L23: What Makes Android a Good OS?

(Code from Classroom to Global Stage)

Xin Liu

Florida State University
xl24j@fsu.edu

COP 4610 Operating Systems
https://xinliulab.github.io/cop4610.html
December 4, 2024

Overview

Review:

- **Application Perspective of Operating Systems**
  - Objects + APIs
- **Hardware Perspective of Operating Systems**
  - A program that controls the entire computer hardware

Question for This Lecture:

- **Q: What does a truly ”practical” operating system require?**

Main Content for This Lecture:

- Android Applications and System

Towards the Mobile Internet Era

The Transformation of Our World

We can no longer imagine life without mobile phones.

Enjoy: Nokia Ringtone Evolution

Again and Again, Changing the World!

The first smartphone: iphone 2G, 2007.

Android

Android Official Website

- **Linux + Framework + JVM**
  - Is it secondary development on Linux/Java?
  - Not exactly: Android defines an application model.
- Supporting Java was a highly visionary decision.
  - **Qualcomm MSM7201:**
    - ARMv6 instruction set
    - 528MHz x 1CPU, in-order 8-stage pipeline
    - TSMC 90nm
- "Even running a map app would lag..."
  - But Moore's Law came to the rescue!

The first Android phone: HTC G1, 2008

Android Apps: Write Once, Run Anywhere

An application running on the Java Virtual Machine (Android Runtime):

- **Platform (Framework)**
- **NDK (Native Development Kit)**
- **Java Native Interface (C/C++ Code)**

Official Documentation (RTFM):

- **Kotlin**
- **Platform**
  - **android.view.View:** “the basic building block for user interface components”
  - **android.webkit.WebView:** Embedding web pages in your app
  - **android.hardware.camera2:** Camera
  - **android.database.database:** Database

Comparison: Symbian (C++) vs. Android (Java)

Symbian (C++)

- **Manual Memory Management**
  - Developers use `malloc/free`, prone to memory leaks.
  - Risks: dangling pointers, segmentation faults.
- **Pointer Operations**
  - Direct pointer manipulation increases error risks.
  - Debugging pointer issues is time-consuming.
- **Complex Syntax**
  - C++ syntax is powerful but hard to maintain.
  - Error handling is not standardized (e.g., exceptions).

Android (Java)

- **Automatic Memory Management**
  - Garbage Collection handles memory allocation and release.
  - Fewer memory leaks and dangling references.
- **No Pointer Exposure**
  - Object references instead of raw pointers.
  - Safer access to memory and resources.
- **Simplified Syntax**
  - Built-in exception handling.
  - Higher-level APIs reduce coding complexity.

Key Takeaways

Symbian (C++):

- Powerful and performance-oriented but requires expert-level skills.
- Higher error rates due to manual memory management and pointer issues.

Android (Java):

- Developer-friendly with built-in safety mechanisms.
- Automatic garbage collection simplifies memory management.
- Encourages faster application development and wider adoption.

Conclusion: Java's design prioritizes developer productivity and safety, making it a better fit for large-scale mobile application ecosystems.

The Strategic Bet on Java and Moore's Law

Challenges at the time:

- Limited processing power on mobile chips.
- Java's higher demands on hardware made it seem risky.

Key Assumption:

- Inspired by Intel's Pentium and multi-core advancements in PCs.
- *What happened in PC chip development will repeat in mobile chips.*

Vision:

