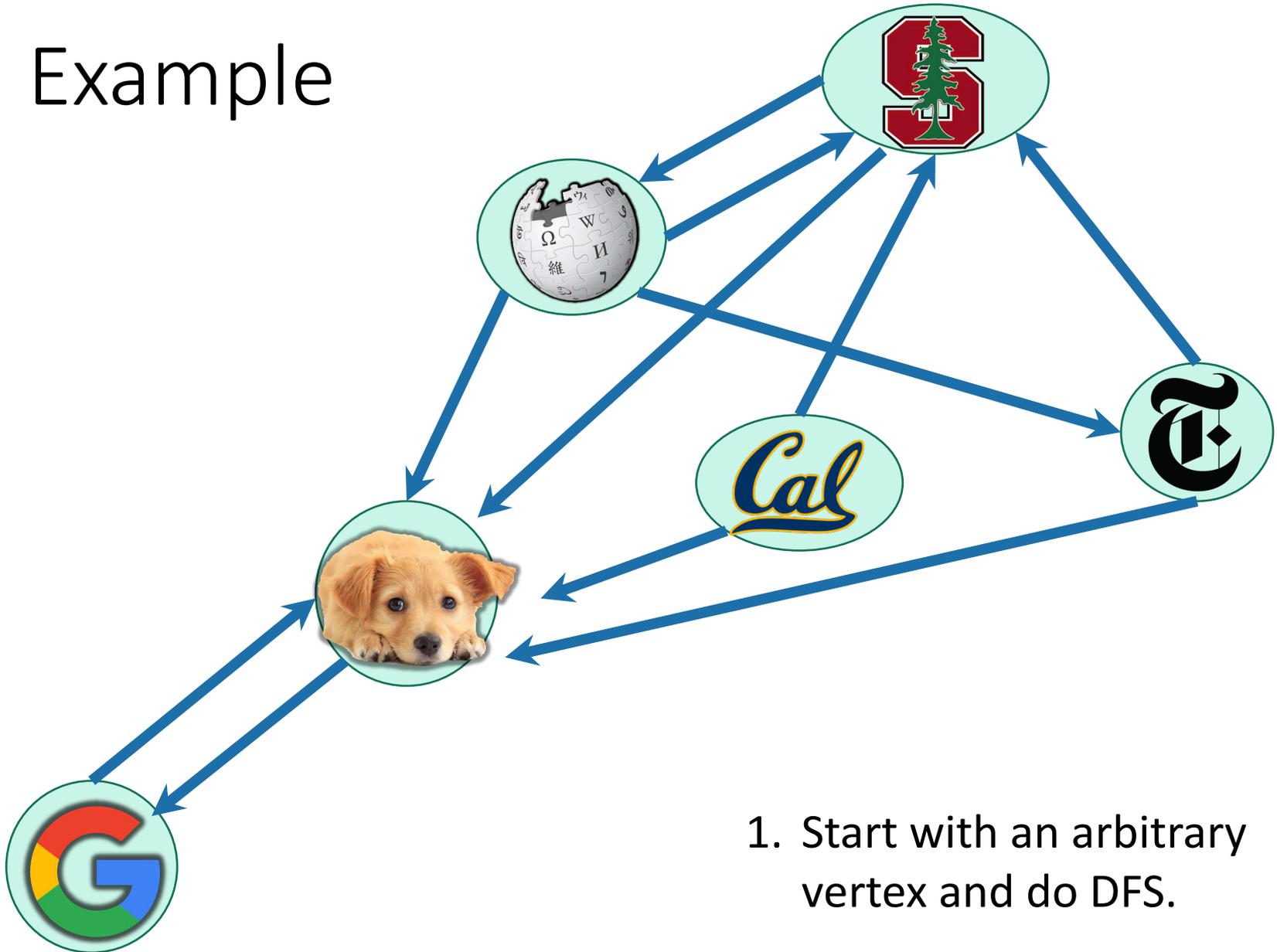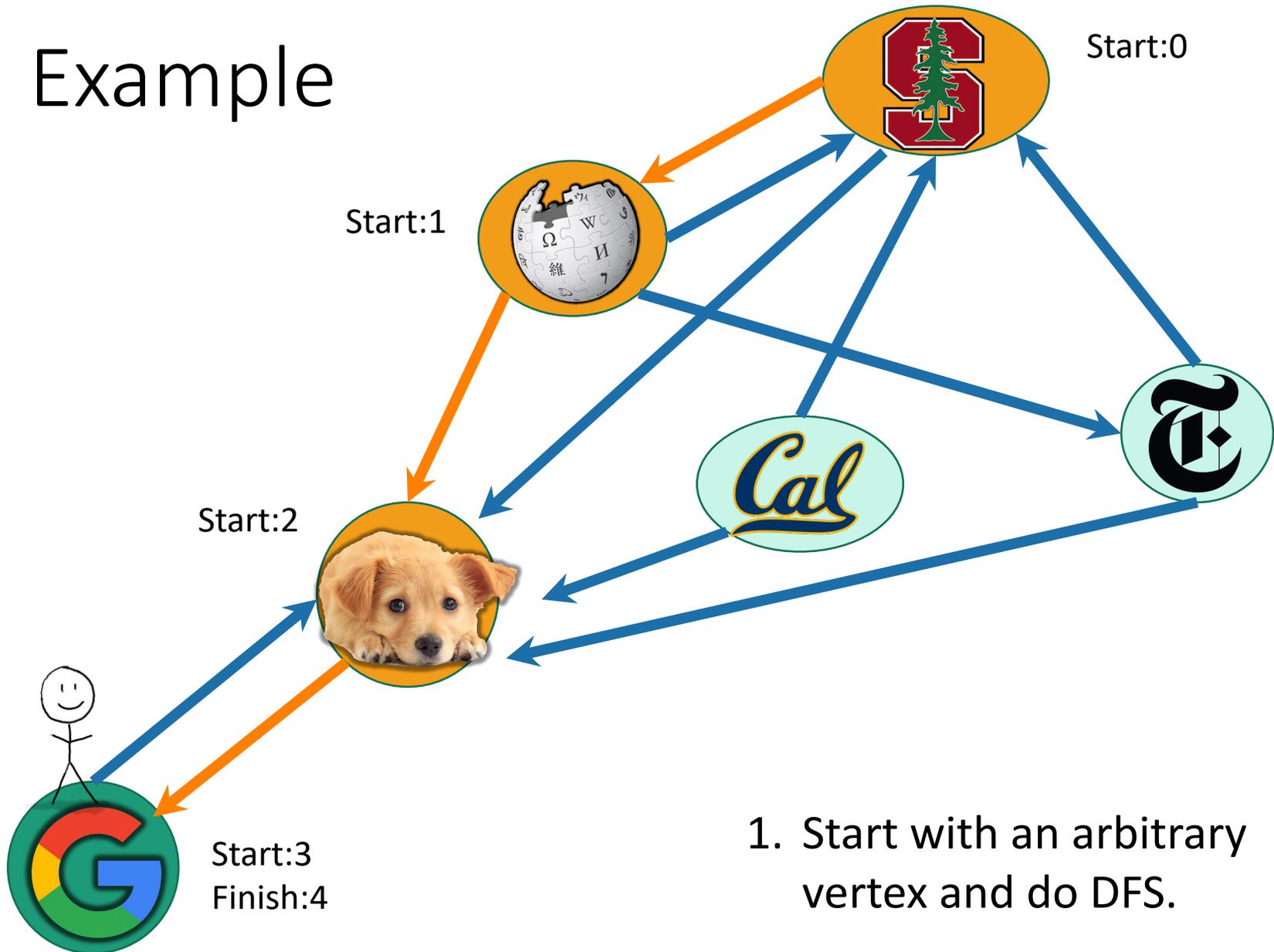
Example

1. Start with an arbitrary vertex and do DFS.

Example

Start:0

1. Start with an arbitrary vertex and do DFS.

Example

Start:0

Start:1

1. Start with an arbitrary vertex and do DFS.

Example

Start:0

Start:1

Start:2

1. Start with an arbitrary vertex and do DFS.

Example

Start:0

Start:1

Start:2

Start:3

1. Start with an arbitrary vertex and do DFS.

Example

Start:0

Start:1

Start:2

Start:3
Finish:4

1. Start with an arbitrary vertex and do DFS.

57

Example

Start:0

Start:1

Start:2
Finish:5

Start:3
Finish:4

1. Start with an arbitrary vertex and do DFS.

58

Example

Start:0

Start:1

Start:2
Finish:5

Start:3
Finish:4

1. Start with an arbitrary vertex and do DFS.

Example

Start:0

Start:1

Start:2
Finish:5

Start:6

Start:3
Finish:4

1. Start with an arbitrary vertex and do DFS.

60

Example

Start:0

Start:1

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4

1. Start with an arbitrary vertex and do DFS.

Example

Start:0

Start:1
Finish:8

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4
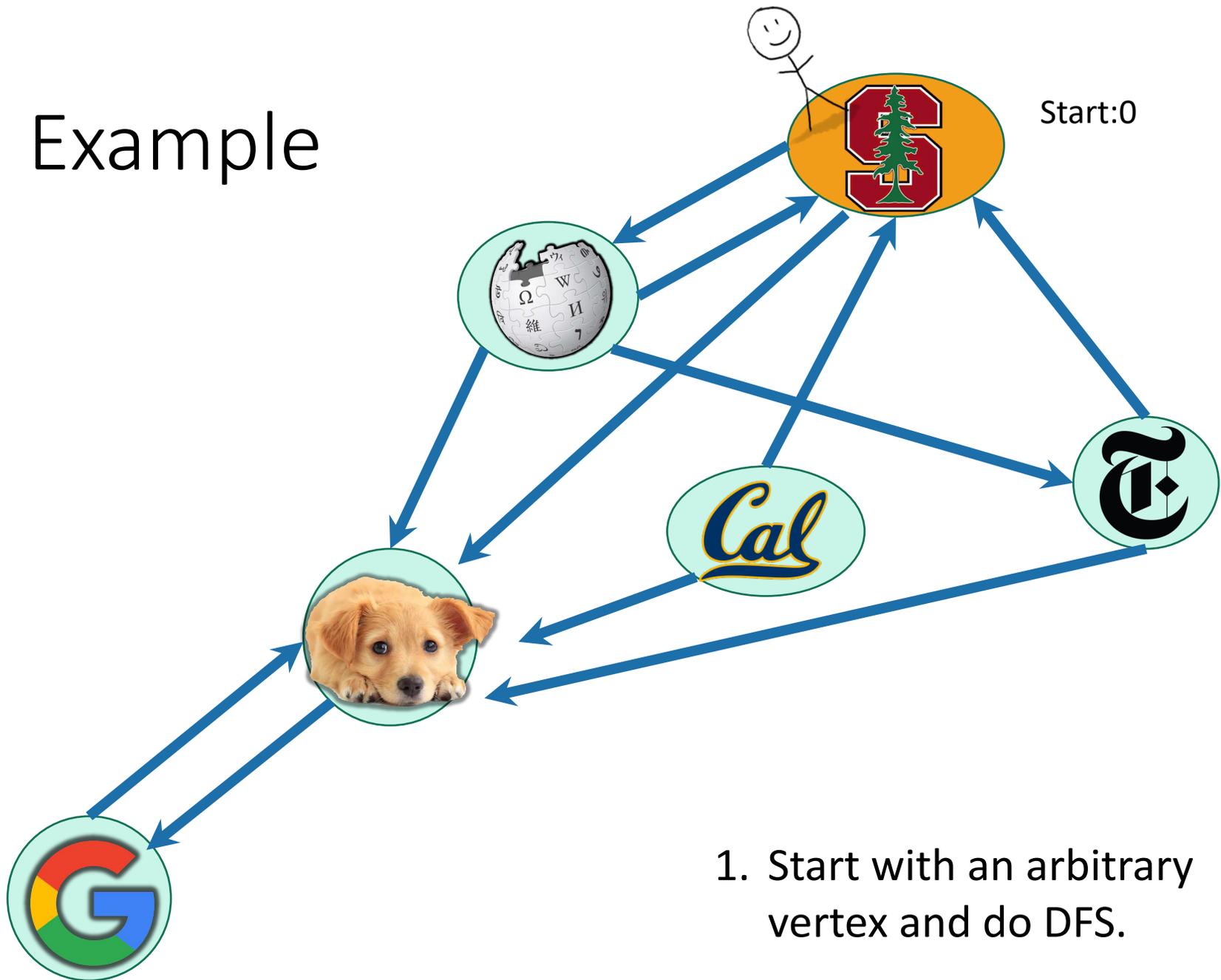
1. Start with an arbitrary vertex and do DFS.

