Genome 540 discussion

March 6th, 2025
Joe Min

Genome Sciences
UNIVERSITY OF WASHINGTON

Agenda

A short history of programming languages

Microservice architecture in web applications

A short history of programming languages

General purpose programming languages

What we know and love

Used for making:

- Data analysis pipelines
- Websites
- Machine learning models

https://cdn.mycplus.com/mycplus/wp-content/uploads/2009/02/programming-languages.jpg

How we got here: punch cards

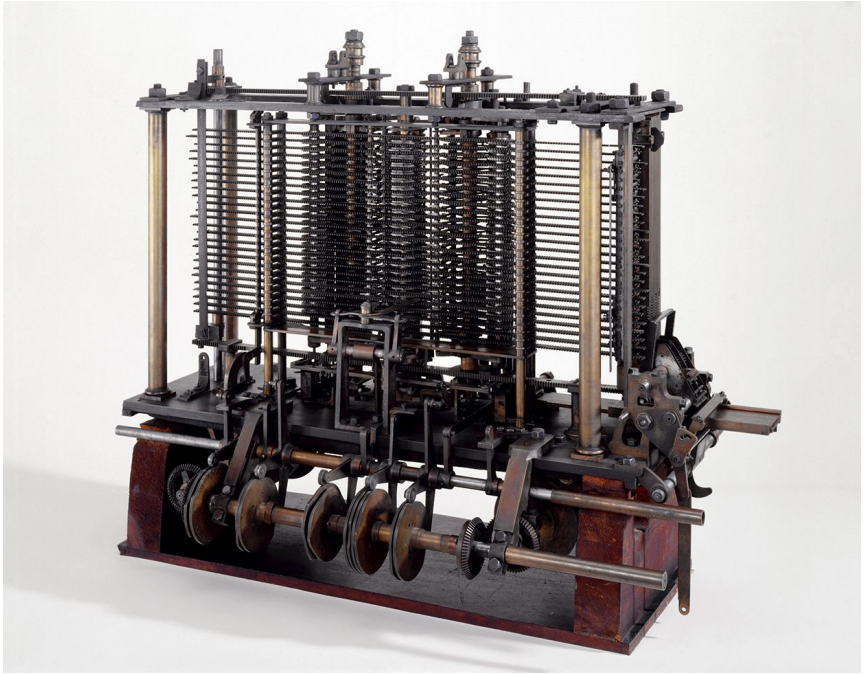
1804 (!!): Joseph Marie Jacquard

Textiles could be woven into repeatable patterns using punch cards

Holes in the cards influence the loom mechanically

https://www.mingei-project.eu/2020/04/02/the-story-of-jacquard-weaving/
https://en.wikipedia.org/wiki/Jacquard_machine

How we got here: punch cards



First described in **1837**, by Charles Babbage

Similar punch card concept

“Memory” of 1000 numbers of up to 40 digits (~17kb)

Steam powered???

https://en.wikipedia.org/wiki/Analytical_engine#/media/File:Babbages_Analytical_Engine,_1834-1871._(9660574685).jpg

