Introduction

CS 346: Application Development

What’s an “application”?

Thinking about the current software development landscape.

What is *software*?

A set of instructions that tell a machine what to do.

- Produced for a specific hardware system. e.g., ARM, Intel x86.
- Assembled into an executable format for the OS on that hardware.

An executable packaged for distribution

- e.g., ZIP file containing executable + supporting files

```
$ file /bin/bash
/bin/bash: Mach-O universal binary with 2 architectures:
        [x86_64:Mach-O 64-bit executable x86_64]       ←——————   executable/lib file format
        [arm64e:Mach-O 64-bit executable arm64e]
/bin/bash (for architecture x86_64):    Mach-O 64-bit executable x86_64    ←——————   architecture
/bin/bash (for architecture arm64e):    Mach-O 64-bit executable arm64e
```

What is an *application*?

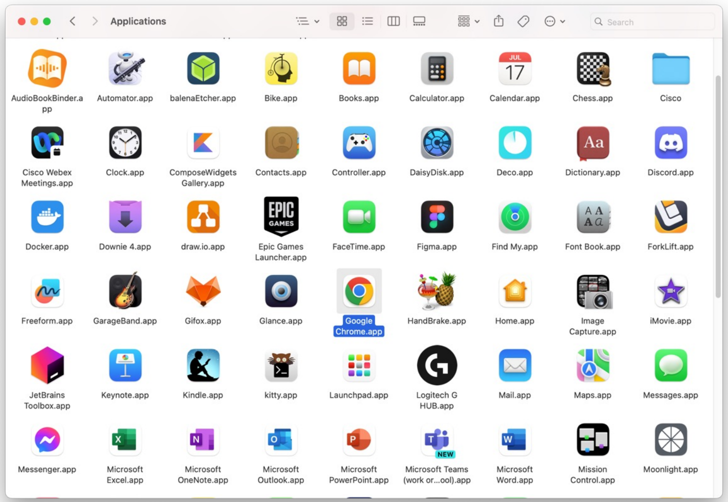
System Software: A specific type of software that exists to provide services to other software. e.g., operating systems, drivers, services.

Application Software: Software that exists to solve problems for users.

- Tends to be interactive software.
- Can include mobile, desktop, web apps.
- e.g., FaceTime, iMovie, Gmail, WhatsApp.

It’s not a clear-cut definition!

- Programming languages? Kiosks?



History of software

How did software development change over time?

1940s

Von Neuman. ENIAC.

No programming languages.

Punch cards.

1950s

Fortran, LISP, Algol, COBOL

Functional programming.

Single programmers.

1960s

PL/1, Basic. Procedural.

Single programmers.

Command-Line

1970s

C, Pascal.

Structured programming.

Abstraction. Small teams.

1980s

C++.

Object-oriented.

Commercial software

Graphical interfaces

1990s

Java, C#.

Internet.

Large teams.

Open Source rises!

2000s

JavaScript, Python, Ruby.
Scripting.

Distributed teams.

Internet dominates

2010

RUST, Dart, Kotlin, Scala.