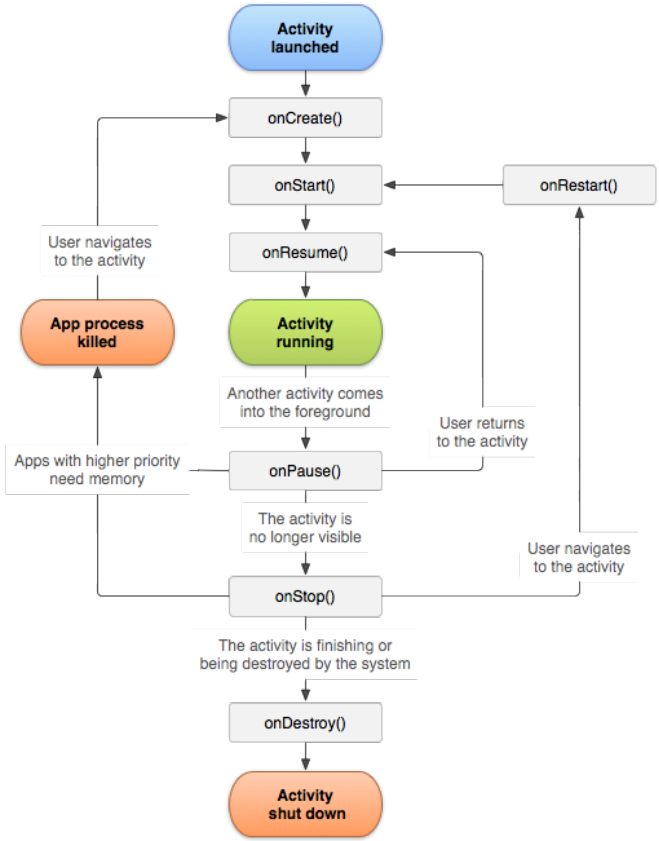
- Moore's Law predicted rapid improvement in chip performance.
- As hardware evolved, Java's demands would no longer be a bottleneck.

Lessons from History: Design for the future, not for the present.

Android Apps

Four Components of Android

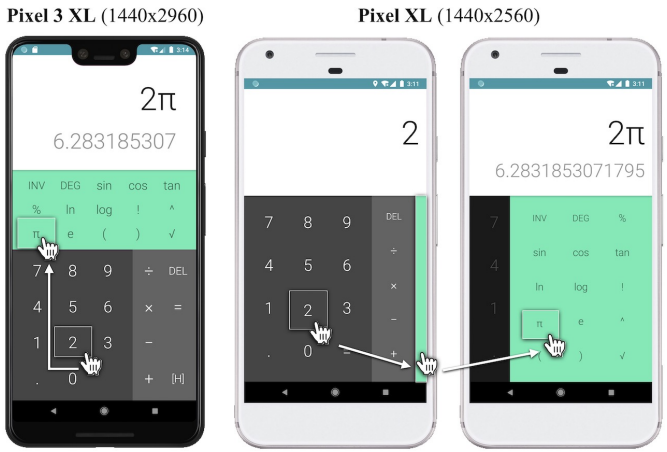
- **Activity**
  - UI interface of the application (Event Driven)
  - Exists in an Activity Stack (Application back stack)
- **Service**
  - Background service without UI
- **Broadcast**
  - Receives system messages and responds
  - Examples: ”Plugged in power”, ”WiFi disconnected”
- **ContentProvider**
  - Shares data storage across applications (insert, update, query, ...)



Example: Calculator

Calculator

- **AndroidManifest.xml - Application Metadata**
  - For example: required permissions, listening Intents, etc.
- **res - Resource Files**
  - Translations for different languages
  - Image files (e.g., icons)
- Writing an application logic only requires overriding `onCreate` in the Activity class.