How we got here: punch cards

First “program” was made by Ada Lovelace to calculate the Bernoulli numbers

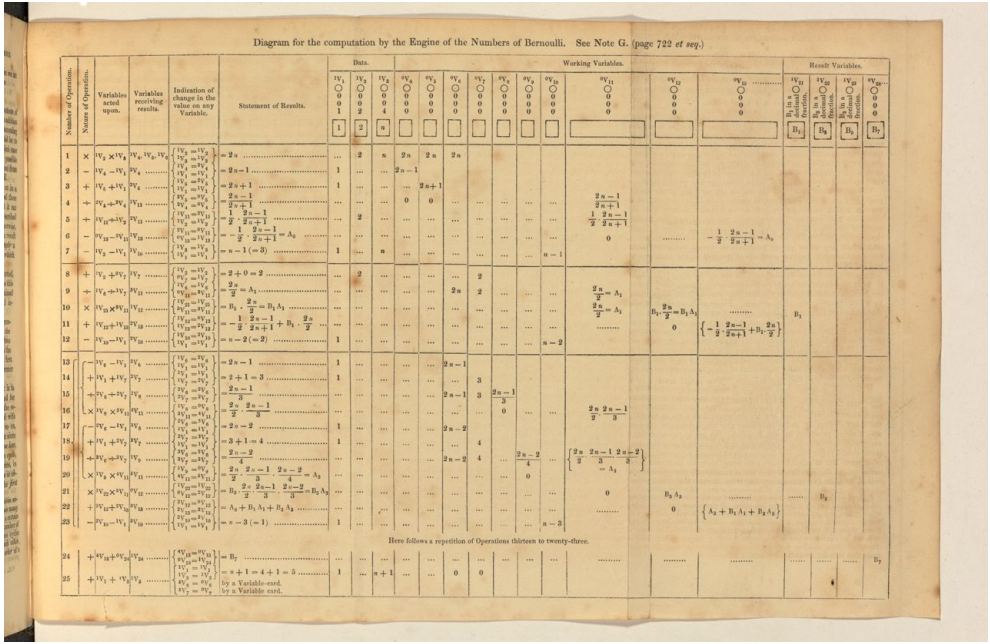


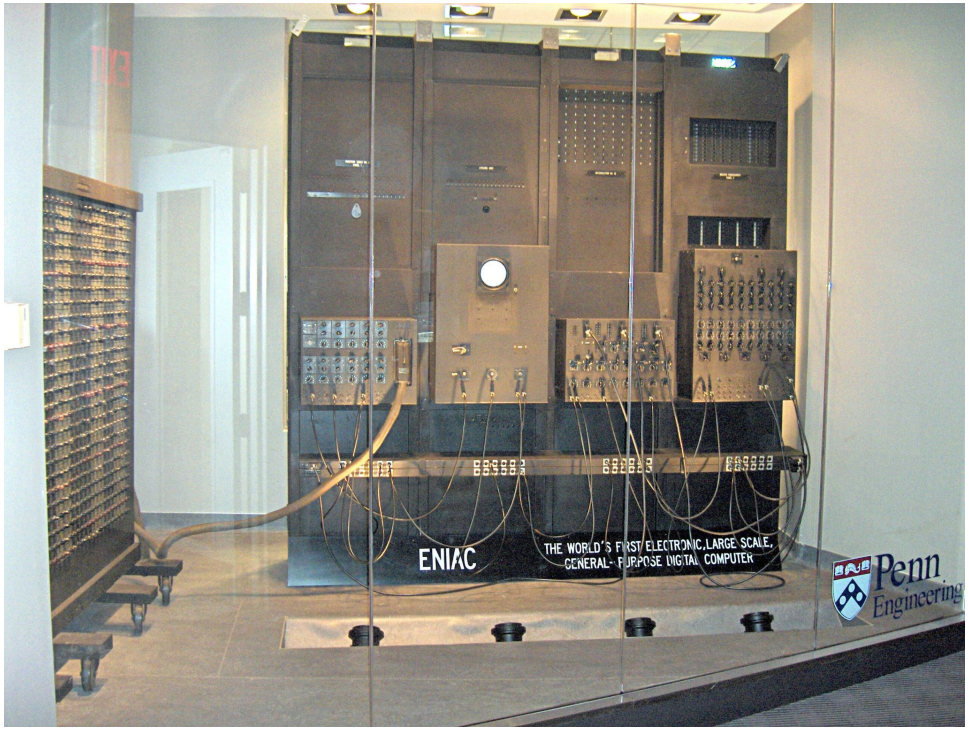
Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

https://blogs.bodleian.ox.ac.uk/adalovelace/2018/07/26/ada-lovelace-and-the-analytical-engine/
https://en.wikipedia.org/wiki/Analytical_engine