Web, Parallel computing,
Concurrency

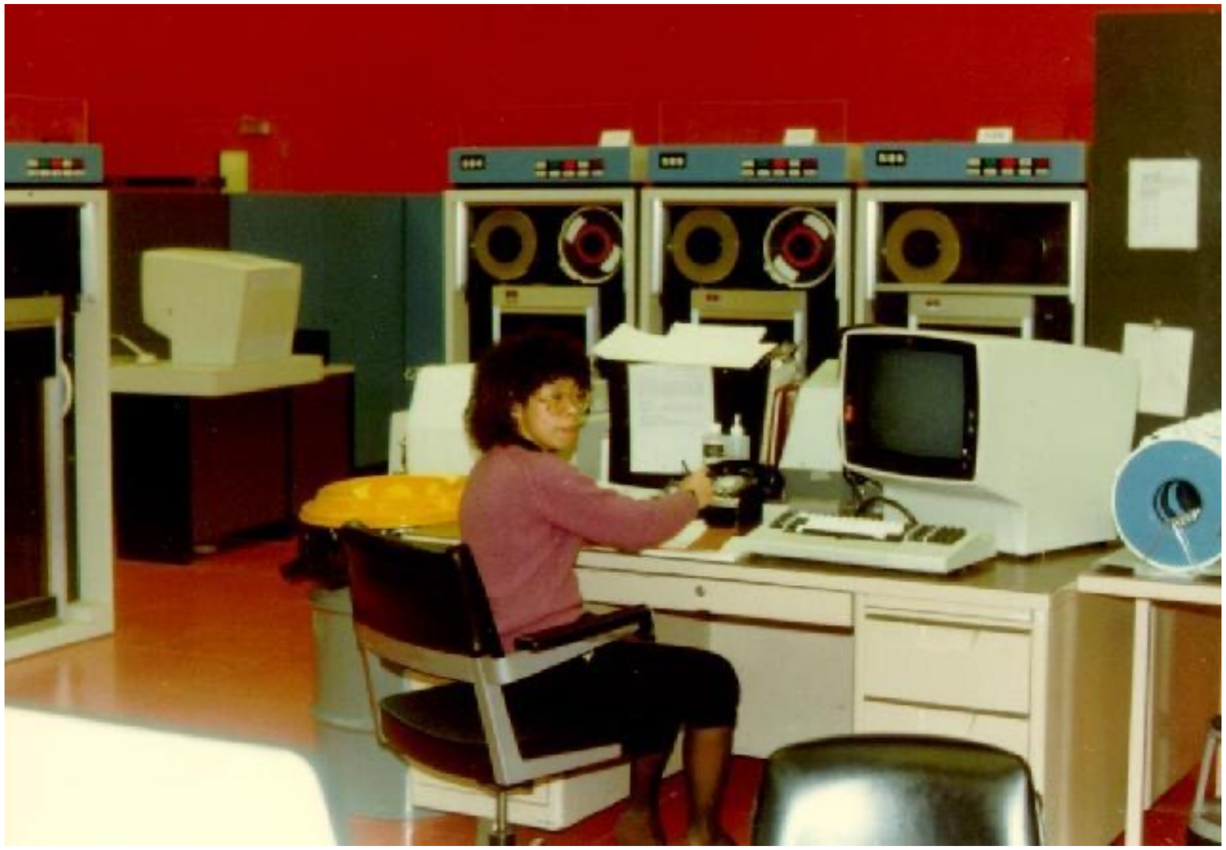
Mobile applications

1960s: Terminals

In the early days of computers, mainframe computers dominated; they were large, expensive, and only available to large institutions. Batch processing was common, where users would submit their jobs (programs), and wait for the results.

By the mid 1960s, we saw the introduction of terminals and time-sharing systems.

```
SLMIN .SYS    12P 20-Dec-85    VM    .SYS     3P 13-Aug-86
XL    .SYS     4P 20-Dec-85    LD    .SYS     8P 23-Aug-86
SP    .SYS     6P 13-Aug-86    DL    .SYS     5P 13-Aug-86
RT11SJ.SYS    78P 13-Aug-86    DU    .SYS     8P 13-Aug-86
NL    .SYS     2P 13-Aug-86    TT    .SYS     2P 13-Aug-86
SQ    .SYS     5P 31-May-85    RL02DG.SYS    71P 21-Nov-84
BASIC .SAV    56  24-May-79    BINCOM.SAV    24  20-Dec-85
DATIME.SAV     4  20-Dec-85    DIR   .SAV    19  20-Dec-85
DUMP  .SAV     9  20-Dec-85    DUP   .SAV    47  20-Dec-85
TSXMOD.SAV    78  27-Nov-92    FORTRA.SAV   206  21-May-85
HARRIS.SAV    41  12-Jun-85    LET   .SAV     5  20-Dec-85
START .P36     2  21-Dec-91    RETRO .OBJ 1519P 16-May-88
EM1A  .STM    19P 02-Feb-93    TSXUCL.TSX    22  07-Mar-92
STAND .LIN    12P 15-Aug-83    TSXV6 .MSC     1P 04-Sep-95
EM1B  .STM    19  11-Feb-93    JKFLIP.SAV    30  08-Mar-96
JKFLIP.FOR     3  08-Mar-96    RT11FB.SYS    93P 20-Dec-85
ANNOT .SAV    38  18-Apr-93    TSXP23.NEW 1200P 27-Nov-92
DUO   .DIR    18  16-Jul-96    DL    .DIR     7  16-Jul-96
DIR   .DIF    26  16-Jul-96    DEMOFG.OBJ     1    -BAD-
DEMOBG.OBJ     1    -BAD-      EVAN  .ID      1
 114 Files, 5949 Blocks
 14433 Free blocks

.o
```



The "Red Room" in the Math and Computing building. In 1967 it housed an IBM 360 Model 75, the largest computer in Canada at that time.