Example

Start:0
Finish:9
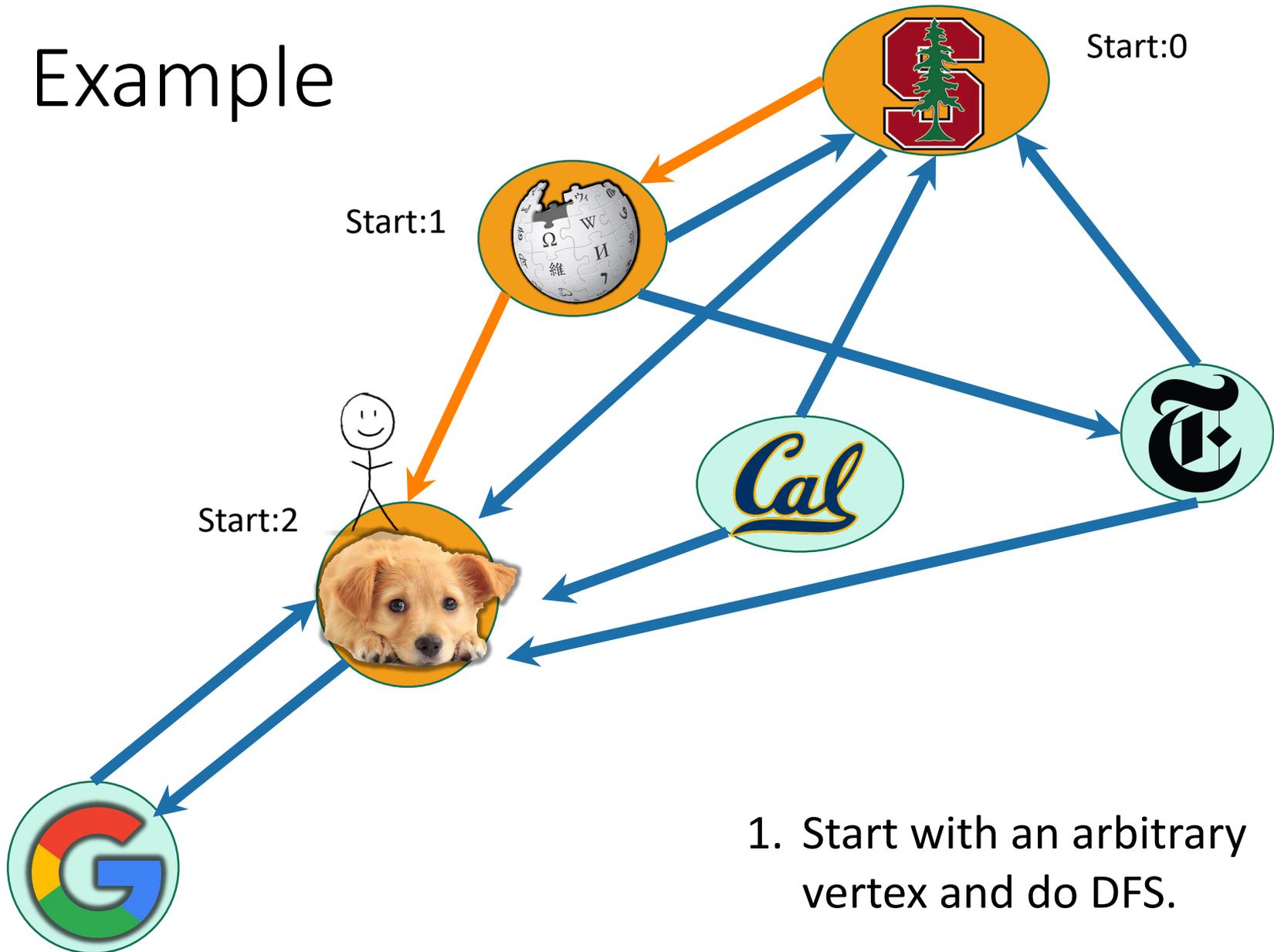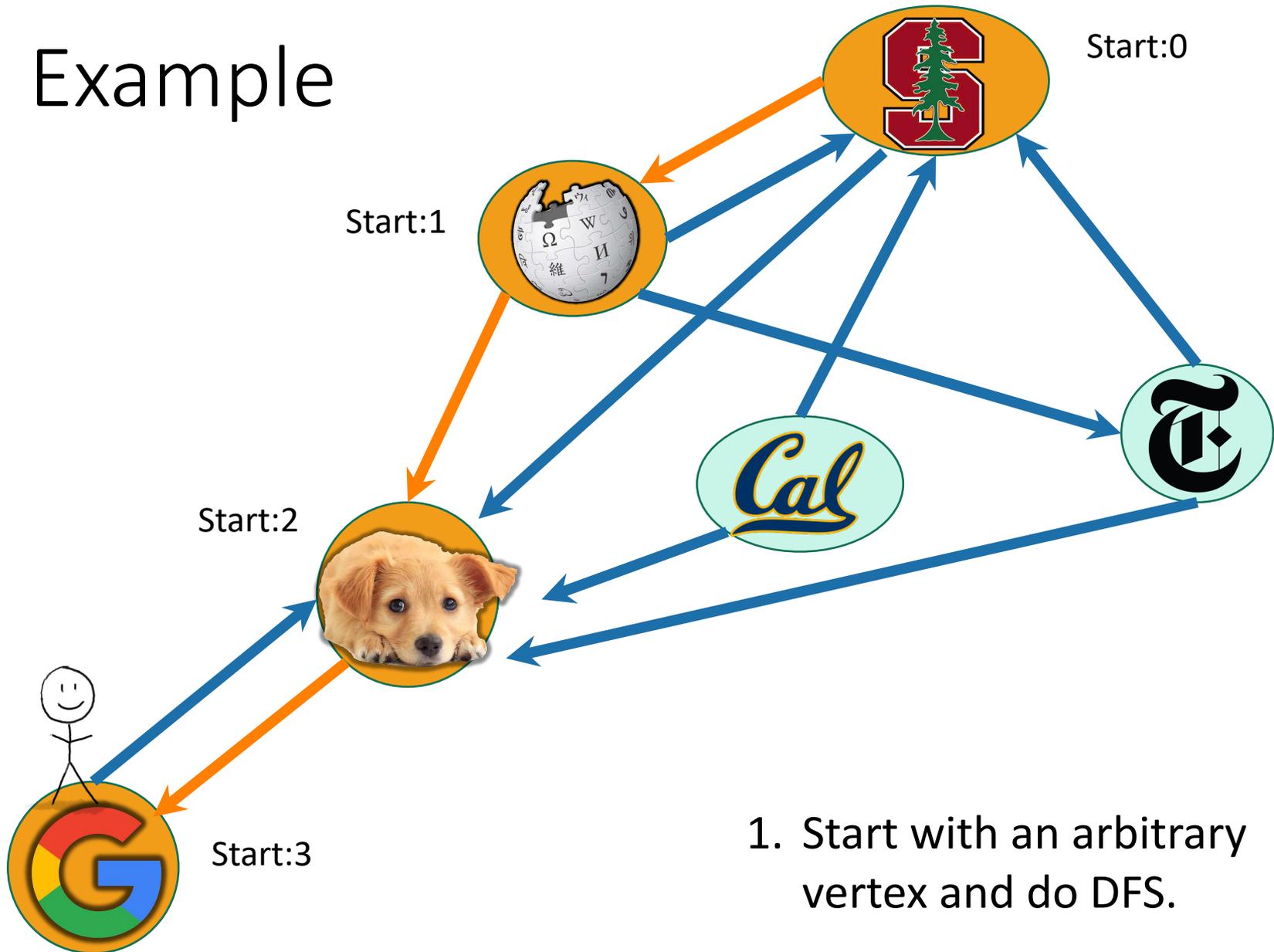
Start:1
Finish:8

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4

1. Start with an arbitrary vertex and do DFS.

63

Example

Start:0
Finish:9

Start:1
Finish:8

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4

1. Start with an arbitrary vertex and do DFS. Repeat until done.

64

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4

1. Start with an arbitrary vertex and do DFS.
   Repeat until done.

65

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

2. Reverse all the edges.

Start:3
Finish:4

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

Start:3
Finish:4

2. Reverse all the edges.

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

This is one DFS tree in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

70

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

This is one DFS tree
in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

This is one DFS tree
in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

Start:3
Finish:4

This is one DFS tree
in the DFS forest!

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

73

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

This is one DFS tree
in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

74

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

This is one DFS tree
in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

Example

Start:0
Finish:9

Start:1
Finish:8

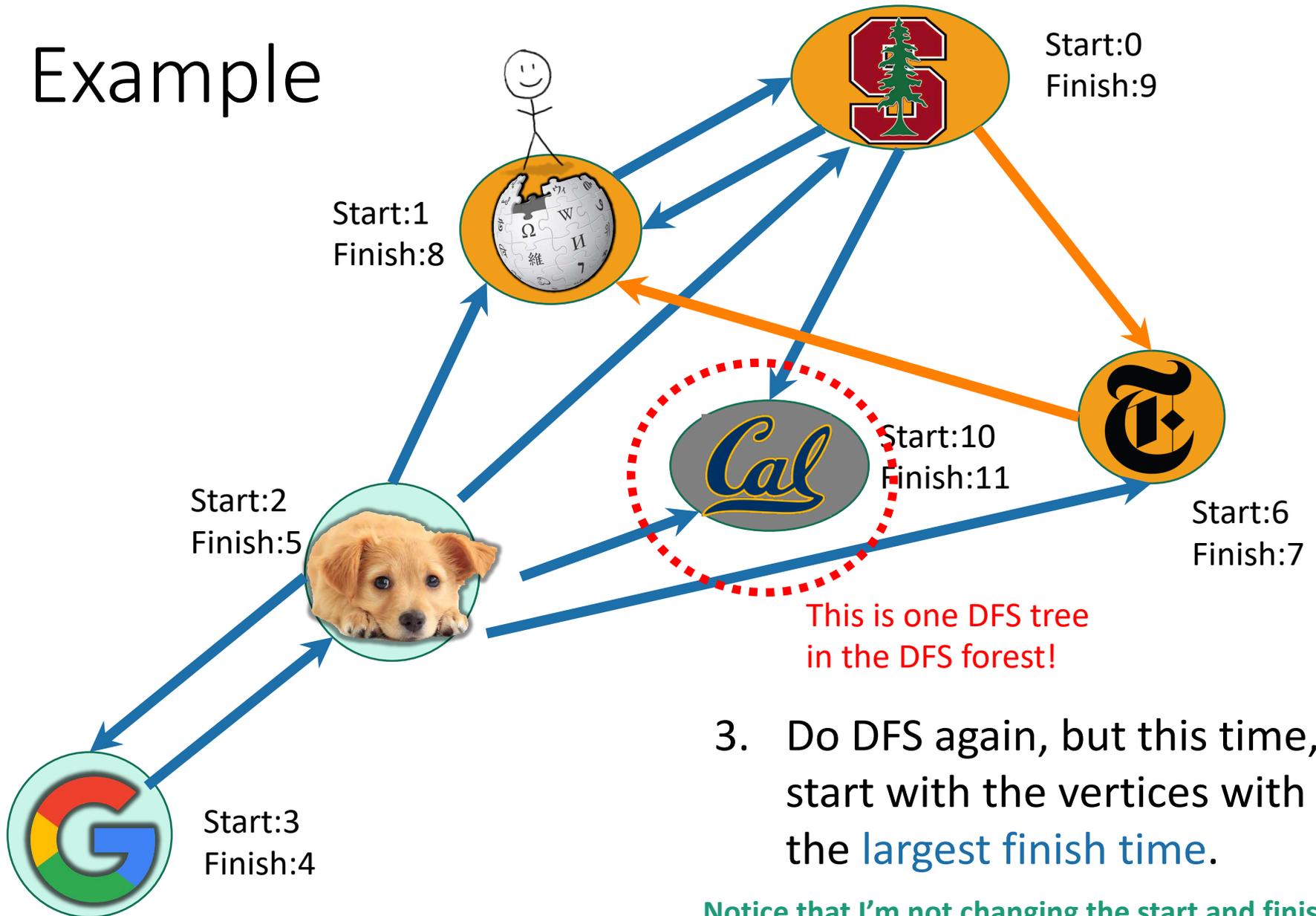Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

This is one DFS tree
in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

76

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

This is one DFS tree
in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

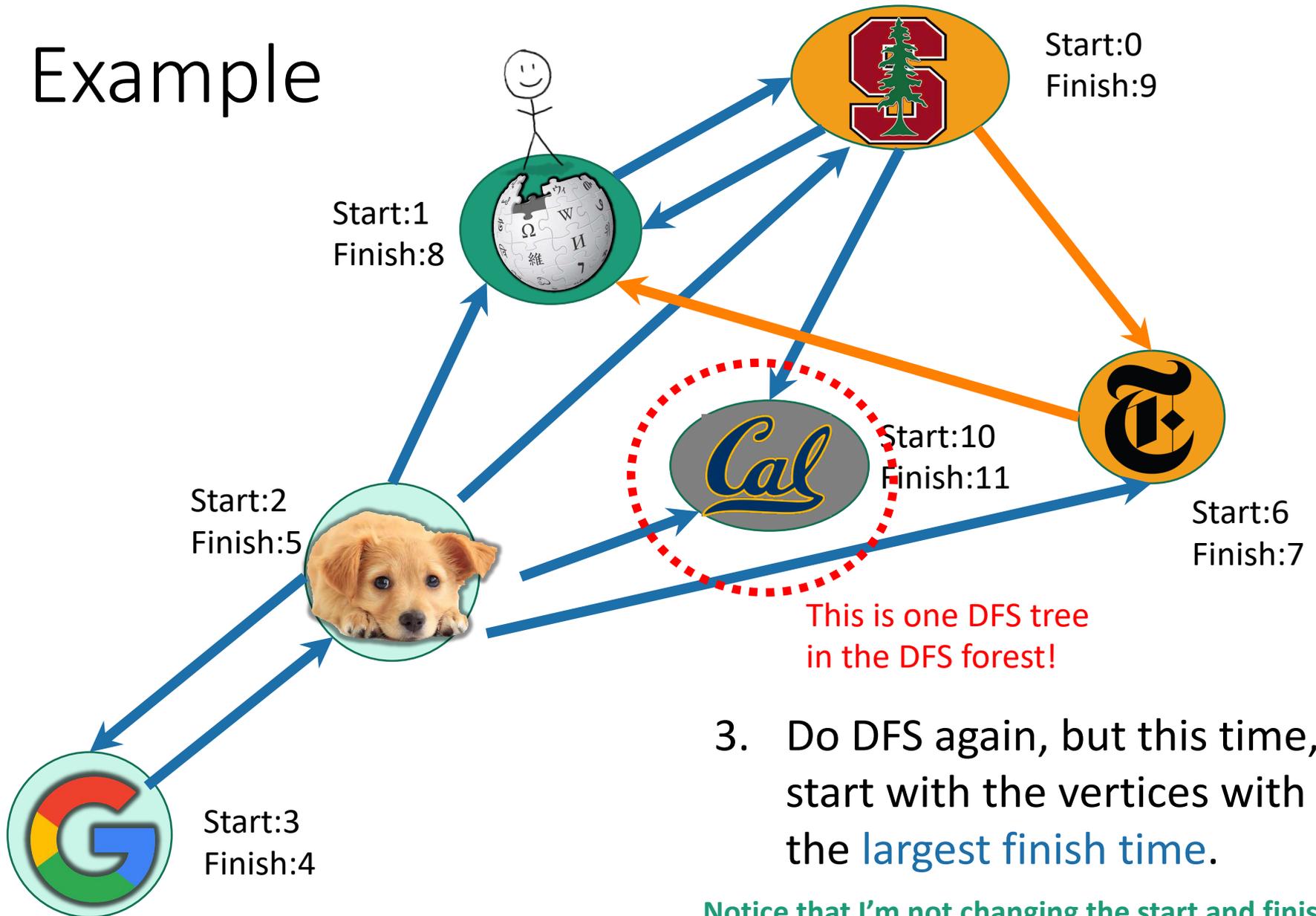Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

