COS 418 Precept: System Design

Jennifer Lam, Yue Tan

But First…

- System design is more of an art.
  - This precept is more prescriptive than system design is meant to be.
  - Best taken as a starting point.
- Helpful references:
  - Lampson, “Hints for Computer System Design.”
  - Saltzer, “End to End Arguments in System Design.”

Outline

1. A simple example: an interview question.
2. The end-to-end principle in computer networking.

A Simple Example

AN INTERVIEW QUESTION

The Browser History Question

Question: design a class that keeps track of the most recently visited websites.

Breaking It Down: A Sentence

Question: design a class that keeps track of the most recently visited websites.

Breaking it down: **subject/verb/object**

- What is your input? From whom/ how will you receive it? **(subject)**
- What needs to happen input from to output? **(verb)**
- What is your output? In what form will you return it? **(object)**

Fill in the Details

Question: design a class that keeps track of the most recently visited websites.

Input: a stream over time of discrete ticks with website names.

Output: a list of the most recent *n* websites.

Fill in the Details

Question: design a class that keeps track of the most recently visited websites.

Input: a stream over time of discrete ticks with website names.

Output: a list of the most recent *n* websites.

Input / output **informs your interface (API)**.

Fill in the Details

Question: design a class that keeps track of the most recently visited websites.

Input: a stream over time of discrete ticks with website names. **API: OnTick() callback.**

Output: a list of the most recent *n* websites.

Input / output **informs your interface (API)**.

Fill in the Details

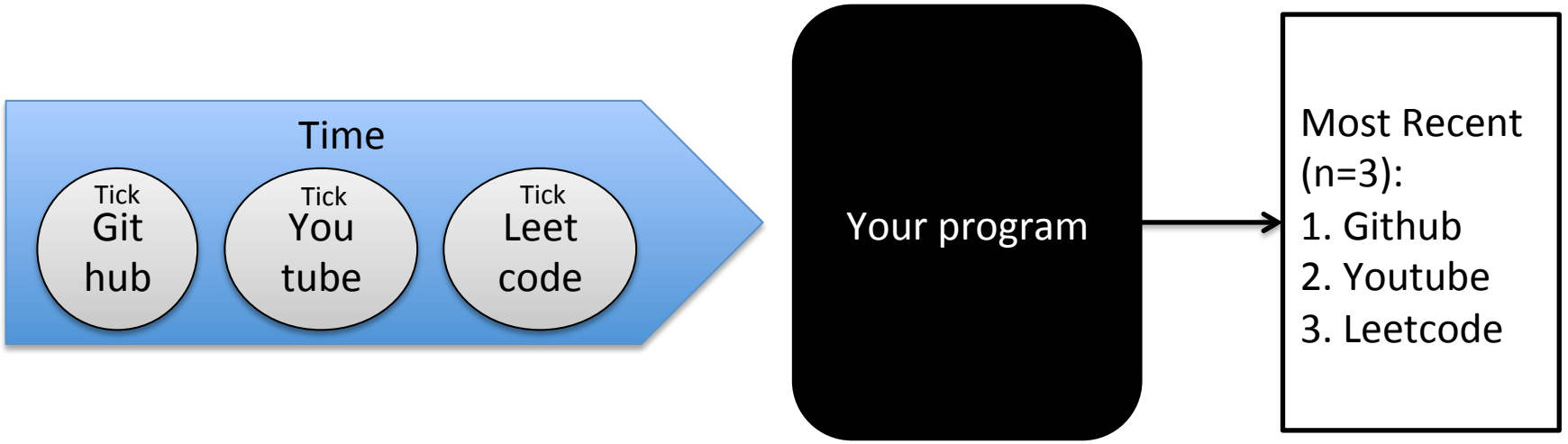
Question: design a class that keeps track of the most recently visited websites.

Input: a stream over time of discrete ticks with website names. **API: OnTick() callback.**

Output: a list of the most recent *n* websites. **API: GetMostRecentWebsites() with some parameter n and some list output.**

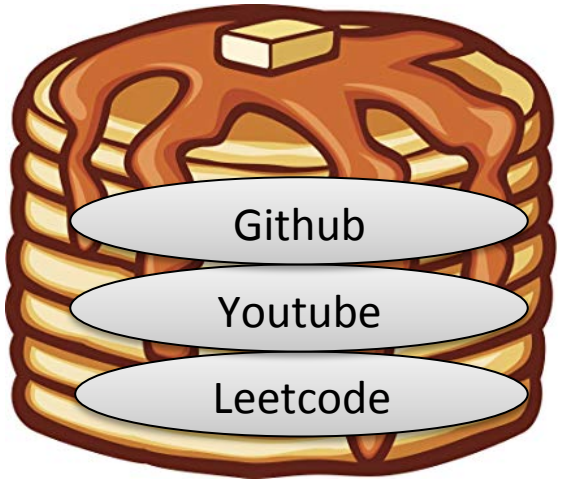
Input / output **informs your interface (API)**.