How we got here: machine code

1945: John Mauchly and J. Presper Eckert introduce the first general purpose electronic computer, **E**lectronic **N**umerical **I**ntegrator **A**nd **C**omputer (ENIAC)

All binary instructions

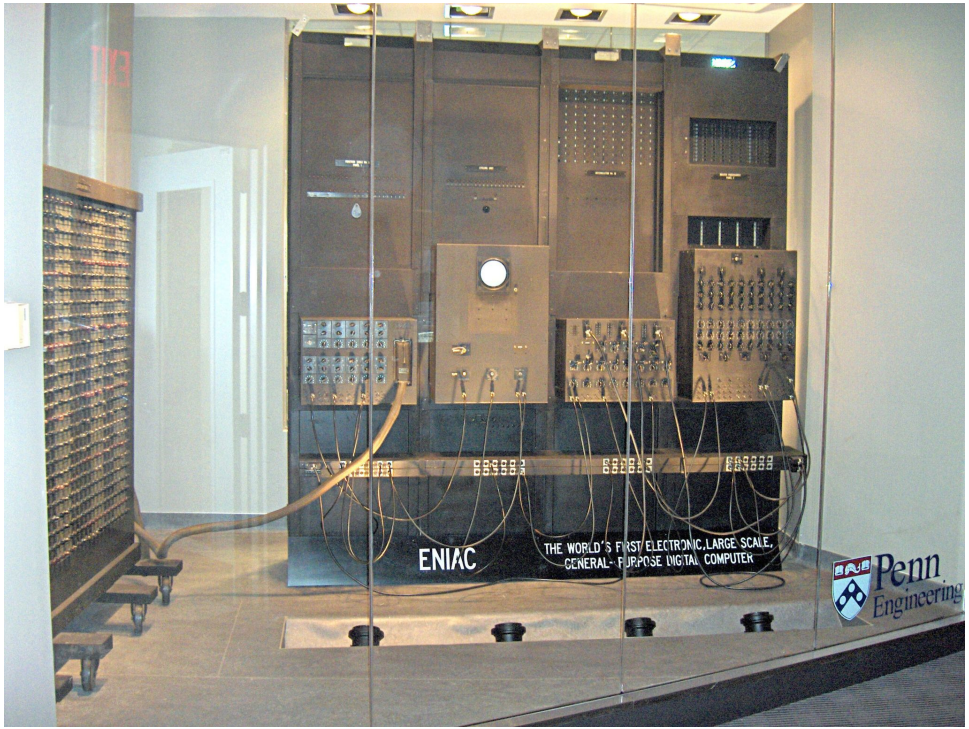


https://upload.wikimedia.org/wikipedia/commons/thumb/6/6c/ENIAC_Penn1.jpg/440px-ENIAC_Penn1.jpg

How we got here: machine code

ENIAC is basically a set of pre-programmed functional units wired up to each other

E.g., if you wanted to take a product after a sum, you would run a wire from the multiplier to the adder



https://upload.wikimedia.org/wikipedia/commons/thumb/6/6c/ENIAC_Penn1.jpg/440px-ENIAC_Penn1.jpg

How we got here: assembly code

In **1951**, the UNIVAC I (Universal Automatic Computer I) came on the scene

Used assembly instead of raw binary instructions

https://en.wikipedia.org/wiki/UNIVAC#/media/File:Univac_I_Census_dedication.jpg

How we got here: assembly code

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


 C000                    ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START    LDS    #STACK