77

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.
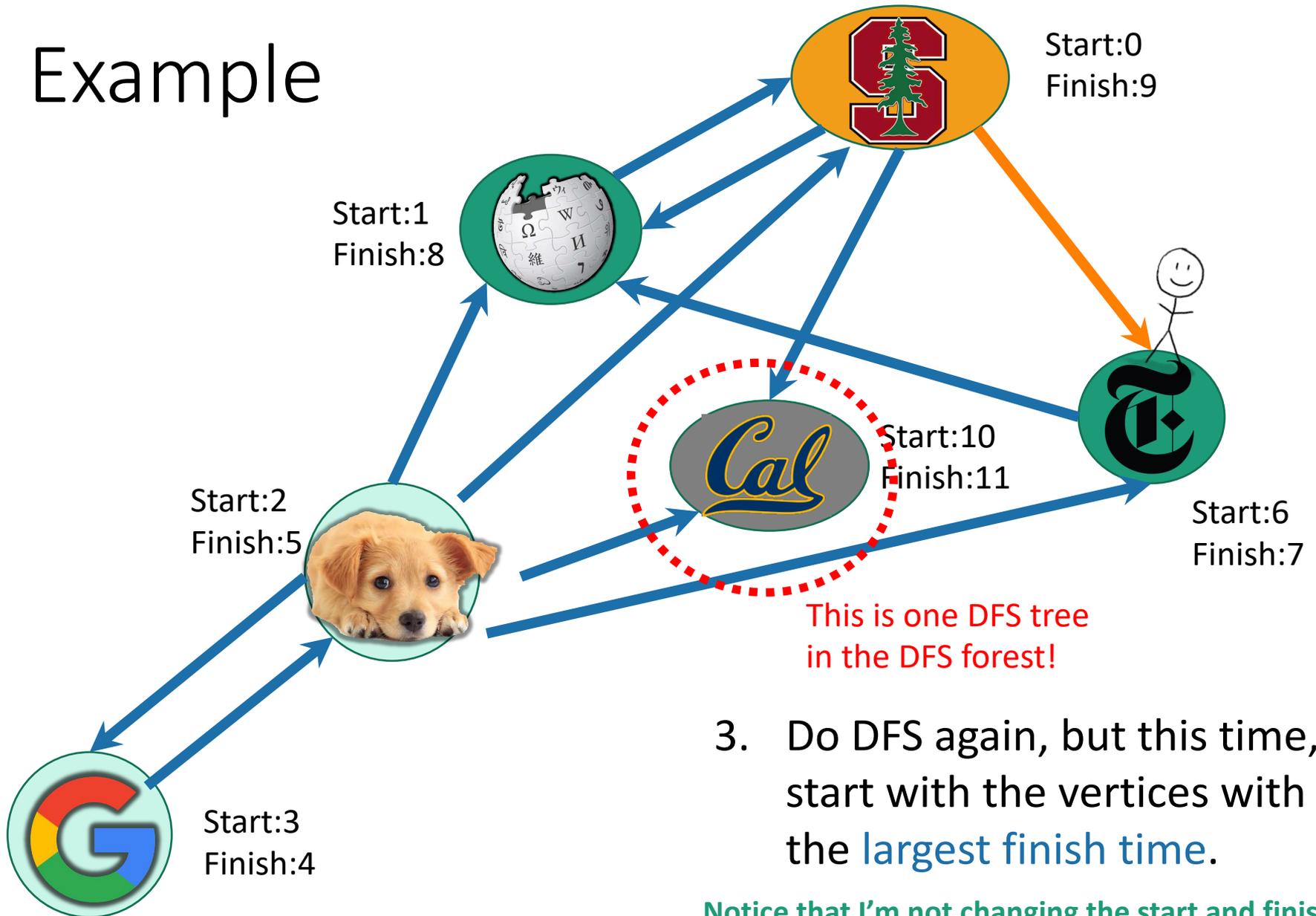
78

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

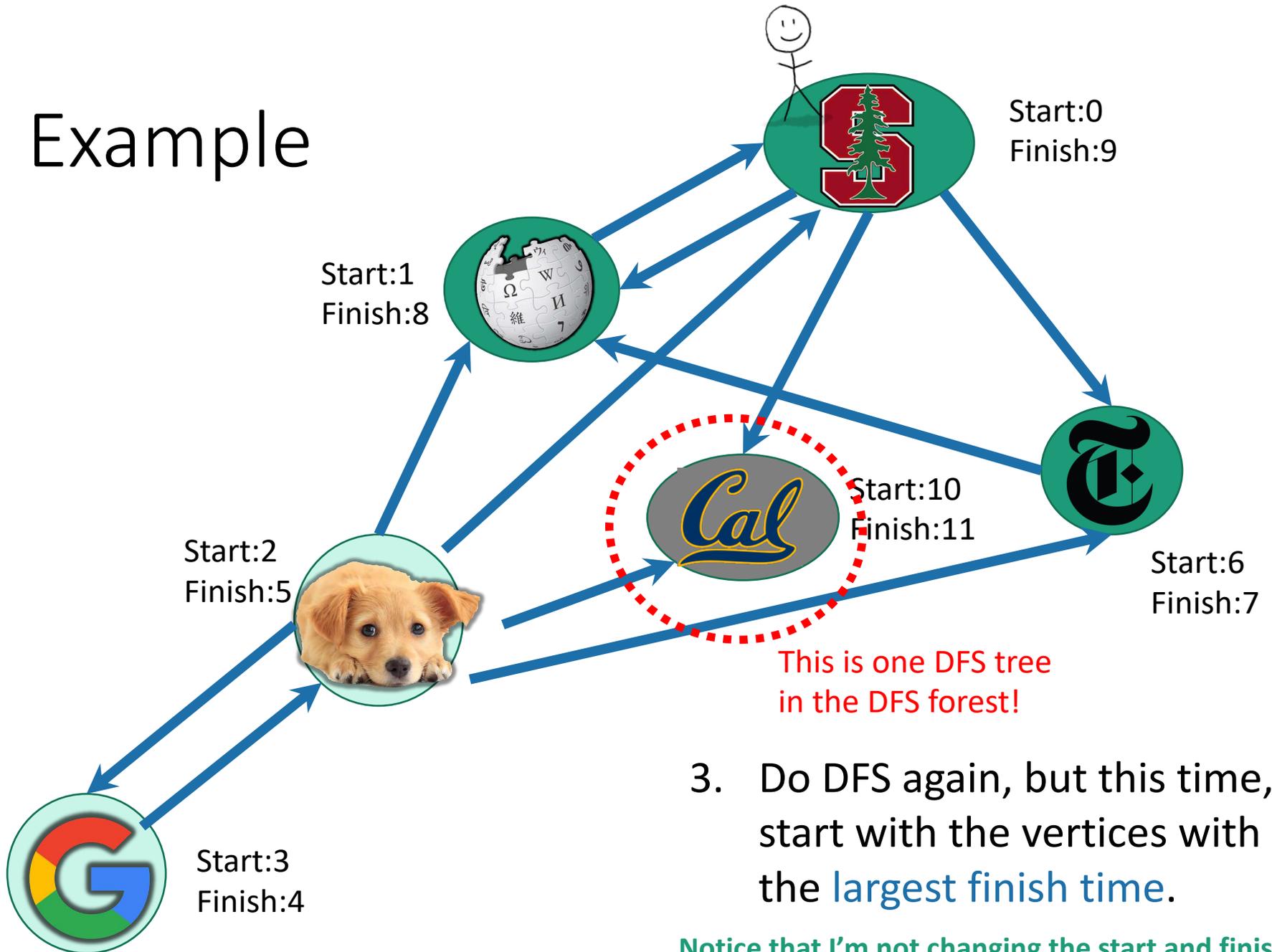Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

79

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

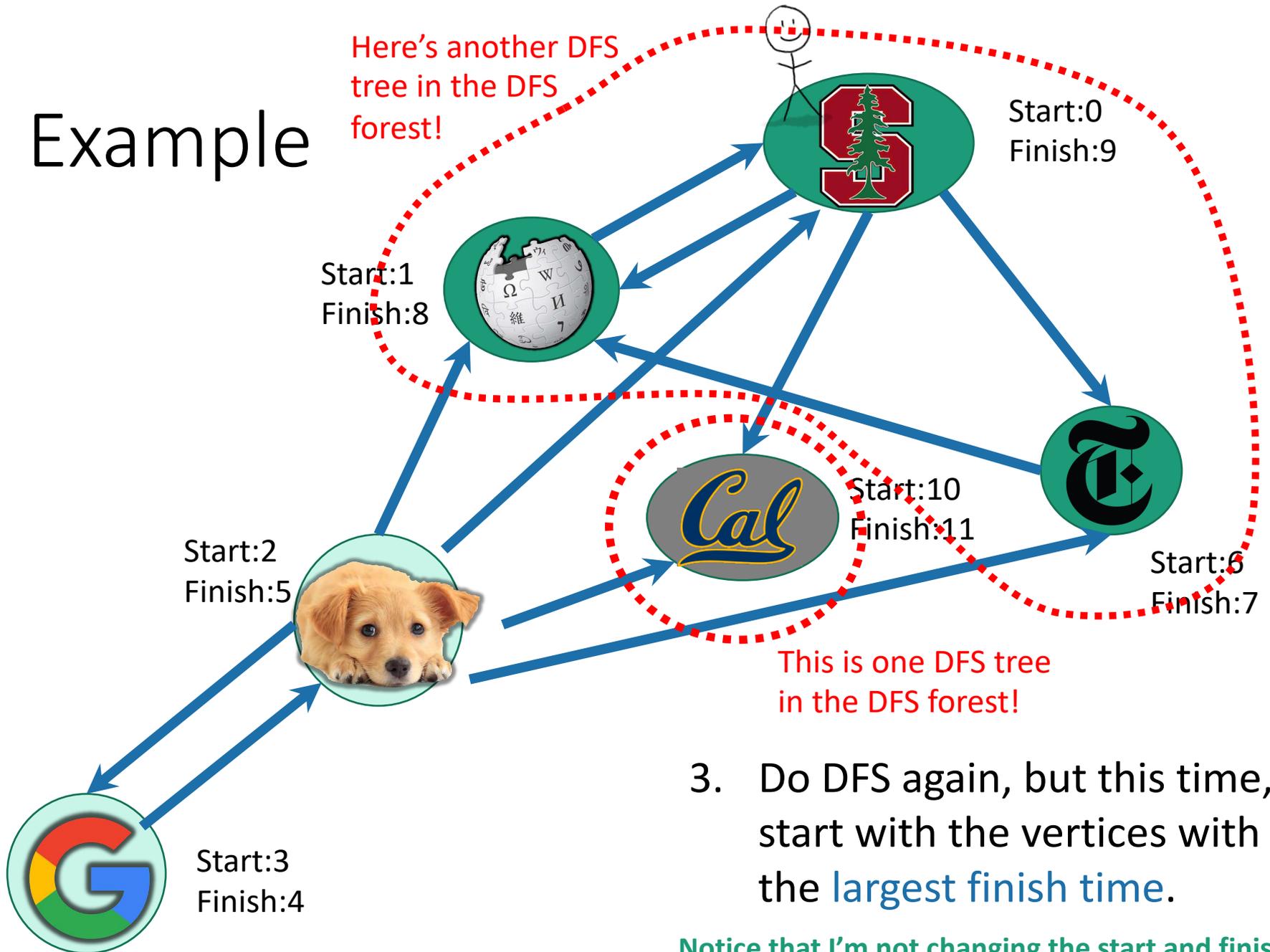Notice that I'm not changing the start and finish times – I'm keeping them from the first run.
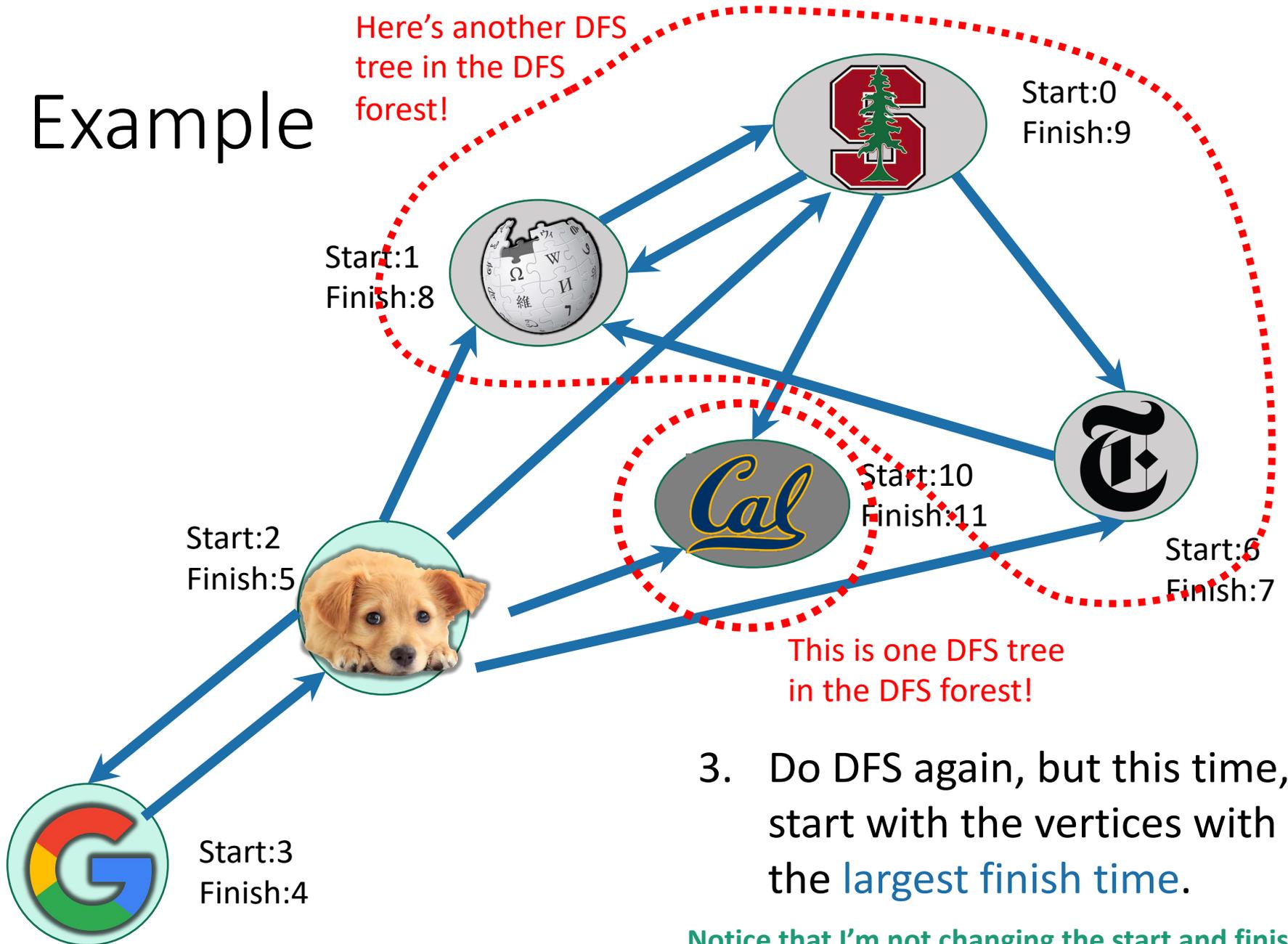
80