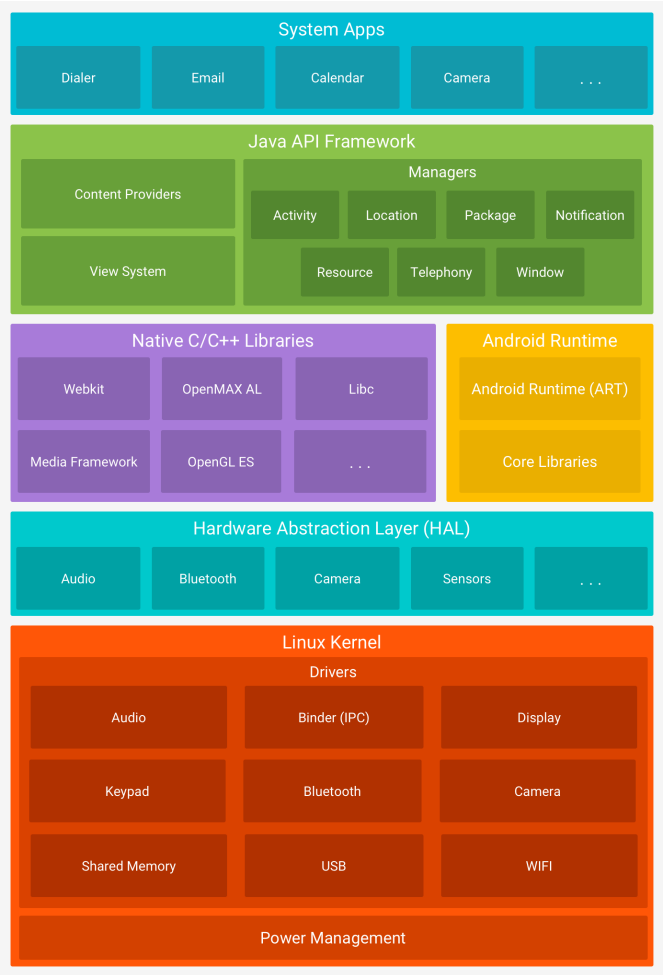


Android Operating System

Platform API: A ”Microkernel”

Using ”Binder IPC”

- **Remote Procedure Call (RPC):**
  - `remote.transact()`
- **A balance between performance and ease of use:**
  - **Registration Mechanism**
    - Compared to pipes/sockets, it’s more ”low-level,” requiring too much manual management.
  - **Based on Shared Memory:**
    - Linux Kernel binder driver
  - **Service Thread Pool**



And Then... A Ton of Code

Example: How to kill an Android process?

- **RTFSC:** ActivityManagerService.java
- Each Android App has a unique UID
- Traverse the process list to find processes belonging to the UID
- Process.KillProcessGroup