                *****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                *****************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH     LDA A  ACIA      GET STATUS
 C013 47                 ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA              BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05           LDA A  ACIA+1    GET CHAR
 C019 84 7F              AND A  #$7F      MASK PARITY
 C01B 7E C0 79           JMP    OUTCH     ECHO & RTS
```

In **1951**, the UNIVAC I (Universal Automatic Computer I) came on the scene

Used assembly instead of raw binary instructions

https://en.wikipedia.org/wiki/Assembly_language#/media/File:Motorola_6800_Assembly_Language.png

How we got here: human readable code

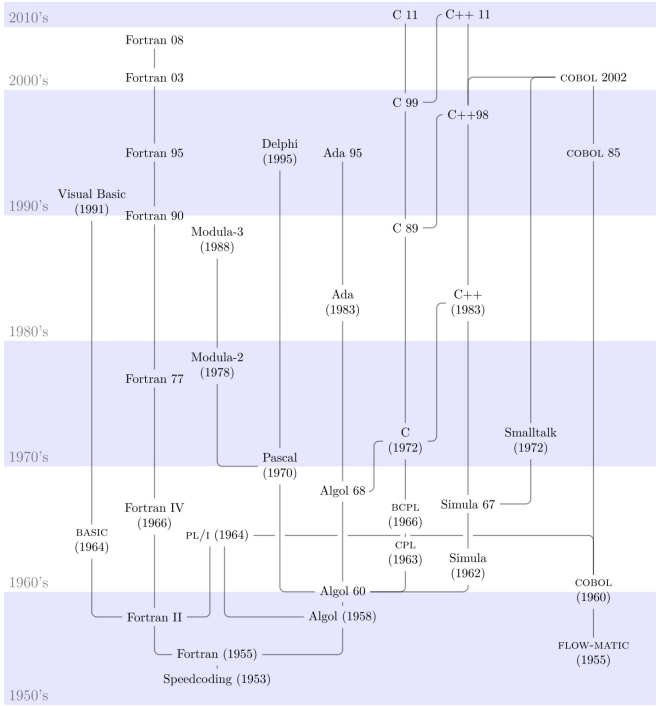
At the Eckert–Mauchly Computer Corporation, Grace Hopper pushed for human-readable programming languages, eventually leading to FLOW-MATIC in **1955**

This was the first English-like programming language