Example

Start:0
Finish:9

Start:1
Finish:8

Start:2
Finish:5

Start:10
Finish:11

Start:6
Finish:7

This is one DFS tree in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

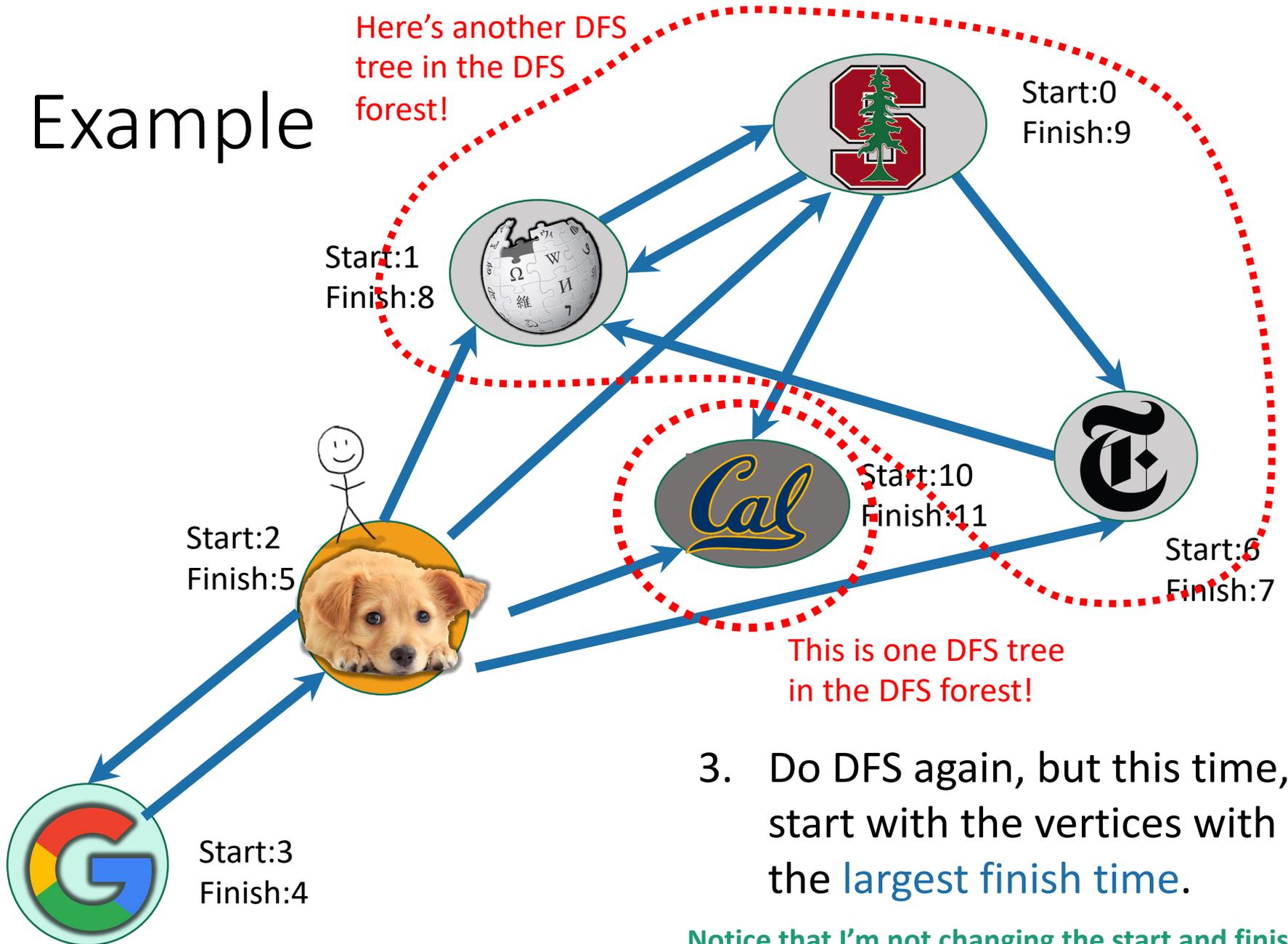Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

81

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
Finish:7

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

82

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:2
Finish:5

Start:6
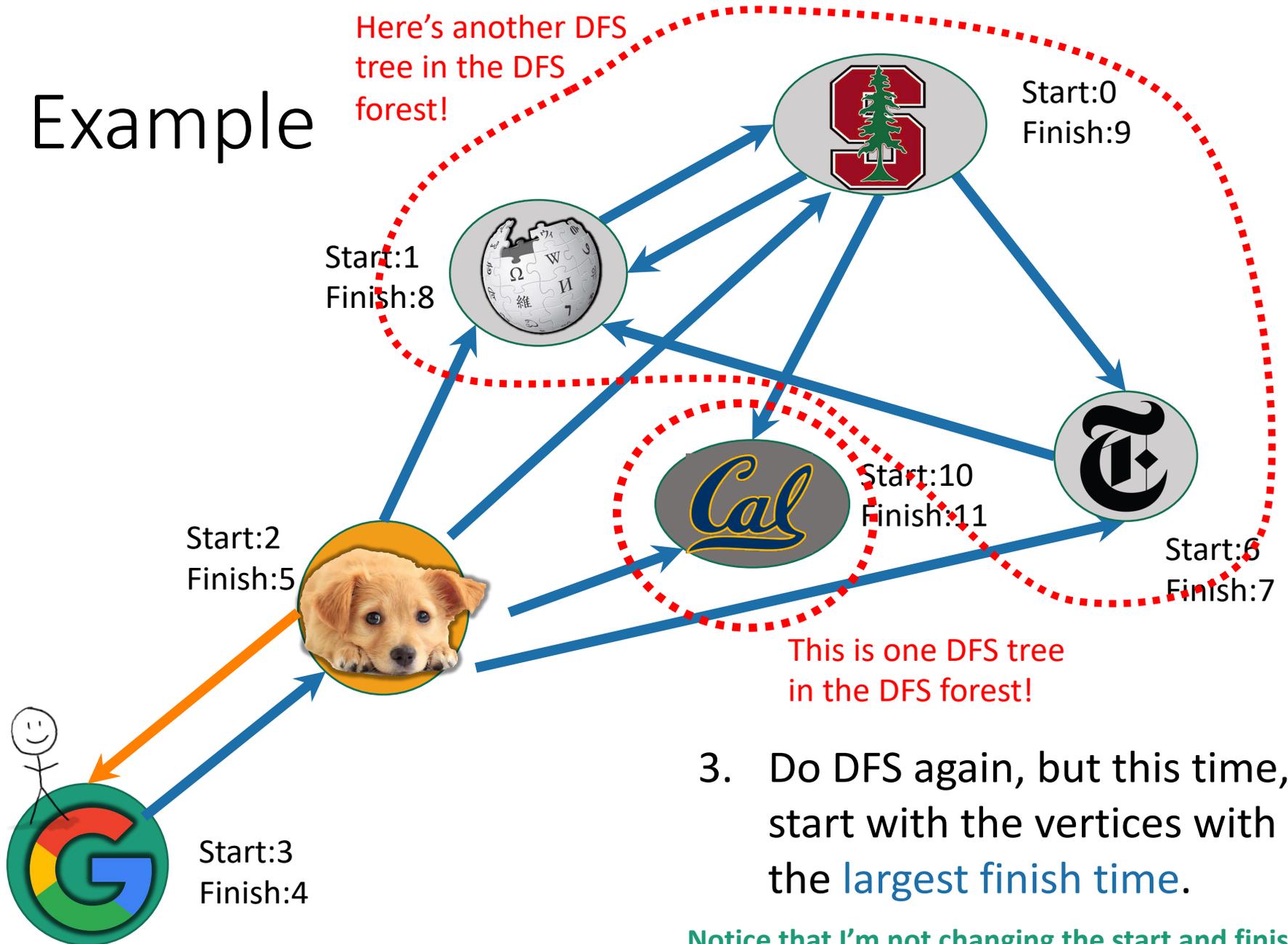Finish:7

This is one DFS tree in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

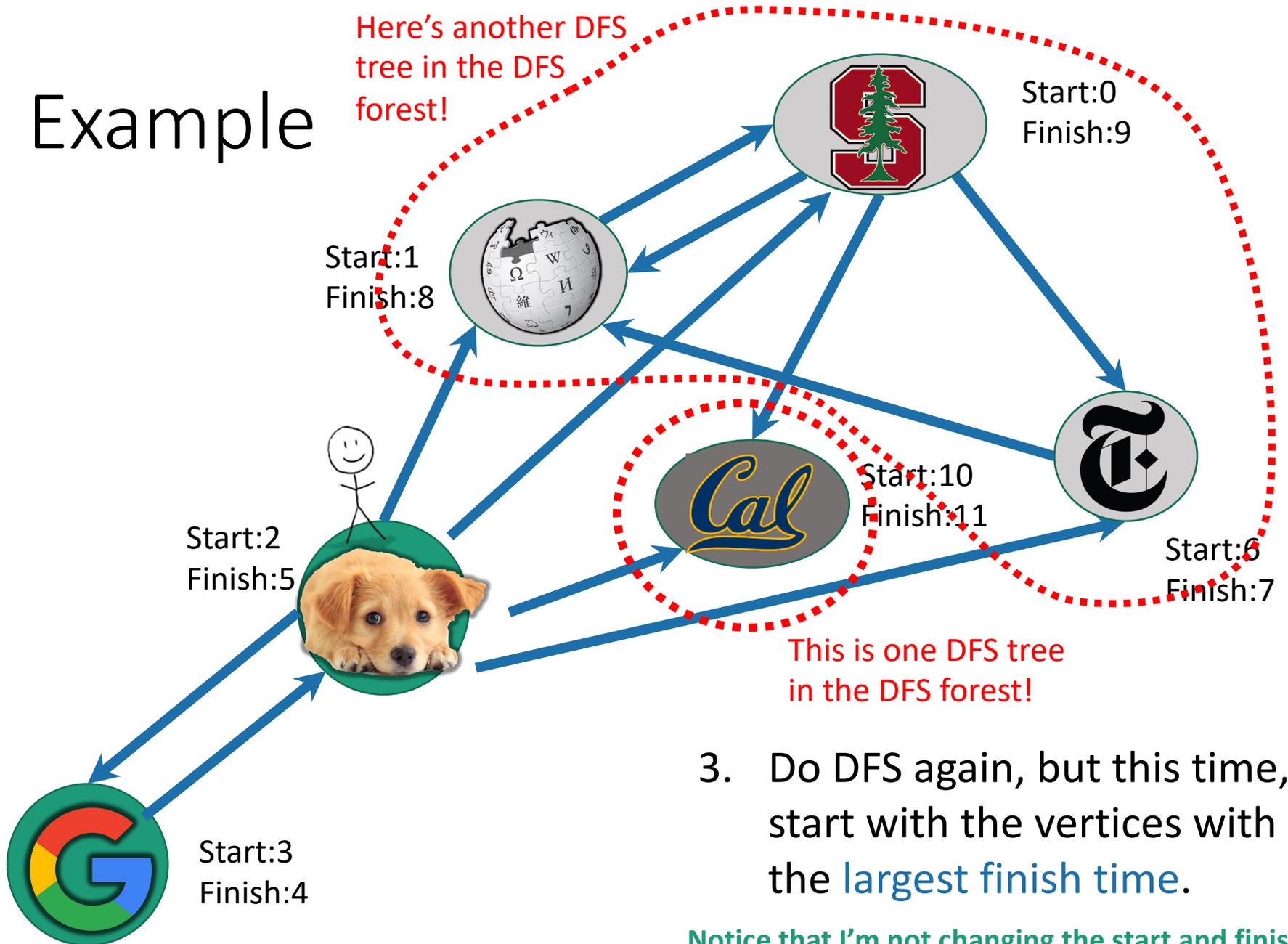**Notice that I'm not changing the start and finish times – I'm keeping them from the first run.**