  - Interval of 5ms, killing continuously 40 times to prevent data race
  - The necessity of Operating System Transactions

Can we exploit data races to keep a process alive?

- Becoming an orphan process won't immediately receive the `SIGKILL` signal
- Upon being killed, another process can be immediately awakened:
  - A lightweight framework for fine-grained lifecycle control of Android applications (EuroSys'19)

A True "Operating System"

adb (Android Debug Bridge):

- `adb push/pull/install`
- `adb shell`
  - `screencap /sdcard/screen.png`
  - `sendevent`
- `adb forward`
- `adb logcat/jdwp`

A series of derived tools:

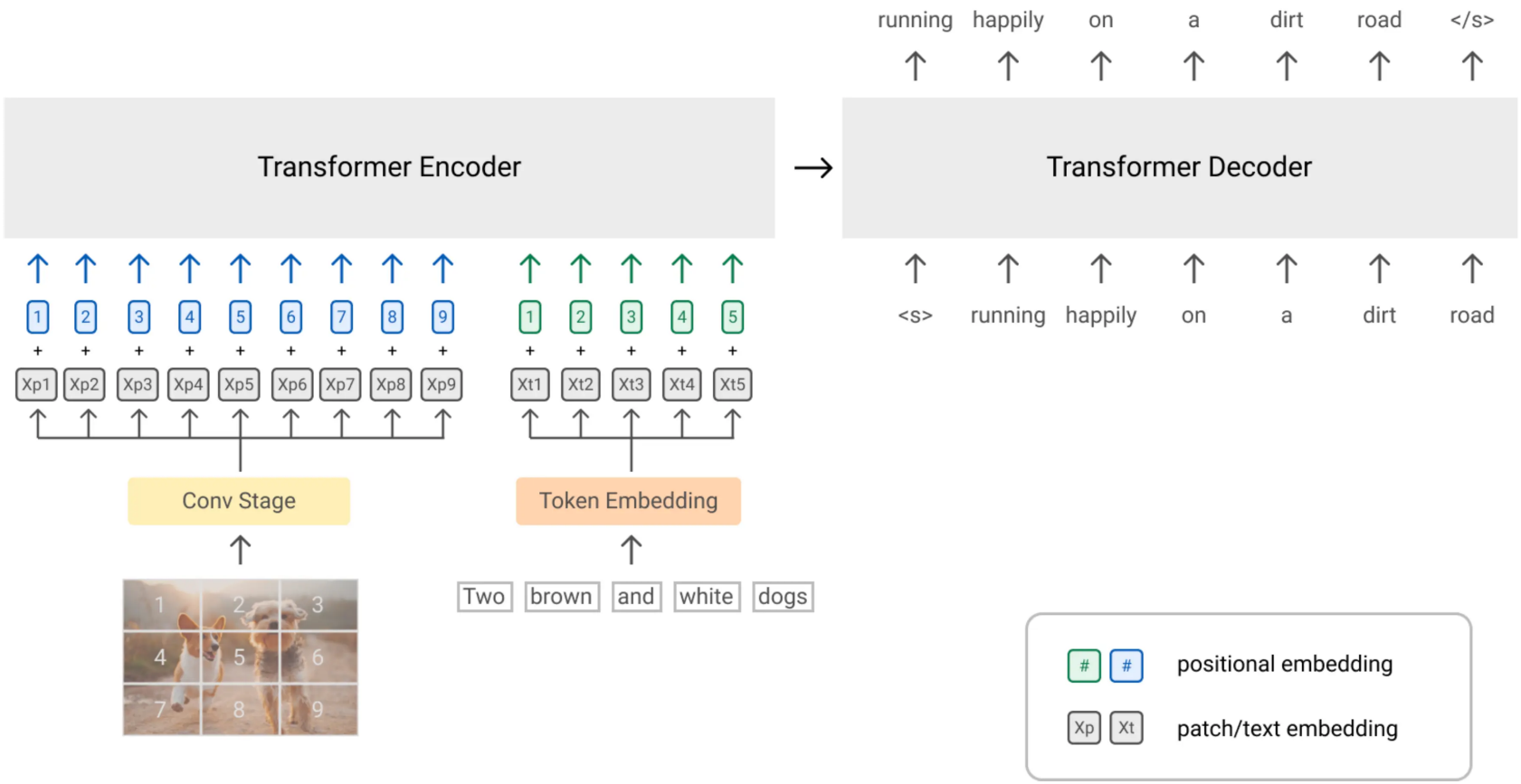
- Developer options
- `scrcpy`
- Monkey / UI Automator