```
(0)  INPUT INVENTORY FILE-A PRICE FILE-B ; OUTPUT PRICED-INV FILE-C UNPRICED-INV
    FILE-D ; HSP D .
(1)  COMPARE PRODUCT-NO (A) WITH PRODUCT-NO (B) ; IF GREATER GO TO OPERATION 10 ;
    IF EQUAL GO TO OPERATION 5 ; OTHERWISE GO TO OPERATION 2 .
(2)  TRANSFER A TO D .
(3)  WRITE-ITEM D .
(4)  JUMP TO OPERATION 8 .
(5)  TRANSFER A TO C .
(6)  MOVE UNIT-PRICE (B) TO UNIT-PRICE (C) .
(7)  WRITE-ITEM C .
(8)  READ-ITEM A ; IF END OF DATA GO TO OPERATION 14 .
(9)  JUMP TO OPERATION 1 .
(10)  READ-ITEM B ; IF END OF DATA GO TO OPERATION 12 .
(11)  JUMP TO OPERATION 1 .
(12)  SET OPERATION 9 TO GO TO OPERATION 2 .
(13)  JUMP TO OPERATION 2 .
(14)  TEST PRODUCT-NO (B) AGAINST ; IF EQUAL GO TO OPERATION 16 ;
    OTHERWISE GO TO OPERATION 15 .
(15)  REWIND B .
(16)  CLOSE-OUT FILES C ; D .
(17)  STOP . (END)
```

https://en.wikipedia.org/wiki/FLOW-MATIC

How we got here: high-level languages



Fortran (FORTRAN) first compiled correctly in **1958**

Developed at IBM by a team led by John W. Backus

Kicked off the era of high-level programming languages

https://en.wikipedia.org/wiki/Fortran#/media/File:Algol&Fortran_Family_By_Skippppp.svg

Microservice architecture in web applications

First, an aside

This blog post was published in 2014 but it holds true to this day over 10 years later

All code is bad

Every programmer occasionally, when nobody's home, turns off the lights, pours a glass of scotch, puts on some light German electronica, and opens up a file on their computer. It's a different file for every programmer. Sometimes they wrote it, sometimes they found it and knew they had to save it. They read over the lines, and weep at their beauty, then the tears turn bitter as they remember the rest of the files and the inevitable collapse of all that is good and true in the world.