83

Example

Start:0
Finish:9

Start:1
Finish:8

Start:10
Finish:11

Start:6
Finish:7

Start:2
Finish:5

The last DFS tree!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

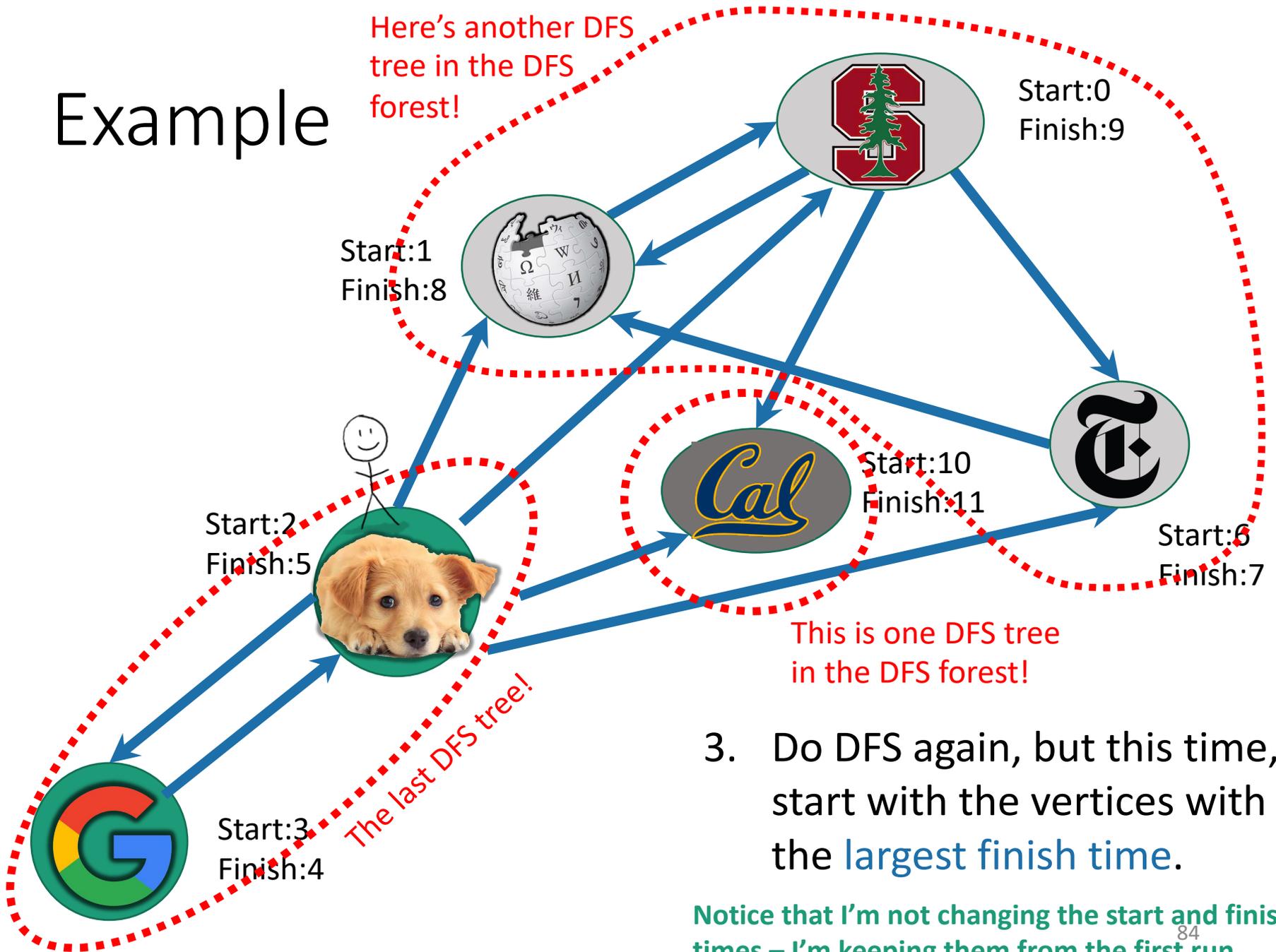Notice that I'm not changing the start and finish times – I'm keeping them from the first run.
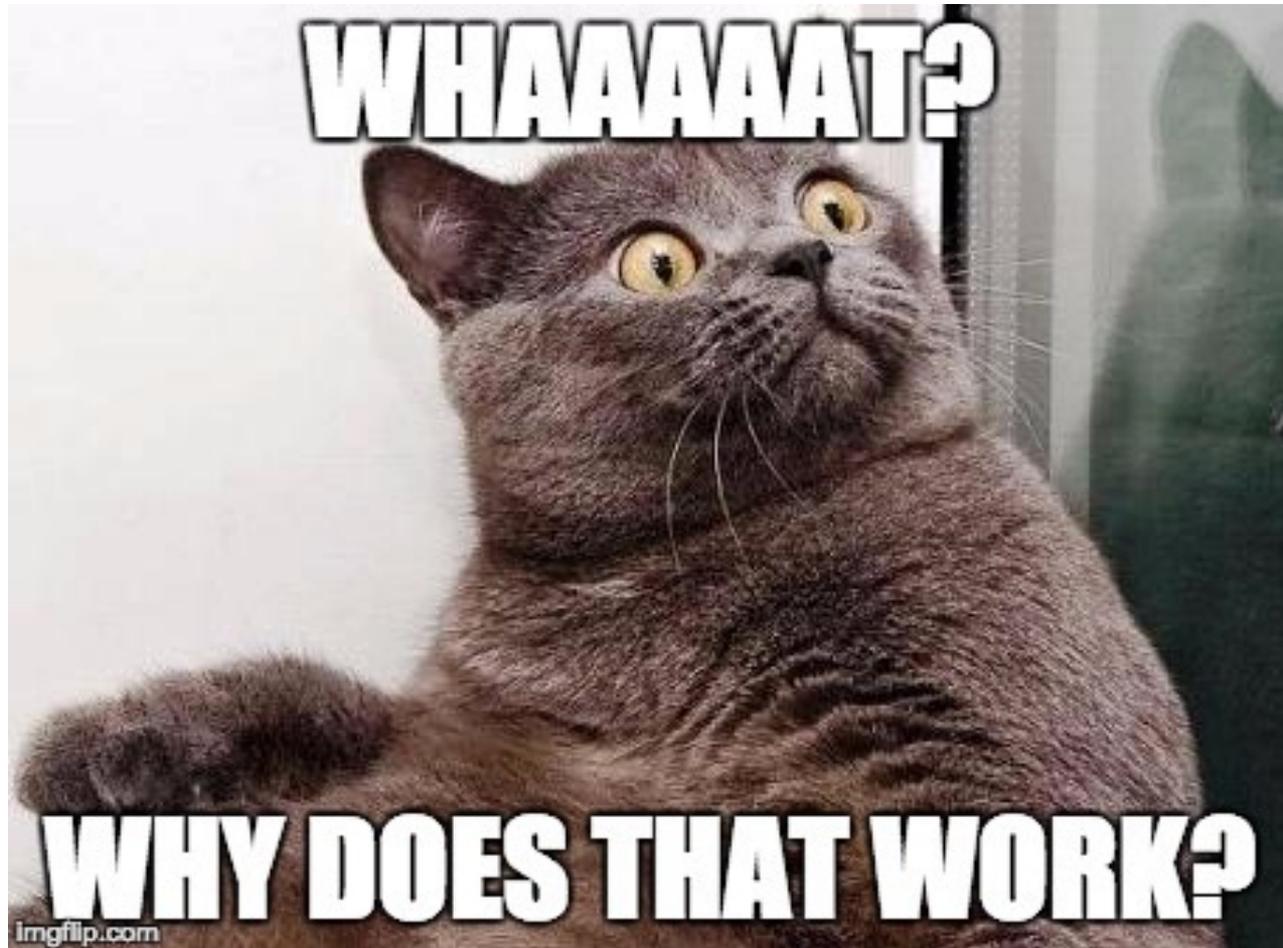
84

Example

Start:0
Finish:9

Start:1
Finish:8

Start:2
Finish:5

Start:10
Finish:11

Start:6
Finish:7

The last DFS tree!

This is one DFS tree in the DFS forest!

Start:3
Finish:4

3. Do DFS again, but this time, start with the vertices with the largest finish time.

IT WORKED!

85

One question

WHAAAAAT?

WHY DOES THAT WORK?

imgflip.com

The SCC graph

THOUGHT EXPERIMENT
memegenerator.net

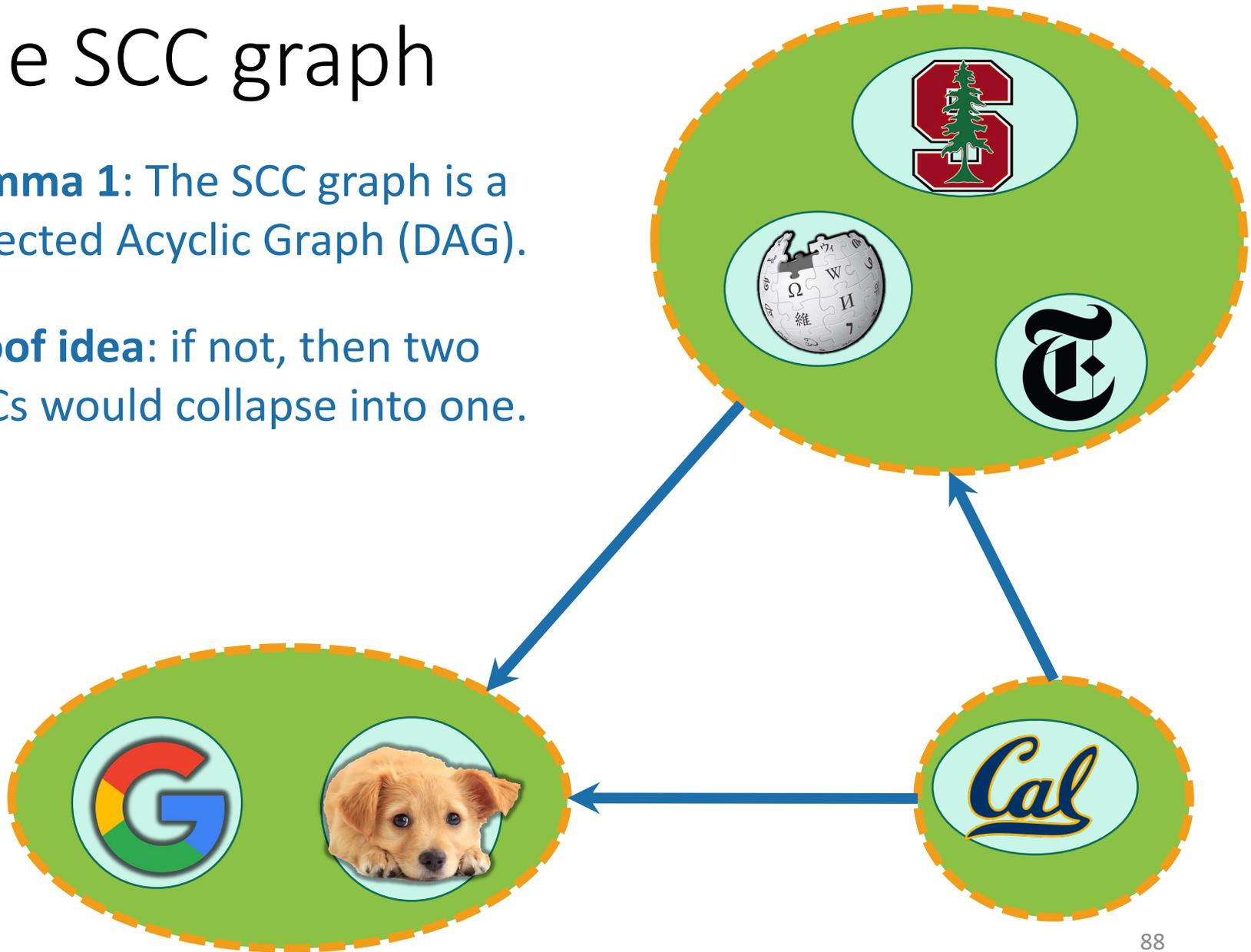- Pretend that each SCC is a vertex in a new graph.