The Era We Live In

A. C. (Anno. ChatGPT) Year 3

AGI is approaching rapidly:

- *Everything is an embedding.* (Below: SimVLM)



We Begin Using LLMs for Complex Programming Tasks

- Explore GCC/LLVM to uncover 100+ bugs.
  - Use ChatGPT to generate test cases.
  - Use ChatGPT to generate Clang AST Transformer.

```
  class ModifyFunctionReturnTypeToVoid : ... {
-   vector<ReturnStmt *> TheReturns;
-   vector<CallExpr *> TheCalls;
+   map<FunctionDecl *, vector<ReturnStmt *>> FuncReturns;
+   map<FunctionDecl *, vector<CallExpr *>> FuncCalls;
  };

  bool ModifyFunctionReturnTypeToVoid::mutate() {
    TraverseAST(getASTContext());
    if (TheFunctions.empty()) return false;

    FunctionDecl *func = randElement(TheFunctions);

    // Change the return type to void
    QualType voidType = getASTContext().VoidTy;
    std::string voidTypeStr = formatAsDecl(voidType, "");

    SourceRange typeRange =
      func->getReturnTypeSourceRange();
    getRewriter().ReplaceText(typeRange, voidTypeStr);
```

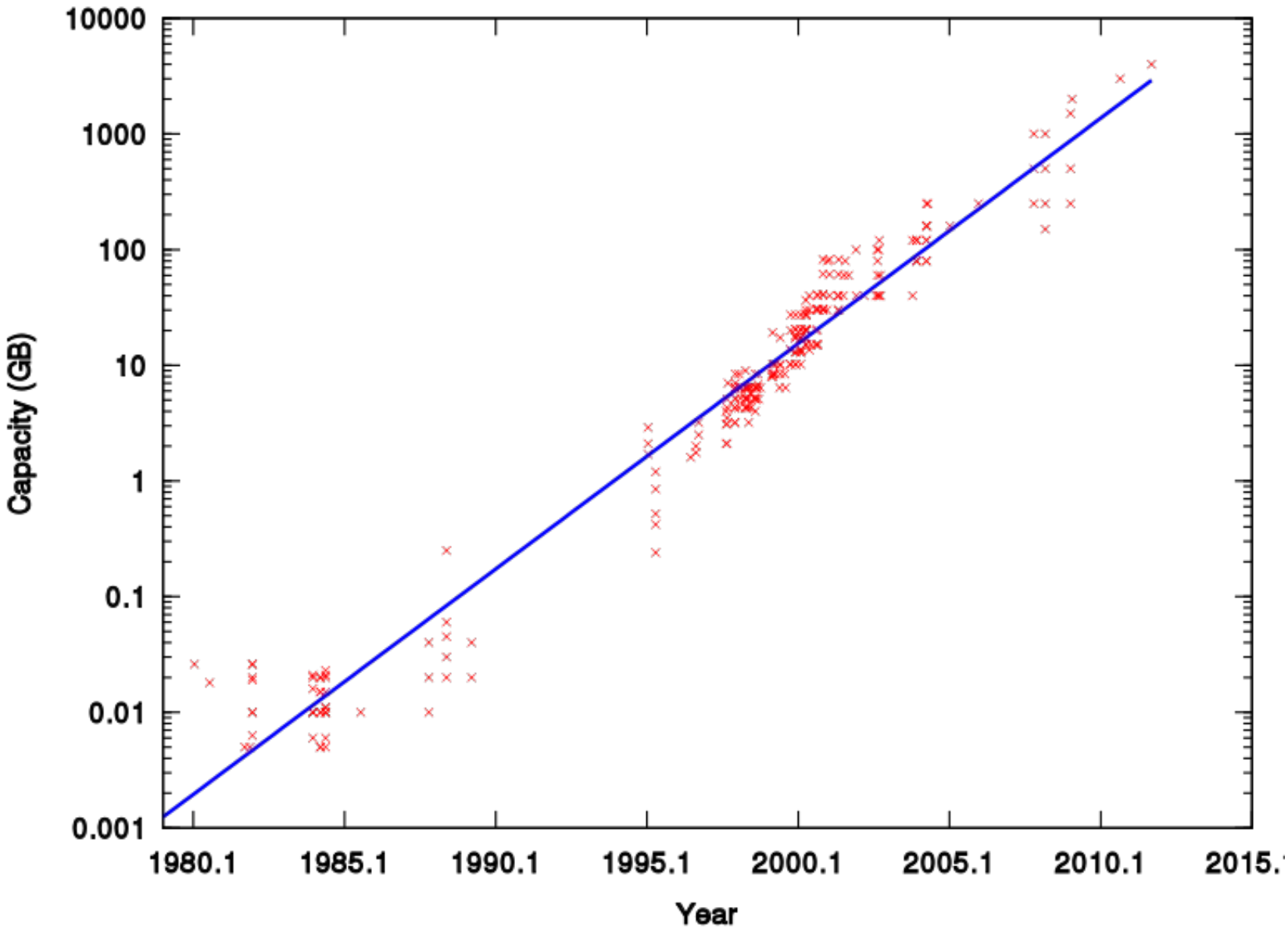
A. C. (Anno. ChatGPT) Year 3

We Begin Experimenting with Natural Programming Languages

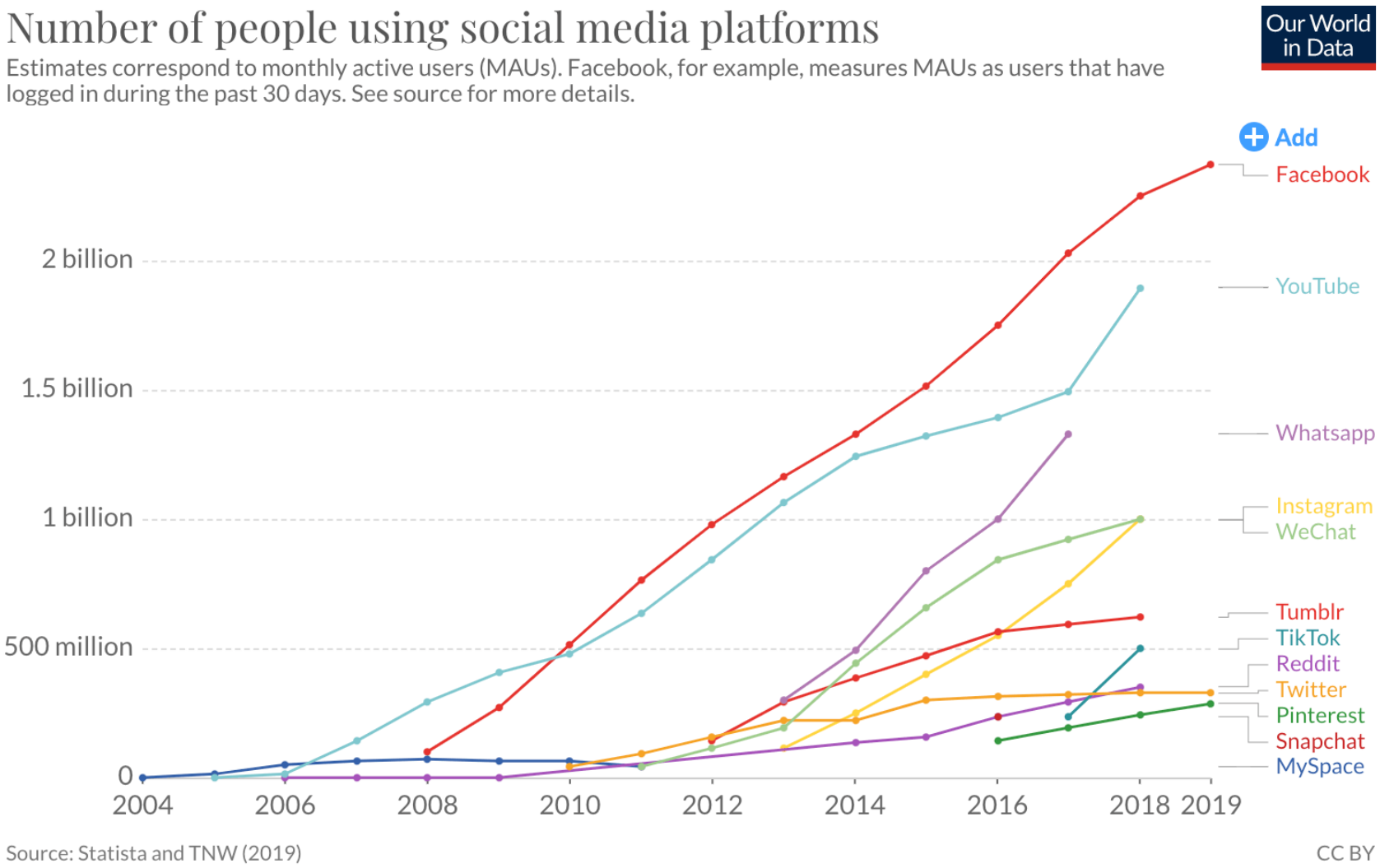
- What should an LLM copilot for command-line tasks look like?
  - **Context:** Shell commands, execution history, and current context.
  - **System prompt:** What am I doing? What should I do next?
  - Workflow integration.
  - ...

Why are we living in such an era?