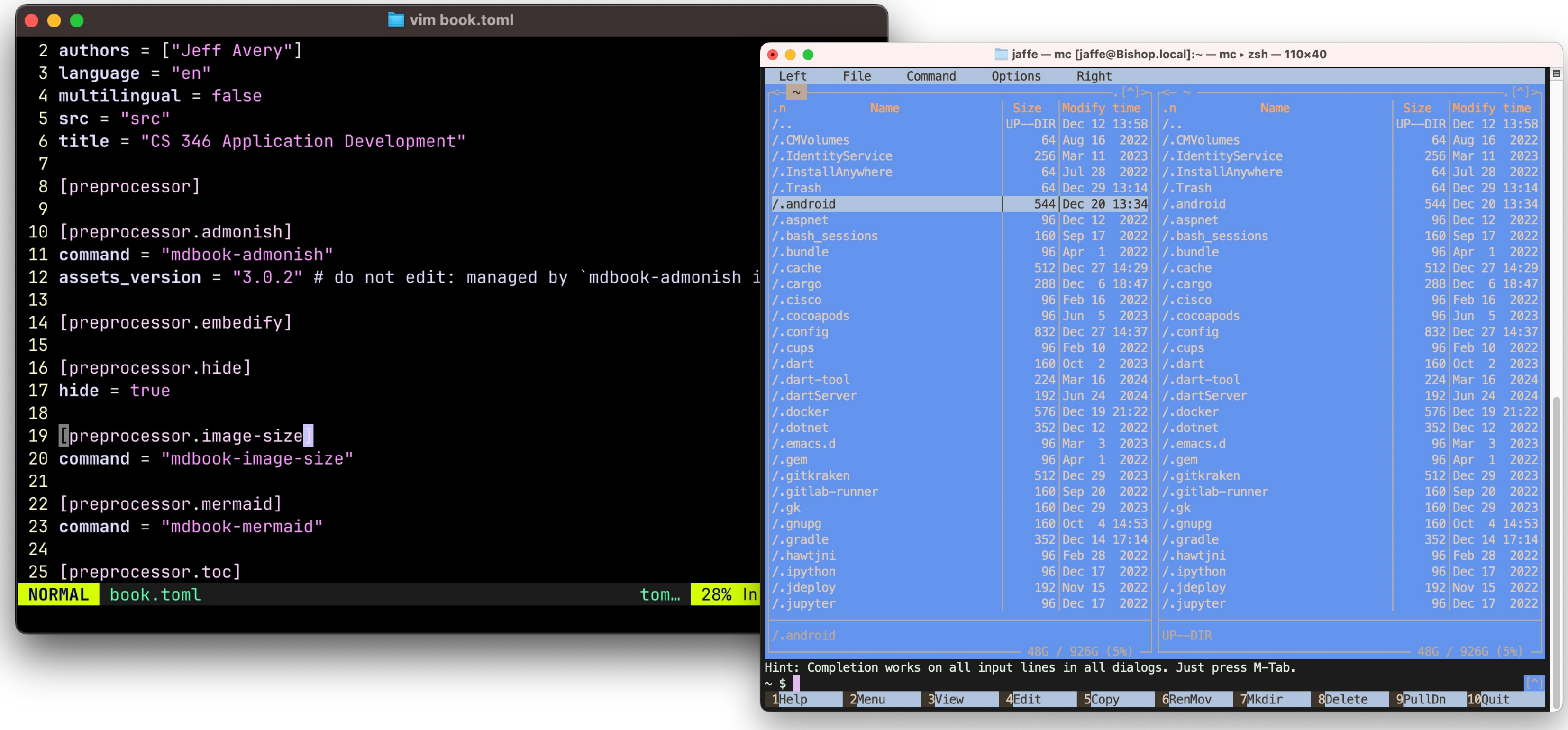
1970s: Personal computing

By the mid 1970s, small and relatively affordable computers were becoming commonplace. These personal computers were designed to be used by a small businesses that couldn't afford a more expensive system.

The CTC Datapoint 2200 terminal.

The CTC Datapoint 2200 would load simple programs into e.g., terminal protocols, but was essentially a fancy terminal.