The SCC graph

Lemma 1: The SCC graph is a Directed Acyclic Graph (DAG).

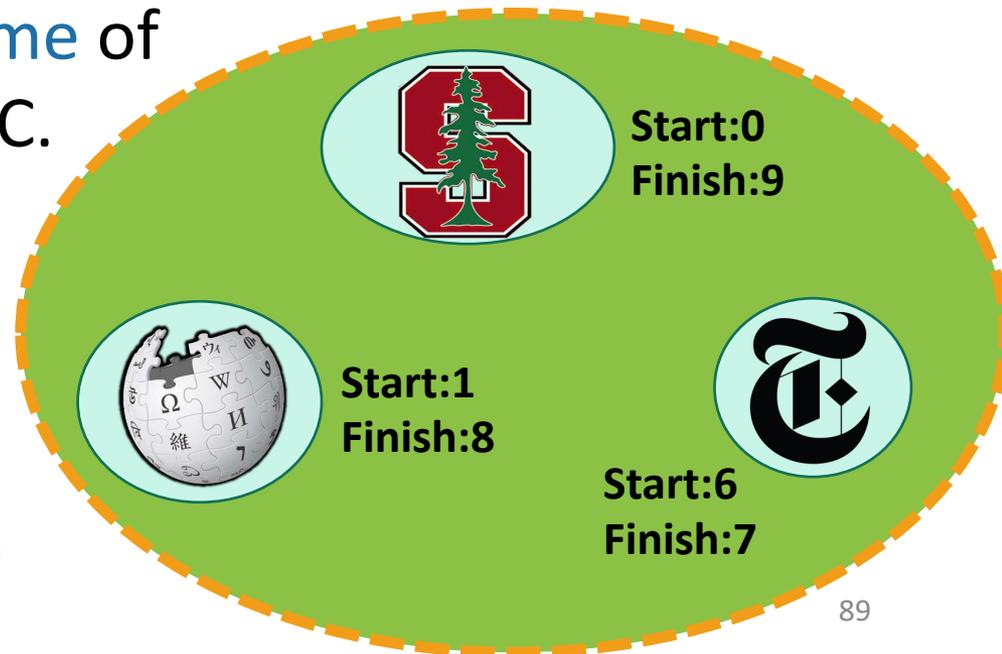**Proof idea**: if not, then two SCCs would collapse into one.
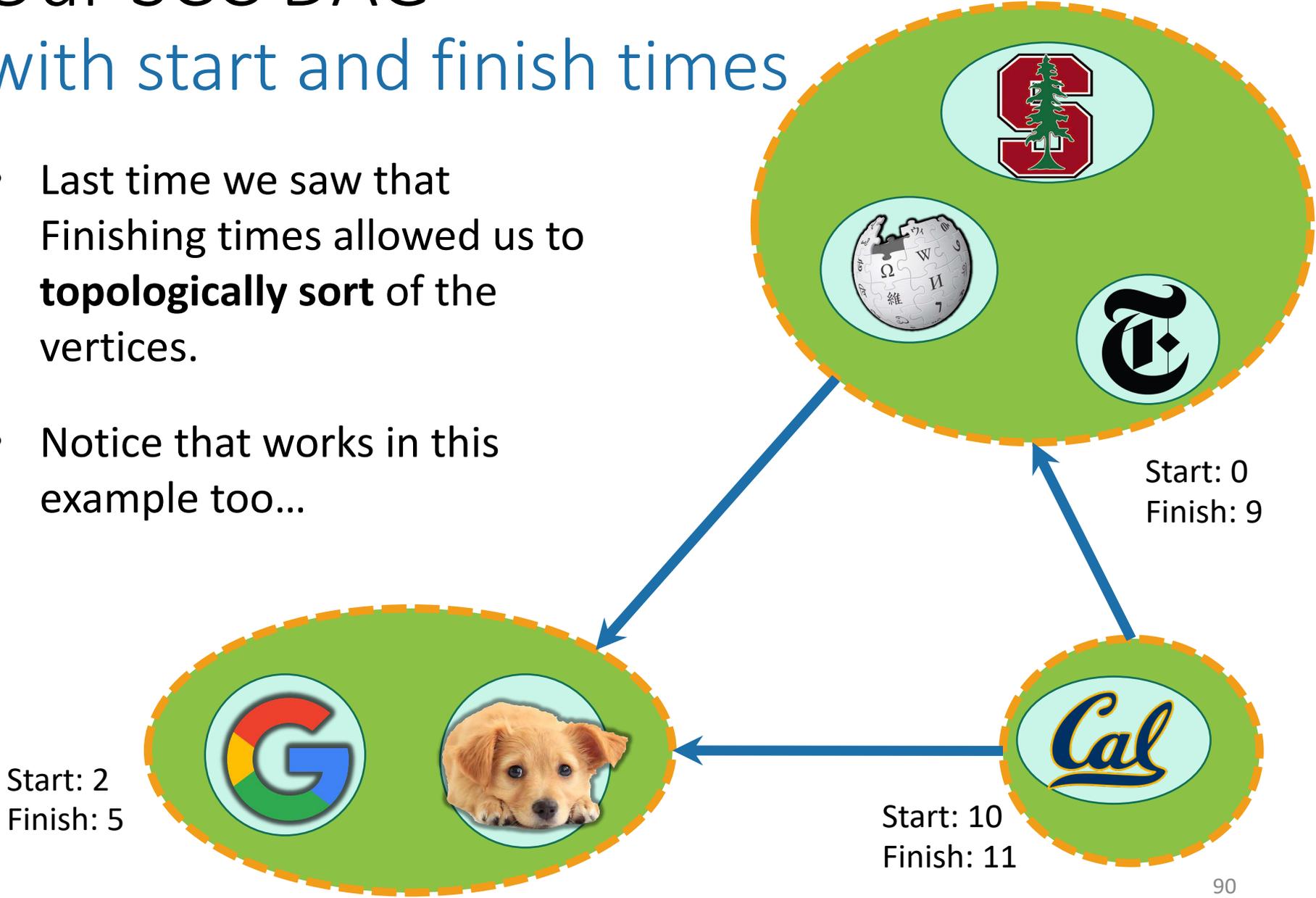
Starting and finishing times in a SCC

Definitions:

- The **finishing time** of a SCC is the largest finishing time of any element of that SCC.

- The **starting time** of a SCC is the smallest starting time of any element of that SCC.

Start:0
Finish:9

Start:1
Finish:8

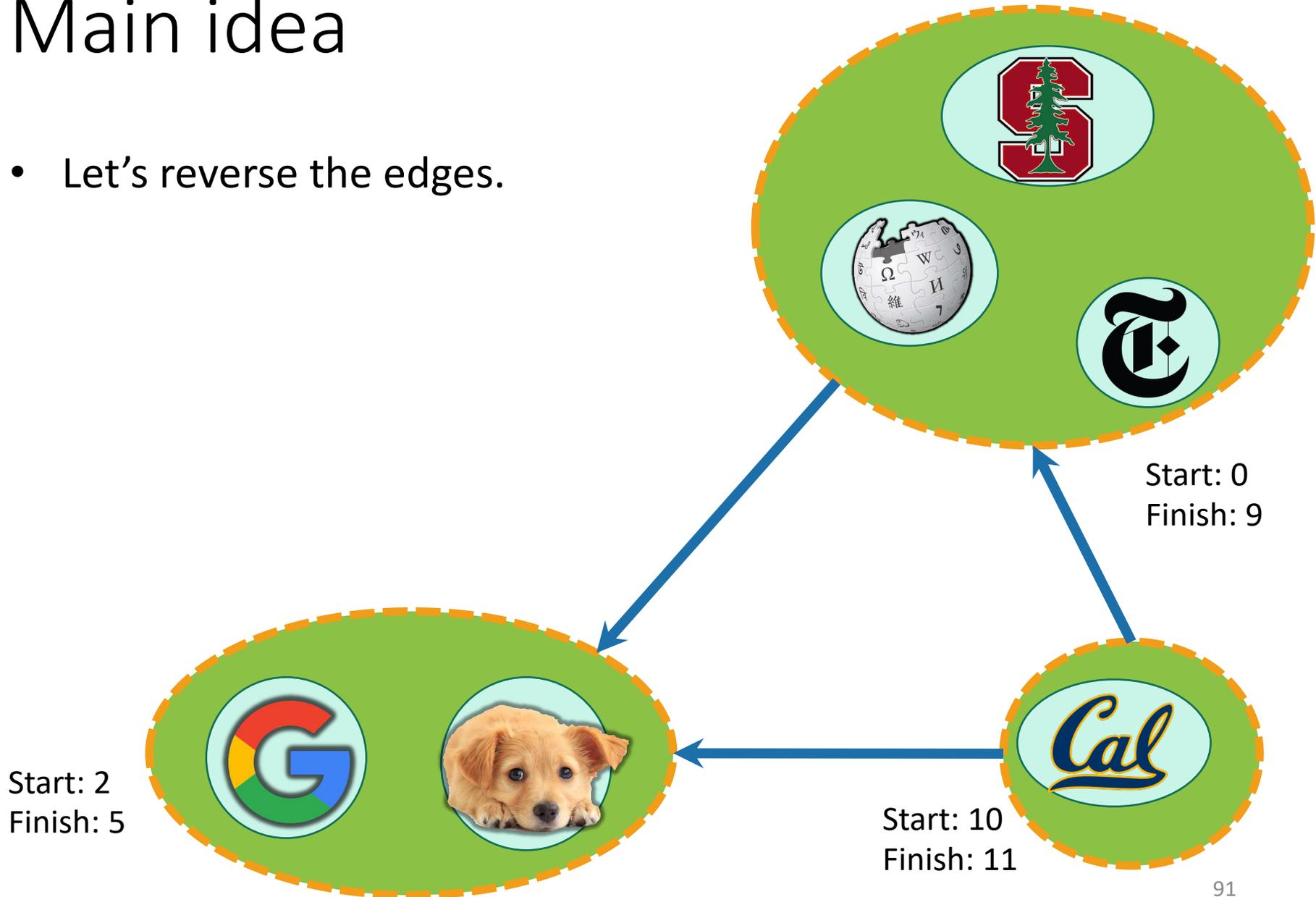Start:6
Finish:7

Start: 0
Finish: 9

89

Our SCC DAG
with start and finish times

- Last time we saw that Finishing times allowed us to **topologically sort** of the vertices.

- Notice that works in this example too…

Start: 0
Finish: 9

Start: 2
Finish: 5

Start: 10
Finish: 11

Main idea

- Let's reverse the edges.

Start: 0
Finish: 9

Start: 2
Finish: 5

Start: 10
Finish: 11

Main idea

- Let's reverse the edges.
- Now, the SCC with the largest finish time has no edges going out.
  - If it did have edges going out, then it wouldn't be a good thing to choose first in a topological ordering!
- If I run DFS there, I'll find exactly that component.
- Remove and repeat.

Start: 0
Finish: 9

Start: 2
Finish: 5

Start: 10
Finish: 11
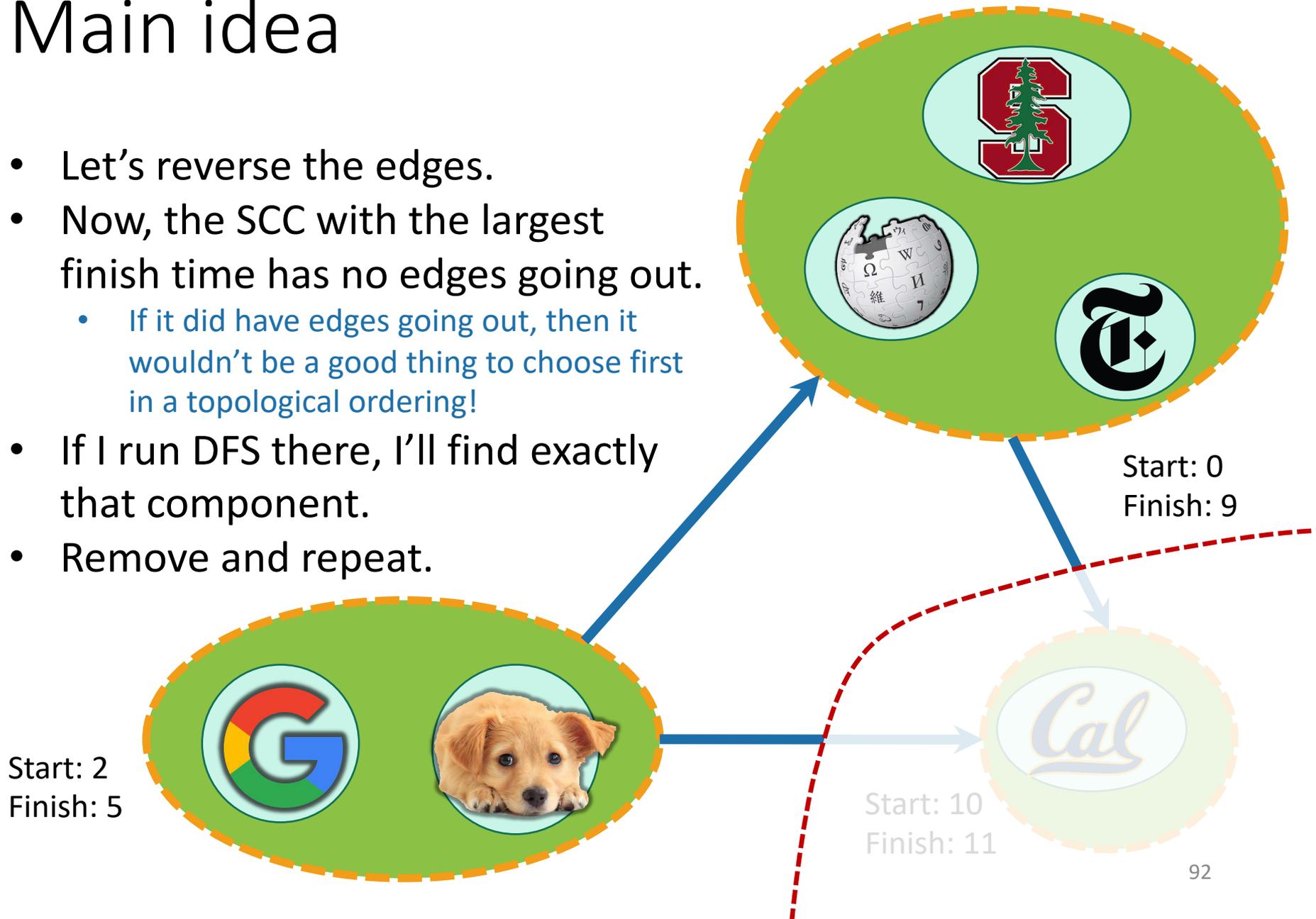
Let's make this idea formal.