Start with the Happy Case(s)



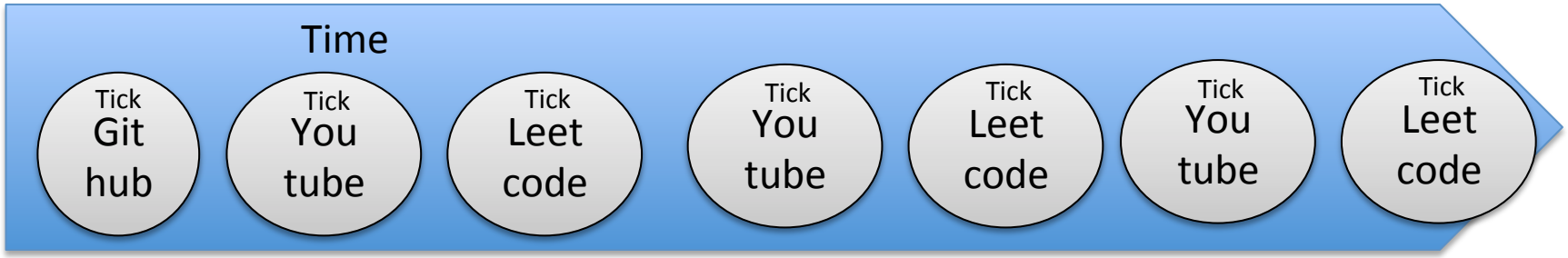
Immediate thoughts:

- List-like structure.
- Retains order of insertion.



Use a stack!

Reason Through the Edge Case(s)



Edge cases:

- Duplicates?
- Large n?

This step is very important.

Most Recent (**n=1000**):

1. Github
2. Youtube
3. Leetcode

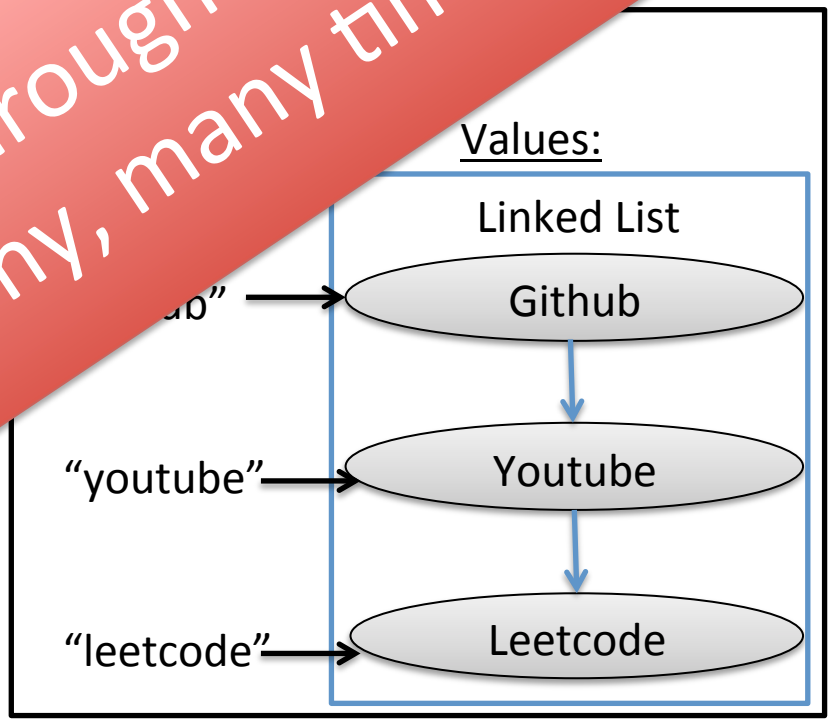
4, 5, 6, ….???

Reason through the Edge Case(s)

Iterate through your design many, many times!

Let's consid
duplica

New d
- List-l
- Retent
- **Set or m**
- **Linked list.**



Trade Offs and Assumptions

Comparing our strawman vs. our new design:

- Is there any case when a stack >> map + linked list?
- What is your expected workload?
  - Is our map + linked list design better for read-mostly or write-mostly?

What assumptions did we make? Are they all correct?

- Concurrency?
- Size of memory stack?