The Radio Shack TRS-80, introduced in 1977. Targeted at hobbyists.



```
authors = ["Jeff Avery"]
language = "en"
multilingual = false
src = "src"
title = "CS 346 Application Development"

[preprocessor]

[preprocessor.admonish]
command = "mdbook-admonish"
assets_version = "3.0.2" # do not edit: managed by `mdbook-admonish i

[preprocessor.embedify]

[preprocessor.hide]
hide = true

[preprocessor.image-size]
command = "mdbook-image-size"

[preprocessor.mermaid]
command = "mdbook-mermaid"

[preprocessor.toc]
NORMAL  book.toml                                          tom…  28%  In
```

```
 Left      File      Command      Options      Right
<─ ~ ──────────────────────────────.[^]>┐┌<─ ~ ──────────────────────────────.[^]>
.n              Name          │ Size │Modify time  .n              Name          │ Size │Modify time
/..                           │UP--DIR│Dec 12 13:58 /..                           │UP--DIR│Dec 12 13:58
/.CMVolumes                   │    64│Aug 16  2022 /.CMVolumes                   │    64│Aug 16  2022
/.IdentityService             │   256│Mar 11  2023 /.IdentityService             │   256│Mar 11  2023
/.InstallAnywhere             │    64│Jul 28  2022 /.InstallAnywhere             │    64│Jul 28  2022
/.Trash                       │    64│Dec 29 13:14 /.Trash                       │    64│Dec 29 13:14
/.android                     │   544│Dec 20 13:34 /.android                     │   544│Dec 20 13:34
/.aspnet                      │    96│Dec 12  2022 /.aspnet                      │    96│Dec 12  2022
/.bash_sessions               │   160│Sep 17  2022 /.bash_sessions               │   160│Sep 17  2022
/.bundle                      │    96│Apr  1  2022 /.bundle                      │    96│Apr  1  2022
/.cache                       │   512│Dec 27 14:29 /.cache                       │   512│Dec 27 14:29
/.cargo                       │   288│Dec  6 18:47 /.cargo                       │   288│Dec  6 18:47
/.cisco                       │    96│Feb 16  2022 /.cisco                       │    96│Feb 16  2022
/.cocoapods                   │    96│Jun  5  2023 /.cocoapods                   │    96│Jun  5  2023
/.config                      │   832│Dec 27 14:37 /.config                      │   832│Dec 27 14:37
/.cups                        │    96│Feb 10  2022 /.cups                        │    96│Feb 10  2022
/.dart                        │   160│Oct  2  2023 /.dart                        │   160│Oct  2  2023
/.dart-tool                   │   224│Mar 16  2024 /.dart-tool                   │   224│Mar 16  2024
/.dartServer                  │   192│Jun 24  2024 /.dartServer                  │   192│Jun 24  2024
/.docker                      │   576│Dec 19 21:22 /.docker                      │   576│Dec 19 21:22
/.dotnet                      │   352│Dec 12  2022 /.dotnet                      │   352│Dec 12  2022
/.emacs.d                     │    96│Mar  3  2023 /.emacs.d                     │    96│Mar  3  2023
/.gem                         │    96│Apr  1  2022 /.gem                         │    96│Apr  1  2022
/.gitkraken                   │   512│Dec 29  2023 /.gitkraken                   │   512│Dec 29  2023
/.gitlab-runner               │   160│Sep 20  2022 /.gitlab-runner               │   160│Sep 20  2022
/.gk                          │   160│Dec 29  2023 /.gk                          │   160│Dec 29  2023
/.gnupg                       │   160│Oct  4 14:53 /.gnupg                       │   160│Oct  4 14:53
/.gradle                      │   352│Dec 14 17:14 /.gradle                      │   352│Dec 14 17:14
/.hawtjni                     │    96│Feb 28  2022 /.hawtjni                     │    96│Feb 28  2022
/.ipython                     │    96│Dec 17  2022 /.ipython                     │    96│Dec 17  2022
/.jdeploy                     │   192│Nov 15  2022 /.jdeploy                     │   192│Nov 15  2022
/.jupyter                     │    96│Dec 17  2022 /.jupyter                     │    96│Dec 17  2022

/.android                                          UP--DIR
                     48G / 926G (5%)                                    48G / 926G (5%)
Hint: Completion works on all input lines in all dialogs. Just press M-Tab.
~ $
 1Help   2Menu   3View   4Edit   5Copy   6RenMov   7Mkdir   8Delete   9PullDn   10Quit
```

Console applications tend to be small, quick to execute and use few resources. They are still popular, at least within specific domains e.g., system administration, software development. All modern operating systems include shells where these applications can execute.

Why does this matter?

The move towards dedicated user-based computers was significant:

- Dedicated hardware meant that we could move towards quick, interactive software i.e. type a command, get an immediate response.
- Personal computers meant that we didn't have to share systems! No more scheduling jobs/tasks, but we could instead run software on-demand.
- Performance boost of working on a local (vs remote) system.
- We were able to design better console software (running locally, limited graphics).