This file is Good Code. It has sensible and consistent names for functions and variables. It's concise. It doesn't do anything obviously stupid. It has never had to live in the wild, or answer to a sales team. It does exactly one, mundane, specific thing, and it does it well. It was written by a single person, and never touched by another. It reads like poetry written by someone over thirty.

https://www.stilldrinking.org/programming-sucks

First, an aside

This blog post was published in 2014 but it holds true to this day over 10 years later

> Every programmer starts out writing some perfect little snowflake like this. Then they're told on Friday they need to have six hundred snowflakes written by Tuesday, so they cheat a bit here and there and maybe copy a few snowflakes and try to stick them together or they have to ask a coworker to work on one who melts it and then all the programmers' snowflakes get dumped together in some inscrutable shape and somebody leans a Picasso on it because nobody wants to see the cat urine soaking into all your broken snowflakes melting in the light of day. Next week, everybody shovels more snow on it to keep the Picasso from falling over.

https://www.stilldrinking.org/programming-sucks

Giant, monolithic code is bad

Imagine building an online store. We need:

- User management (registration; logging in; account details)
- Product catalog (item names, price, and availability)
- Orders (shopping cart, payments)

In general all relevant information for these will live in some database (e.g., a SQL database)

Giant, monolithic code is bad

Consider a user logging in, scrolling through the catalog, and adding something to their cart. This requires:

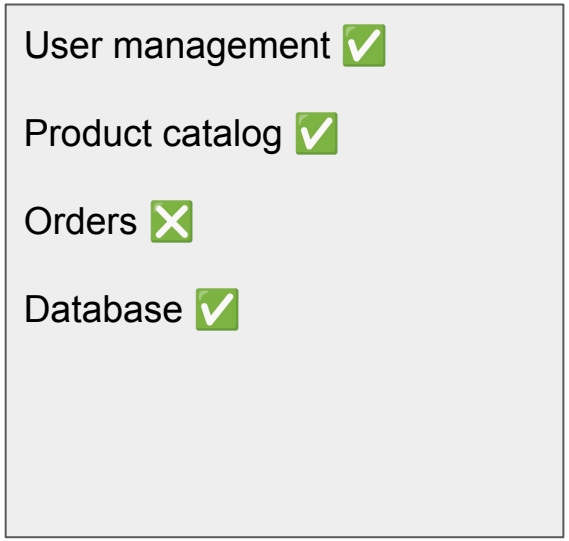
- Securely authenticating the user with their password against the database
- Pulling images and prices for catalog items from the database
- Updating the user’s cart in the database

Giant, monolithic code is bad

If a single part experiences an error, the whole application fails because it’s basically a single copy of a running program

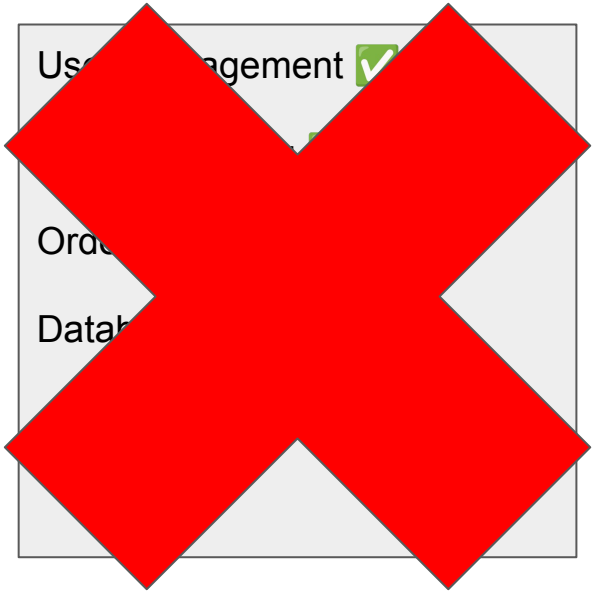
Giant, monolithic code is bad

If a single part experiences an error, the whole application fails because it’s basically a single copy of a running program



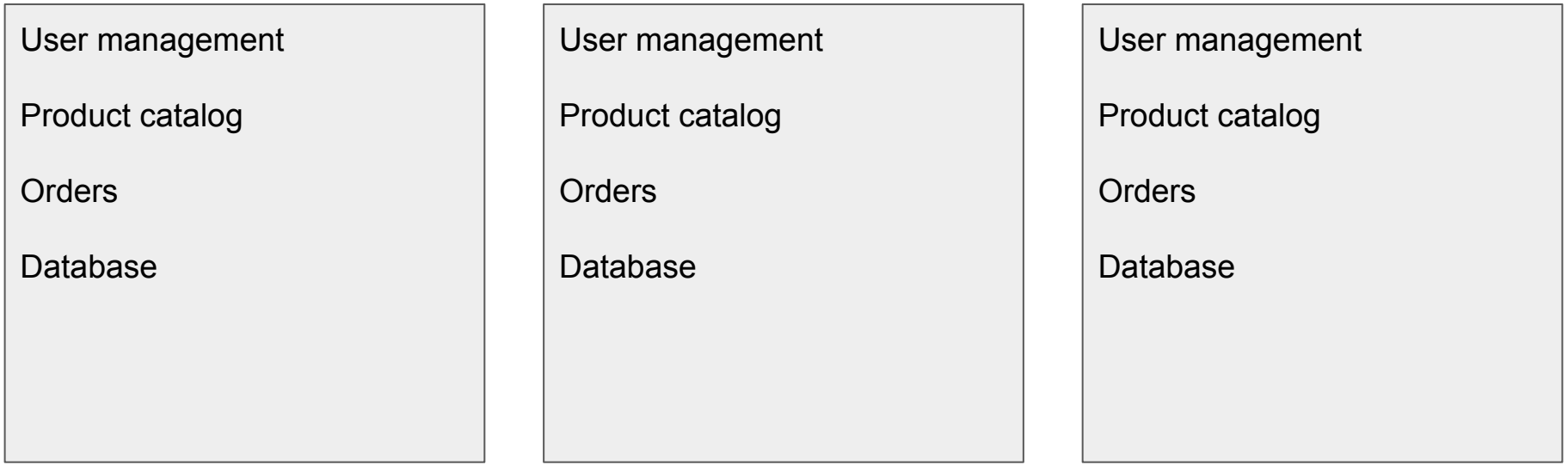
Giant, monolithic code is bad

If a single part experiences an error, the whole application fails because it’s basically a single copy of a running program



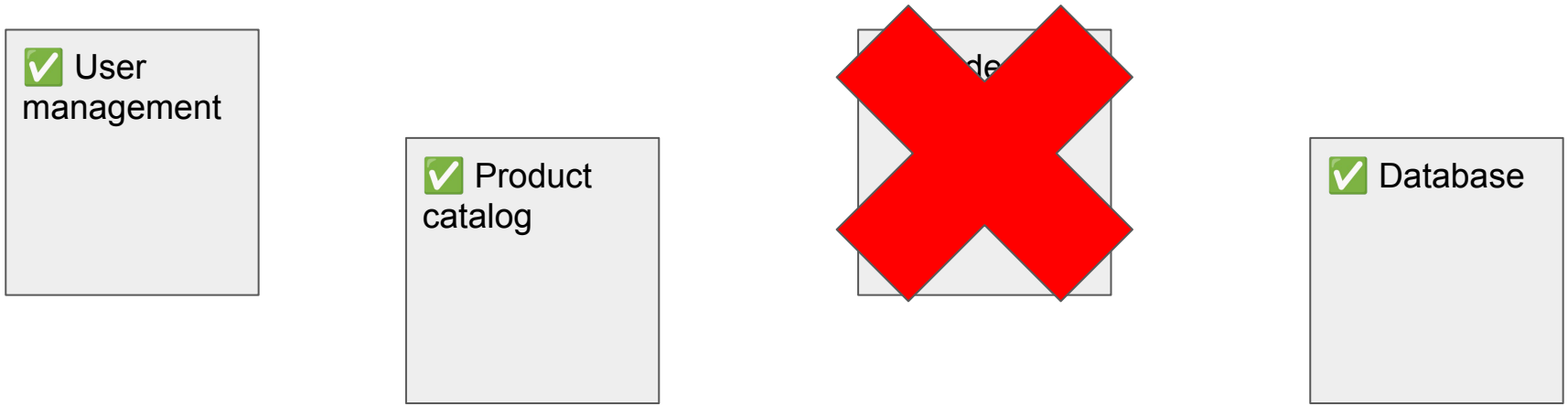
Giant, monolithic code is bad

We could scale out, but each additional copy is huge



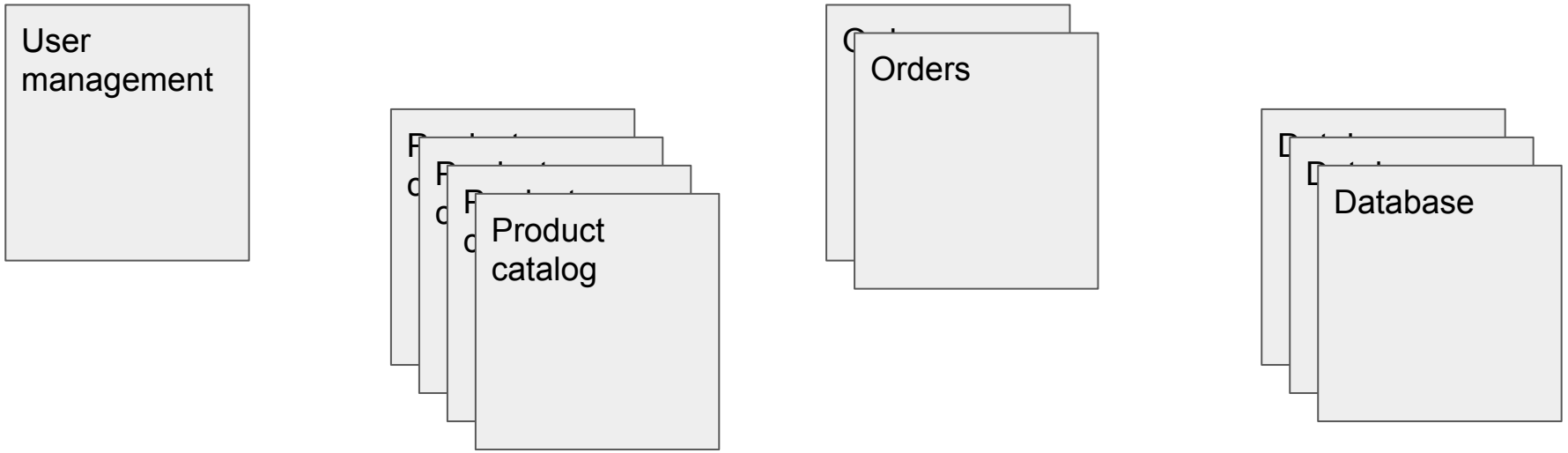
Small microservices can be better

If, instead, we split this into many different programs running separately, even if one service goes down, all other functionality is maintained!



Small microservices can be better

We can also scale out only the services that need to be scaled out (e.g., the most used)



Office hours

Reminder:

Homework 8 is due Sunday, March 9th at 11:59pm!