Recall

- If v is a descendent of w in this tree:

w.start   v.start   v.finish   w.finish

timeline

- If w is a descendent of v in this tree:

v.start   w.start   w.finish   v.finish

- If neither are descendents of each other:

v.start   v.finish   w.start   w.finish

(or the other way around)

w

v
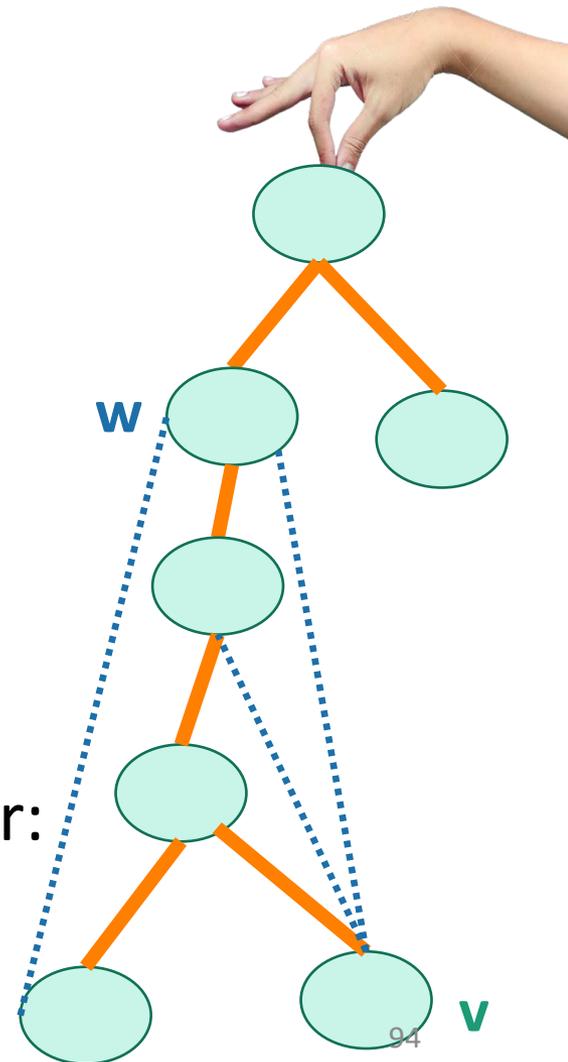
As we saw last time...

Claim: In a DAG, we'll always have:

finish: [larger] → finish: [smaller]
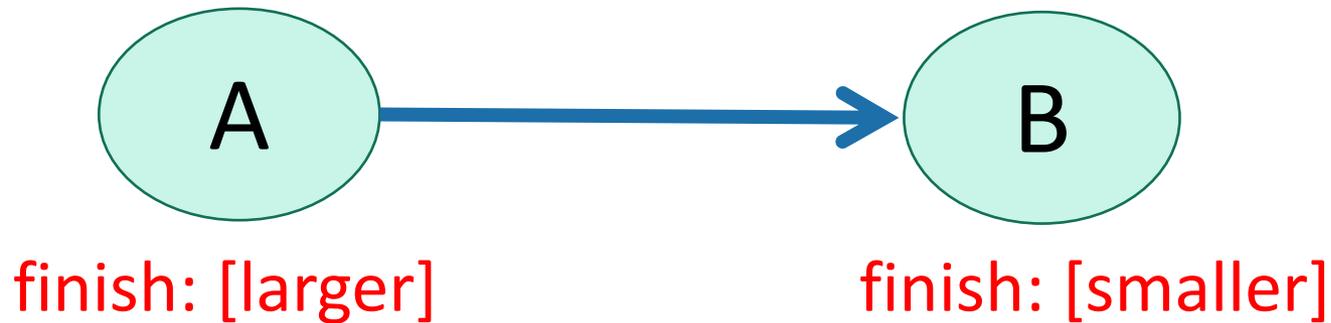
Same thing, in the SCC DAG.

- Claim: we'll always have
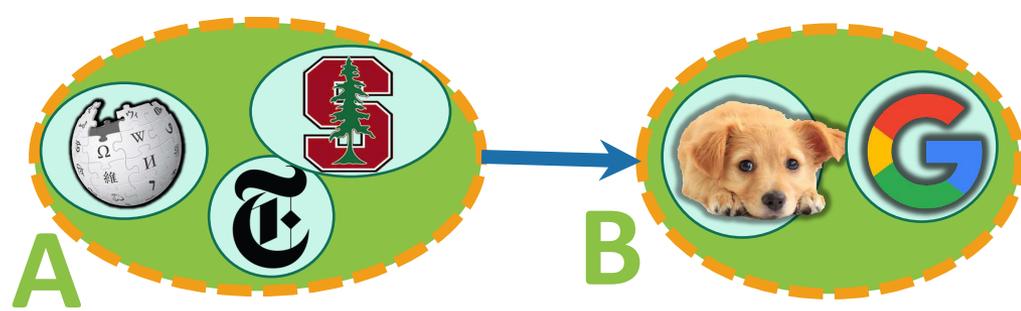

finish: [larger]   finish: [smaller]
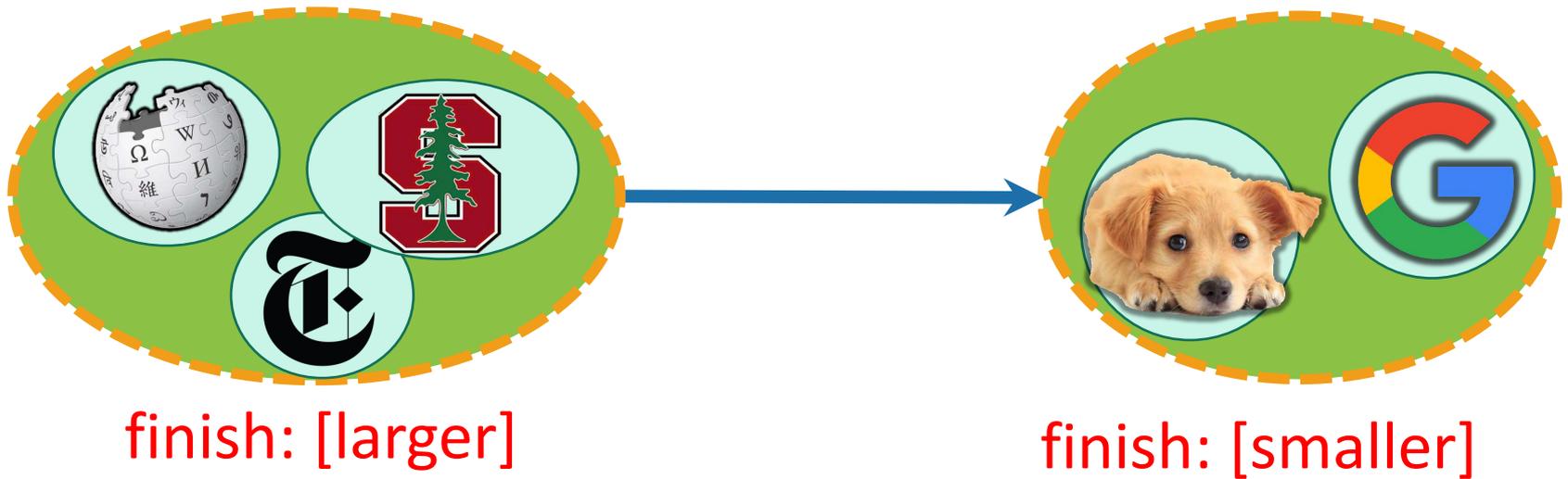
Let's call it Lemma 2

- If there is an edge like this:

A → B
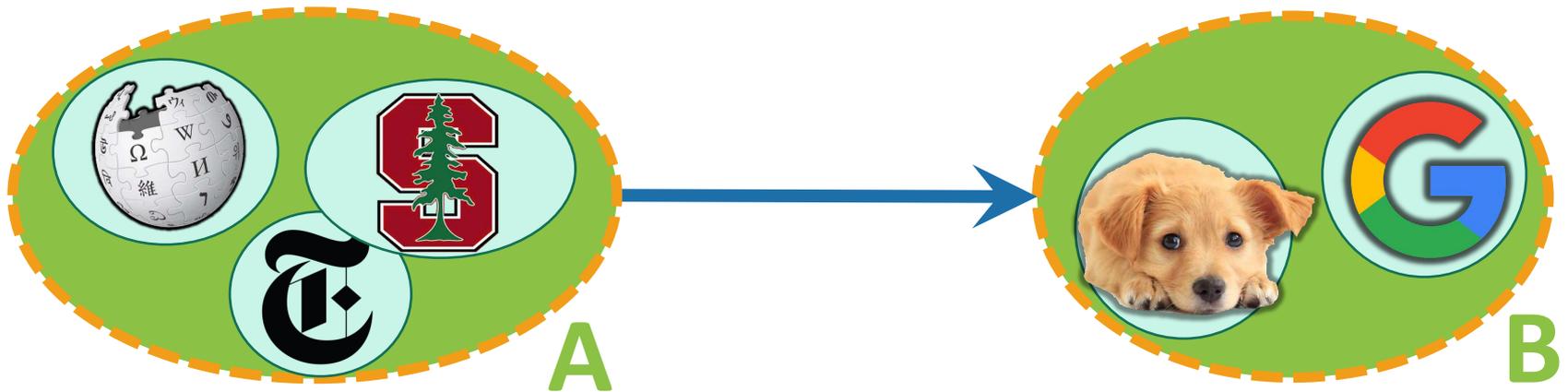
- Then A.finish > B.finish.

Proof idea

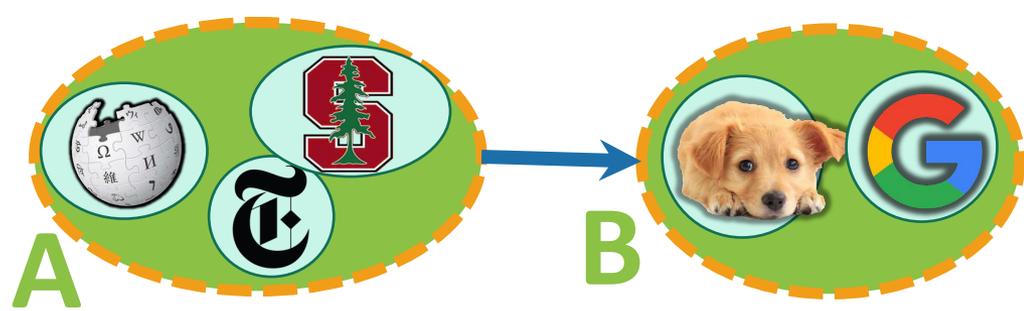

Want to show A.finish > B.finish.

- **Two cases**:
  - We reached **A** before **B** in our first DFS.
  - We reached **B** before **A** in our first DFS.

Proof idea

- **Case 1**: We reached **A** before **B** in our first DFS.

- Say that:
    - **y** has the largest finish in **B**;
    - **z** was discovered first in **A**;

    $$B.finish = y.finish$$
    $$A.finish >= z.finish$$

- Then:
    - Reach **A** before **B**
    - => we will discover **y** via **z**
    - => **y** is a descendant of **z** in the DFS forest.

- Then

y.start
z.finish
B.finish=
y.finish
z.start
$\leq$ A.finish

aka,
A.finish > B.finish

Proof idea

- **Case 2**: We reached **B** before **A** in our first DFS.

- There are no paths from B to A
  - because the SCC graph has no cycles

- So we completely finish exploring B and never reach A.
- A is explored later after we restart DFS.

**aka,
A.finish > B.finish**

Proof idea

- **Two cases**:
  - We reached **A** before **B** in our first DFS.
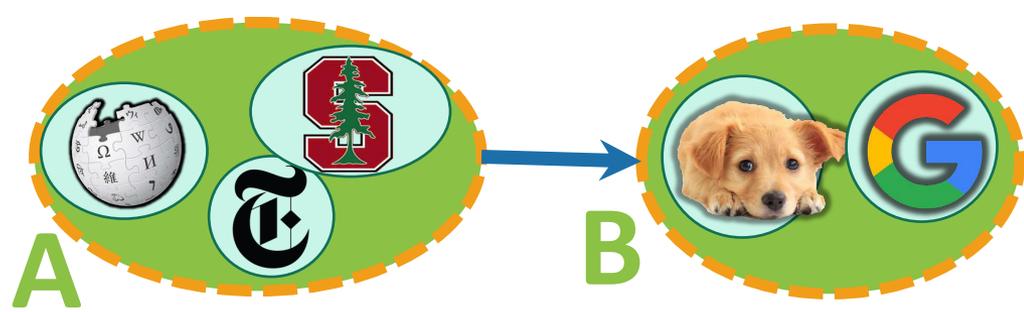 - We reached **B** before **A** in our first DFS.