Development Snapshot: 1970s

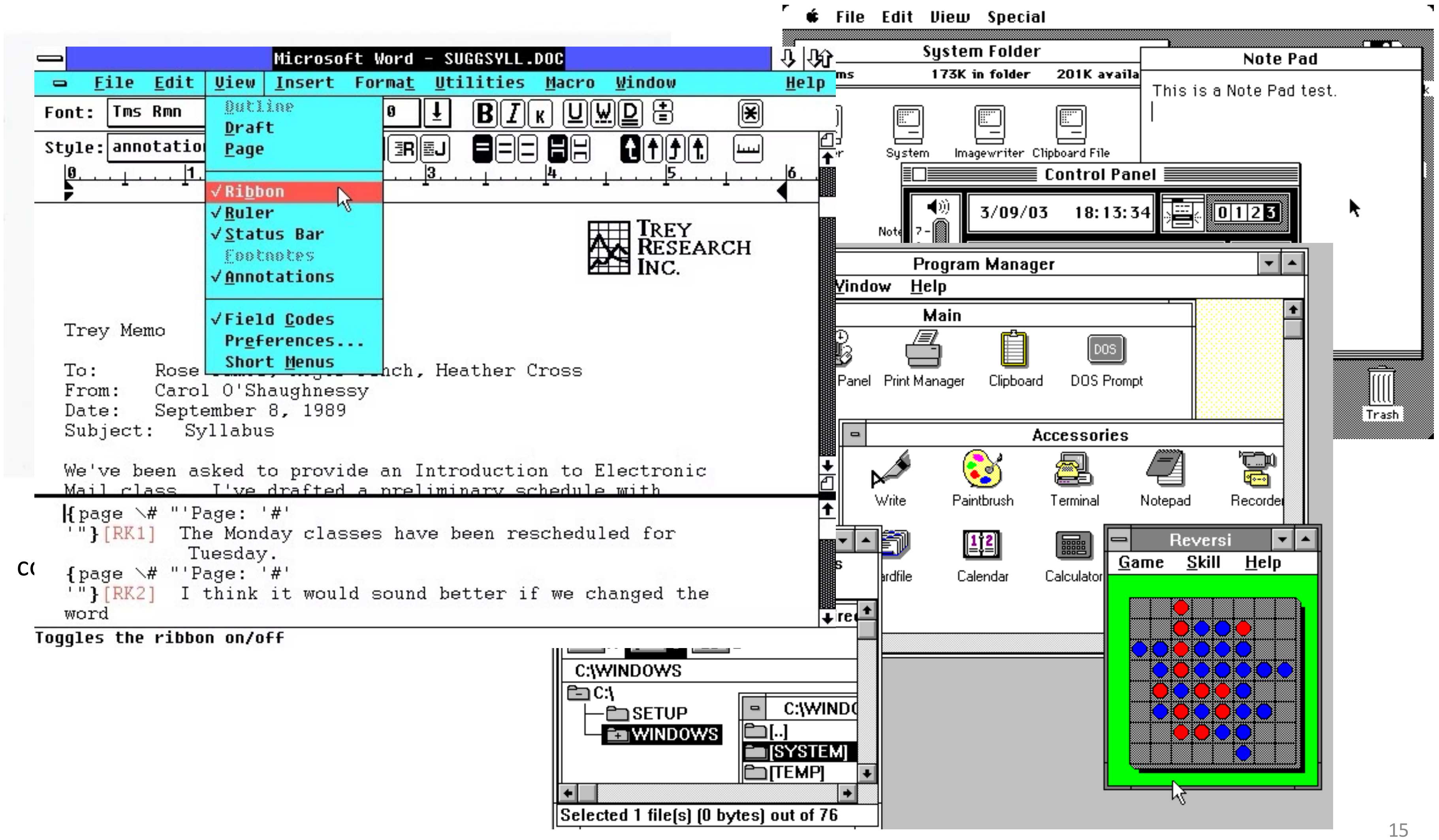
Category	Examples
Programming languages	Basic, **Assembler, C**
Programming paradigm	**Structured programming** (i.e., loops, functions)
Application style	**Console programs** (text and limited graphics)
Editor of choice	Emacs (1976), Vi (1976)
Hardware	IBM PC, terminal software, DEC PDP-10/11, VAX-11
Operating System	IBM PC DOS, MS PC DOS, VMS

"Real programmers code in assembly. C is too high an abstraction; you waste cycles."

1980s: Personal computing++

The PC movement of the 1970s exploded in the early 1980s, with the introduction of the IBM PC and DOS.

The software market also exploded in this era e.g., the invention of spreadsheets with Visicalc for the Apple II in 1979, then followed Lotus 123 for PC in 1983.