Iterating on Your Design

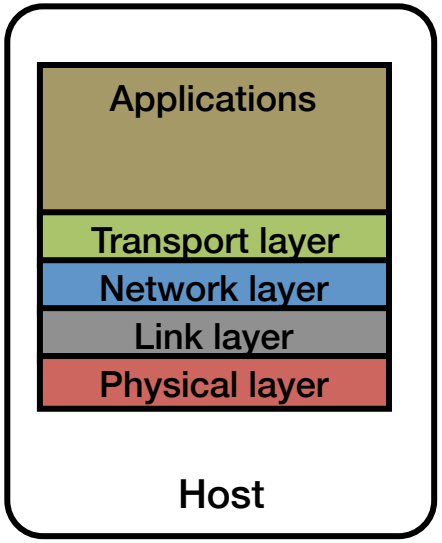
- You understand a little more about your design when you start implementing it.
  - If your design is too difficult to implement, consider that it may not be the right one.
  - A good chance to correct your assumptions.

Encapsulation and Functionality Placement

THE END-TO-END PRINCIPLE

The End-to-End Principle

- Recall the layered network stack from Lecture 2.
- Each layer encapsulates a set of functionalities and only interacts with the layer immediately below it.



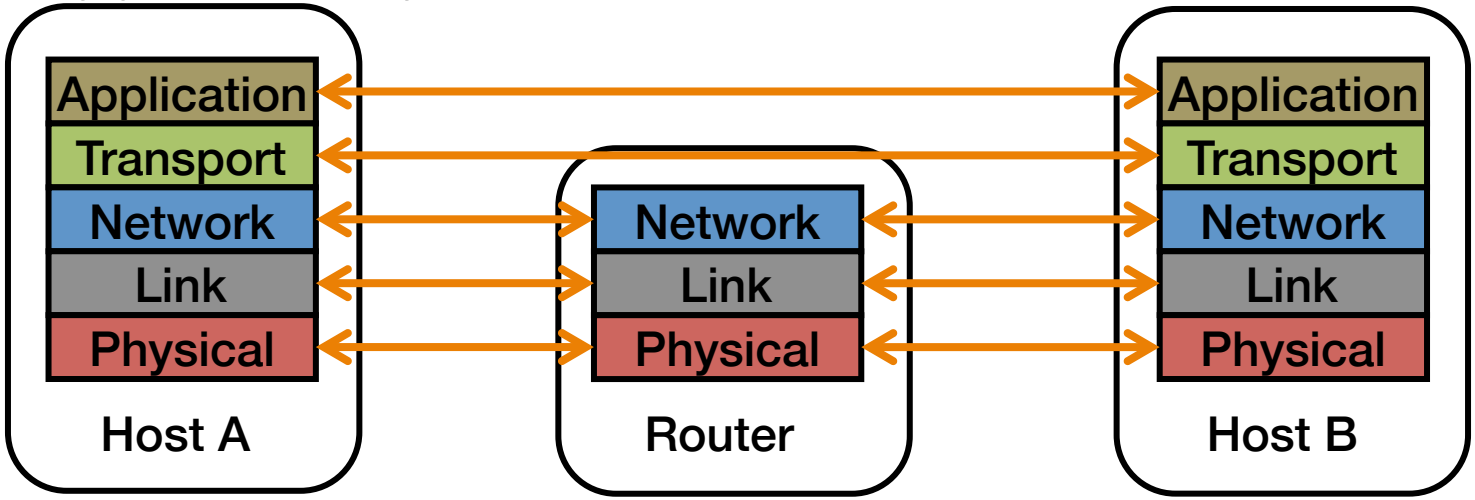
- **Transport:** Provide end-to-end communication between **processes** on different hosts
- **Network:** Deliver packets to hosts on other (heterogeneous) networks
- **Link:** Enables end hosts to exchange atomic messages with each other
- **Physical:** Moves bits between two hosts connected by a physical link

The End-to-End Principle

Recall that a router only implements three layers. The network layer delivers packets to the next node by best efforts.

We push extra functionalities to higher layers and onto end hosts!

e.g. process to process communication and guaranteed in-order delivery provided by TCP



The End-to-End Principle

Why?

- Routers knows nothing about end host processes
  - The network is simply incapable of providing process-to-process communication.
- Cost of extra functionalities can be penalties
  - Not all applications want in-order guaranteed delivery service, e.g. video streaming.
  - And really, applications don't care if a packet is guaranteed to be delivered from one router to another. They want end-to-end guaranteed delivery.

Summary

- Encapsulation
  - Functionalities are encapsulated behind APIs.
- Functionality placement:
  - who should do what
    - E.g. IP only does best-effort delivery and TCP takes care of out-of-order/lost packets.
  - Who can do what
    - E.g. the transport layer is able to provide process-to-process communication.

And Last…

- Again, system design is an ART.
  - There are many, many ways to design a system. We just gave you one of many.
  - You are free to take what you need from this precept and disregard the rest.