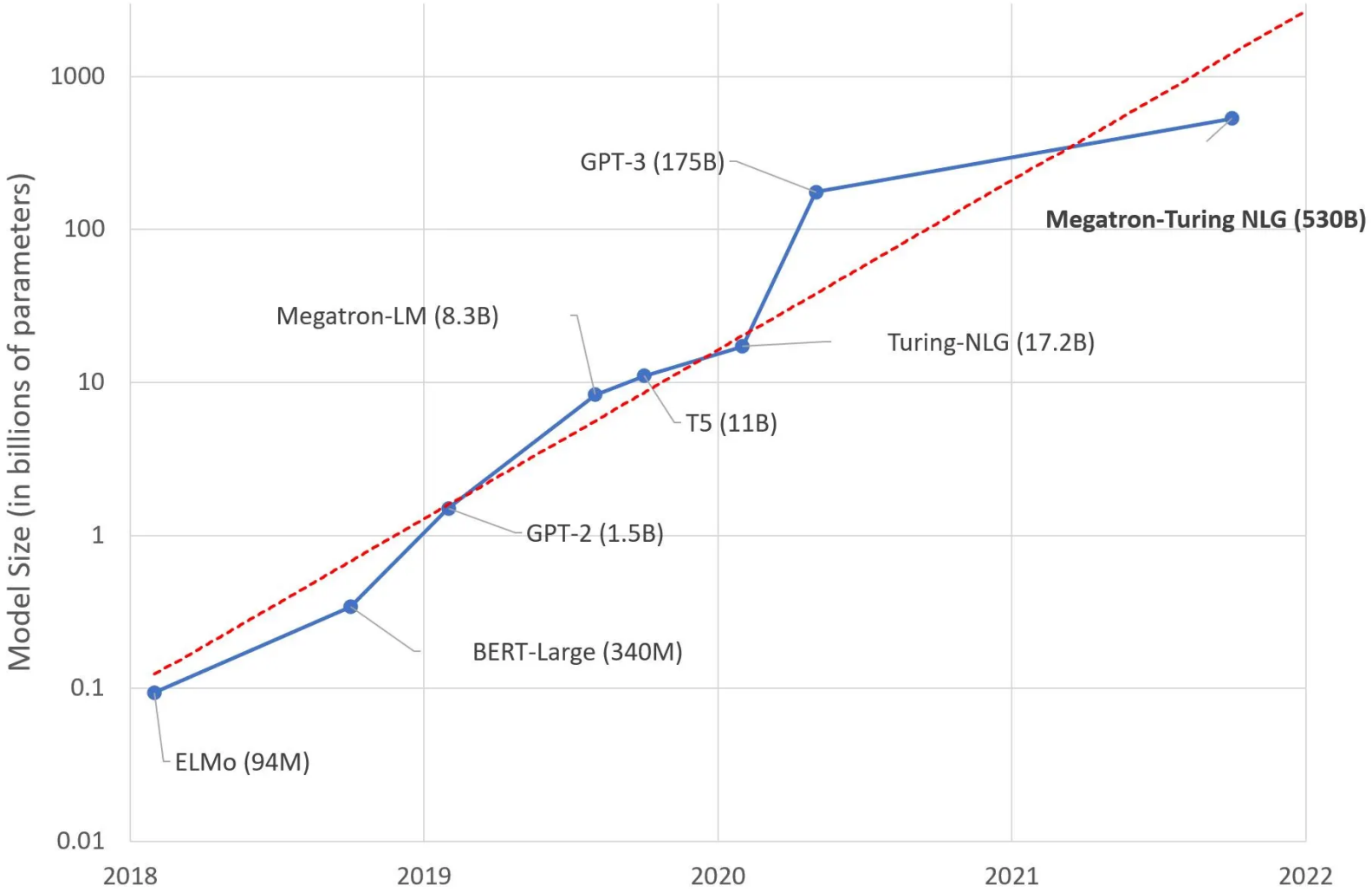
The Prophet of the Era: Google Is No Coincidence



The Prophet of the Era: Facebook Is No Coincidence



The Prophet of the Era: OpenAI Is No Coincidence



The Hero of This World

A computer on every desk and in every home, running Microsoft software.
(1994; Altair BASIC 20 years later)

The Hero of This World

”Nobody actually creates perfect code the first time around, except me. But there’s only one of me.” (2007)

What Makes Us Relentless?

- *"Remember brick walls let us show our dedication. They are there to separate us from the people who don't really want to achieve their childhood dreams."*
- **—— Randy Pausch's Last Lecture**

Do Something the Whole World Only You Believe In

Idea: Start Small, Think Big

- From a simple beginning: 10,000 lines of code ← Linux 0.11

Execution: Scaling Up

- **100,000 lines of code**
  - Laying the foundation: Addressing key technical challenges.
  - Unlocking the potential for scalable and reliable products.
- **1,000,000 lines of code**
  - Solving complex performance bottlenecks.
  - Enabling advanced features and robustness.
- **10,000,000 lines of code**
  - Achieving industrial-grade performance.
  - Delivering a mature, scalable, and widely adoptable product.

Summary

Key Question Addressed Today:

- **Q: How is a truly "practical" operating system (ecosystem) constructed?**

Takeaway Messages:

- **Services, Services, Services**
  - Android Platform API
  - Google Mobile Service (GMS)
  - Countless engineering details
- **Complex systems are everywhere**
  - The role of architects who can drive system-level complexity
  - A large number of highly skilled engineers—where are they?

Congratulations and Best Wishes!

To every student who persisted until the end:

- **Congratulations!**
- May each of you find and pursue your dreams:
  - Become someone who dares to disrupt an industry.
  - Become someone who can manage complex engineering projects.
  - Become someone who can handle large-scale codebases.
- Use your unique way to create something **"breathtakingly brilliant"** and drive progress in the world.

So, persist in writing code!