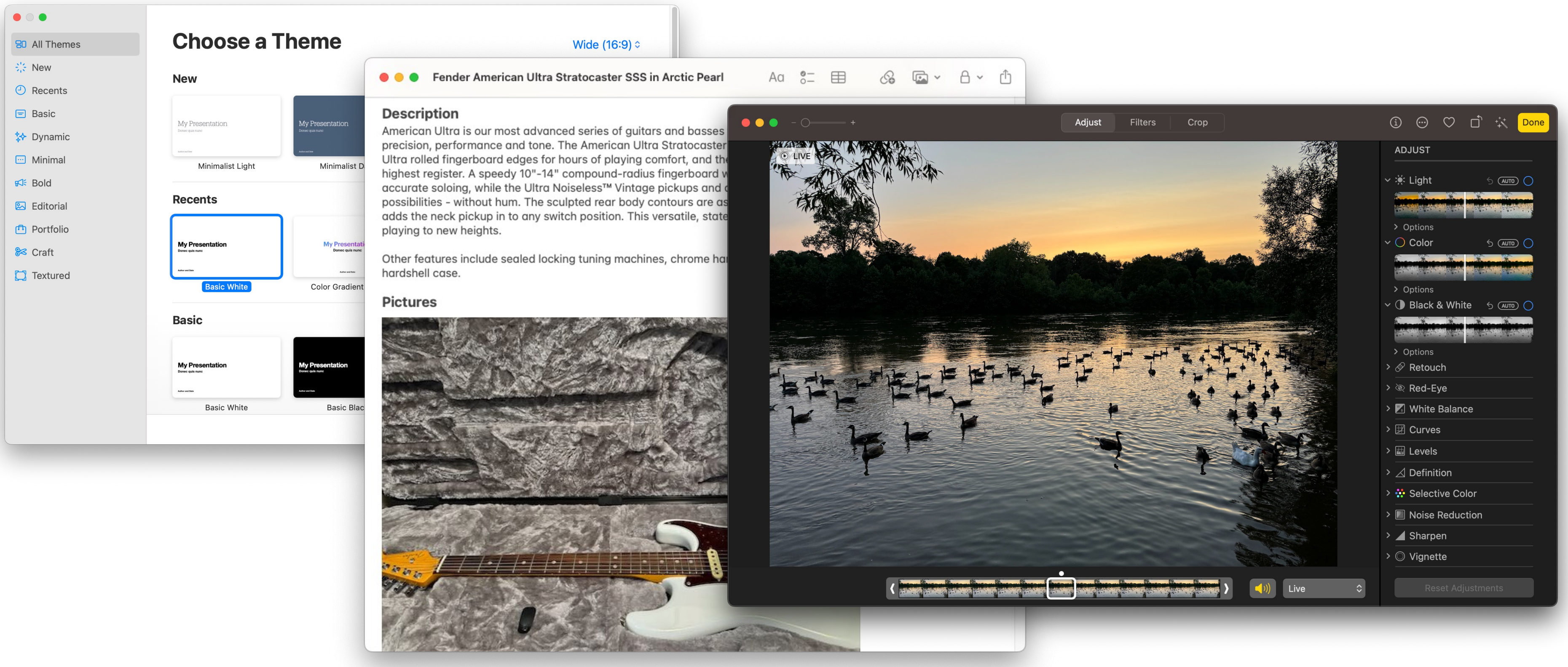


```
Trey Memo

To:      Rose ... ...nch, Heather Cross
From:    Carol O'Shaughnessy
Date:    September 8, 1989
Subject:   Syllabus

We've been asked to provide an Introduction to Electronic
Mail class.  I've drafted a preliminary schedule with

{page \# "'Page: '#'
'"}[RK1]  The Monday classes have been rescheduled for
          Tuesday.
{page \# "'Page: '#'
'"}[RK2]  I think it would sound better if we changed the
word
Toggles the ribbon on/off
```

This is a Note Pad test.

Selected 1 file(s) (0 bytes) out of 76

C(



Description

American Ultra is our most advanced series of guitars and basses
precision, performance and tone. The American Ultra Stratocaster
Ultra rolled fingerboard edges for hours of playing comfort, and th
highest register. A speedy 10"-14" compound-radius fingerboard w
accurate soloing, while the Ultra Noiseless™ Vintage pickups and c
possibilities - without hum. The sculpted rear body contours are as
adds the neck pickup in to any switch position. This versatile, state
playing to new heights.

Other features include sealed locking tuning machines, chrome har
hardshell case.