- In either case:

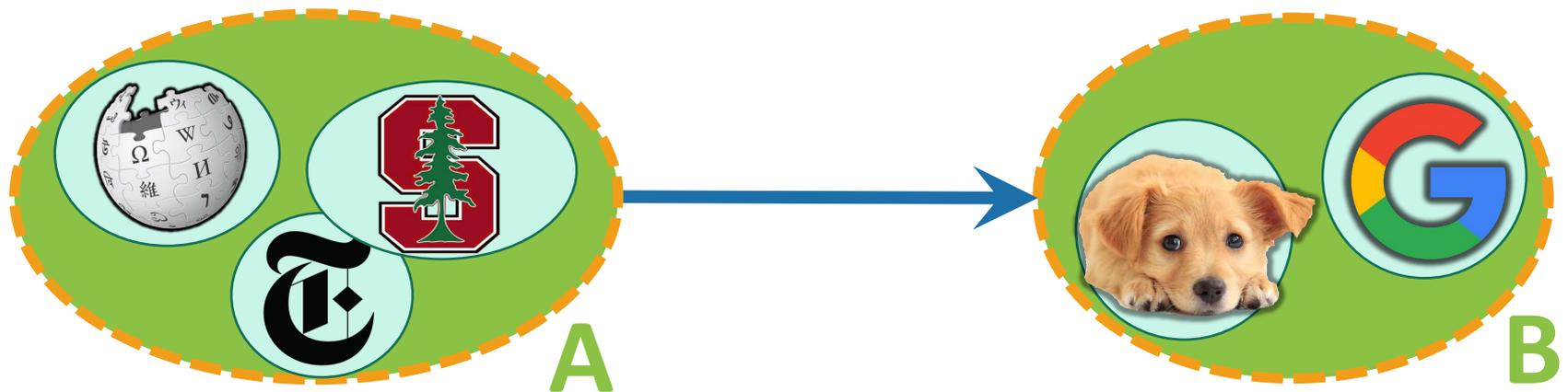$$A.finish > B.finish$$

which is what we wanted to show.

Notice: this is exactly the same two-case argument that we did last time for topological sorting, just with the SCC DAG!
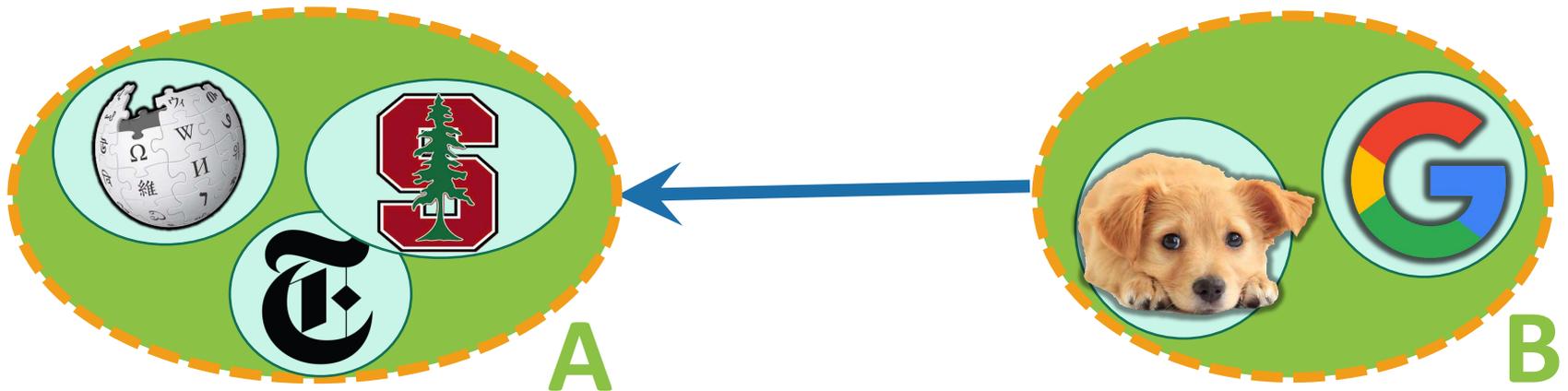
This establishes:

Lemma 2

- If there is an edge like this:

A

B

- Then A.finish > B.finish.

This establishes:
Corollary 1

- If there is an edge like this in the **reversed graph**:

- Then A.finish > B.finish.

Now we see why this finds SCCs.

- The Corollary says that **all blue arrows point towards larger finish times**.

- So if we start with the largest finish time, **all blue arrows lead in**.

- Thus, that connected component, and only that connected component, are reachable by the second round of DFS

- Now, we've deleted that first component.

- The next one has the **next biggest finishing time.**

- So **all remaining blue arrows lead in.**

- Repeat.

Start: 0
Finish: 9

Start: 2
Finish: 5

Start: 10
Finish: 11

104
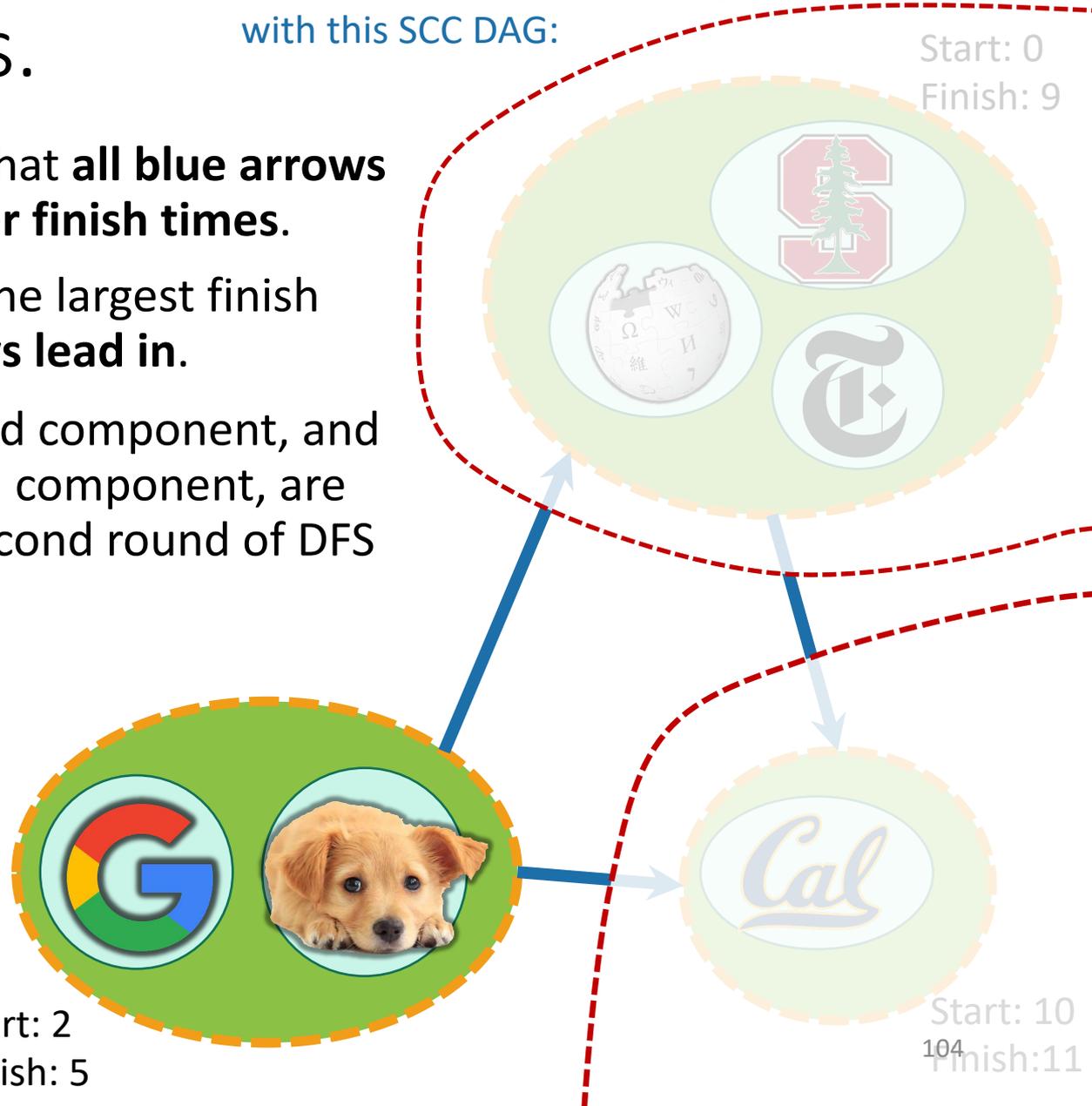
Formally, we prove it by...
Induction!

Formally, we prove it by induction

- **Theorem**: The algorithm we saw before will correctly identify strongly connected components.

- **Inductive hypothesis**:
    - The first t DFS trees found in the second (reversed) DFS forest are the t SCCs with the largest finish times.
- **Base case**: (t=0)
    - The first 0 DFS trees found in the second (reversed) DFS forest are the 0 SCCs with the largest finish times. **(TRUE)**
- **Inductive step:**
    - Fun exercise! [See skipped slide; we already did the intuition]
- **Conclusion:**
   - The IH holds when t = #SCCs, aka the second (reversed) DFS forest contains all the SCCs as its trees!

Inductive step

- **Assume by induction that the first t trees are the last-finishing SCCs.**

- Consider the $(t+1)^{st}$ tree produced, suppose the root is **x**.

- Suppose that **x** lives in the SCC **A**.

- Then **A**.finish > **B**.finish for all remaining SCCs **B**.
  - This is because we chose **x** to have the largest finish time.

- Then there are no edges leaving **A** in the remaining SCC DAG.
  - This follows from the Corollary.

- Then DFS started at **x** recovers exactly **A**.
  - It doesn't recover any more since nothing else is reachable.
  - It doesn't recover any less since A is strongly connected.
  - (Notice that we are using that A is still strongly connected when we reverse all the edges).

- **So the $(t+1)^{st}$ tree is the SCC with the $(t+1)^{st}$ biggest finish time.**

Recap of proof idea

- The finish times of the first DFS basically do topological sort on the SCC DAG.

- Thus, when we reverse all the edges we have:

 - If there is an edge like this in the reversed graph:
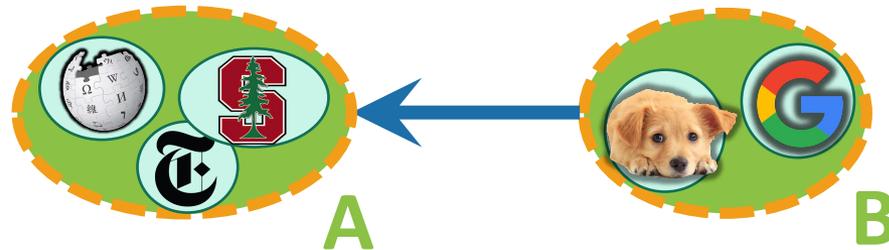
  - Then A.finish > B.finish.

- Then when we do DFS again in reverse order of finish time, we pick off the SCCs.

Punchline:
we can find SCCs in time O(n + m)

`Algorithm:`

- Do DFS to create a **DFS forest**.
  - Choose starting vertices in any order.
  - Keep track of finishing times.

- Reverse all the edges in the graph.

- Do DFS again to create **another DFS forest**.
  - This time, order the nodes in the reverse order of the finishing times that they had from the first DFS run.

- The SCCs are the different trees in the **second DFS forest.**
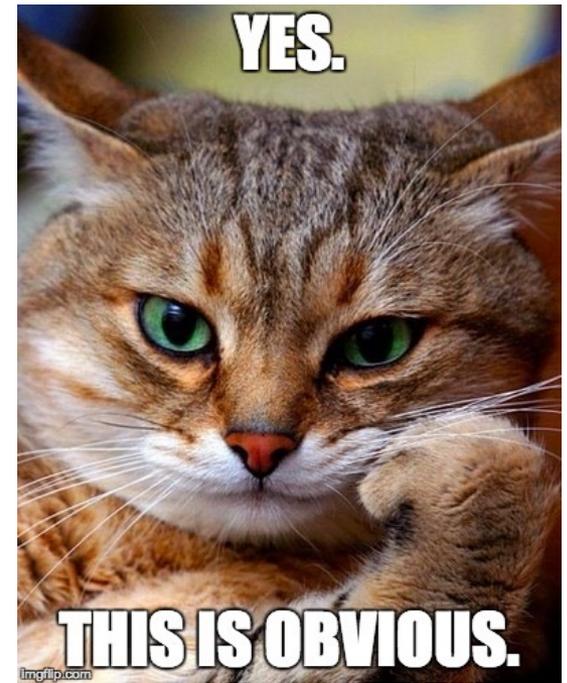
(Clearly it wasn't obvious since it took all class to do! But hopefully it is less mysterious now.)

Recap

- Depth First Search reveals a very useful structure!
  - We saw last week that this structure can be used to do **Topological Sorting** in time O(n+m)

  - Today we saw that it can also find **Strongly Connected Components** in time O(n + m)

  - This was a non-obvious algorithm!
    - (at least, it was non-obvious 80 minutes ago)

Next time

- Dijkstra's algorithm!

BEFORE Next time

- Pre-lecture exercise: weighted graphs!