Pictures

Desktop applications are suitable for tasks that require a lot of screen real-estate, or that need to be used for long periods of time. They also support mouse/trackpad and other input devices which makes them suitable for precise input as well.

Why does this matter?

The move towards graphical systems drove some important innovations:

1. Multitasking operating systems arose to support applications running simultaneously.
2. Graphical user interfaces became the primary interaction mechanism for most users. Every major interface since the 1980s has been graphical.

In many ways, we've spent the last 40 years improving the performance of this interaction model. Everything has become "faster", which for most people, translates into more applications running (and more Chrome tabs).

Development Snapshot: 1980s

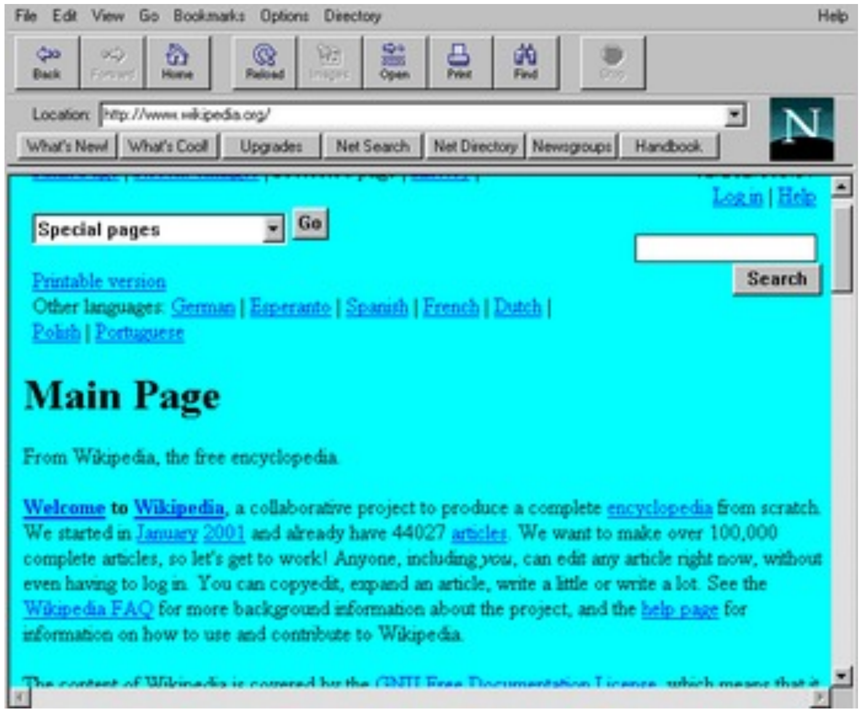
Category	Examples
Programming languages	C, **C++**, Pascal, **Objective C**
Programming paradigm	Structured & **Object-Oriented Programming**
Editor of choice	IBM Personal Editor (1982), Borland Turbo C++ IDE (1990)
Hardware	**IBM PC, Apple Macintosh**
Operating System	**MS PC DOS, MS Windows, Apple Mac OS**

“Real programmers code in C. C++ is too high an abstraction; you waste cycles.”

1990s: The Internet

Prior to the mid-1990s, computers were silos. Businesses might have their own internal networks, but there was very limited ability to connect to any external computers. Modems were used to connect home systems to some limited public networks.

The rise of the Internet in the 1990s changed this. Suddenly, it was possible to connect to any computer in the world, and to share information with anyone, anywhere.


File Edit View Go Bookmarks Options Directory Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://www.wikipedia.org/
What's New! What's Cool! Upgrades Net Search Net Directory Newsgroups Handbook
Log in | Help
Special pages
Go
Search

Printable version
Other languages: German | Esperanto | Spanish | French | Dutch |
Polish | Portuguese

Main Page

From Wikipedia, the free encyclopedia

Welcome to Wikipedia, a collaborative project to produce a complete encyclopedia from scratch. We started in January 2001 and already have 44027 articles. We want to make over 100,000 complete articles, so let's get to work! Anyone, including *you*, can edit any article right now, without even having to log in. You can copyedit, expand an article, write a little or write a lot. See the Wikipedia FAQ for more background information about the project, and the help page for information on how to use and contribute to Wikipedia.

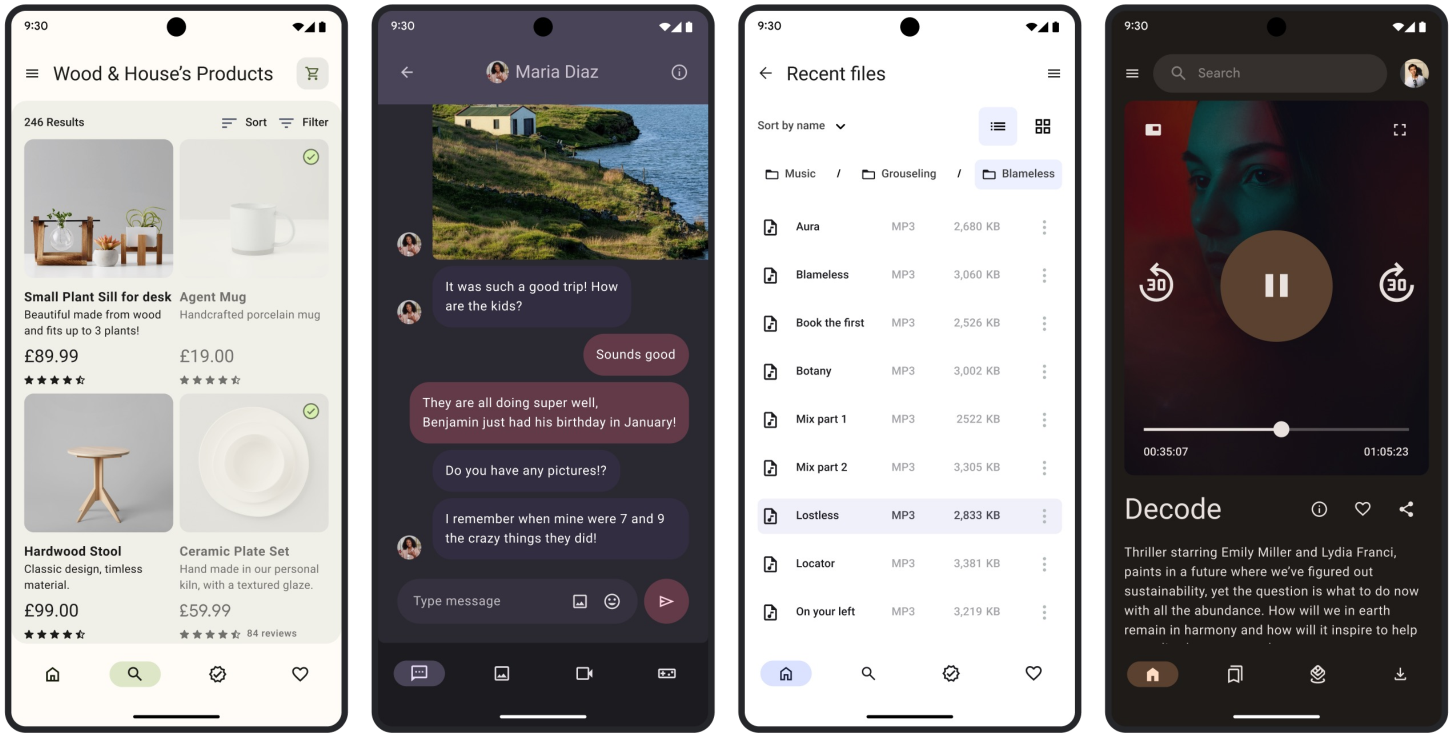
Development Snapshot: 1990s

Category	Examples
Programming languages	C++, Objective-C, Perl, Python, JS
Programming paradigm	**Object-Oriented Programming**
Editor of choice	Vim, Emacs, Visual Studio
Hardware	IBM PC, Apple Mac
Operating System	Windows, OS/2, Mac OS, (IE, Netscape), **Linux**

2000s: Smartphones

The iPhone was launched in 2007, and introduced the idea of a phone as a personal computing device. Society quickly adopted them as must-have devices.

Today, there are more smartphone users than desktop users. Most Internet traffic comes from mobile devices (99.9%, split between iPhones and Android phones).


9:30
Wood & House's Products
246 Results
Sort
Filter
Small Plant Sill for desk
Beautiful made from wood and fits up to 3 plants!
£89.99
Agent Mug
Handcrafted porcelain mug
£19.00
Hardwood Stool
Classic design, timeless material.
£99.00
Ceramic Plate Set
Hand made in our personal kiln, with a textured glaze.
£59.99
84 reviews
9:30
Maria Diaz
Type message
9:30
Recent files
Sort by name
Music / Grouseling / Blameless
9:30
Search
00:35:07
01:05:23
Decode

It was such a good trip! How are the kids?

Sounds good

They are all doing super well, Benjamin just had his birthday in January!

Do you have any pictures!?

I remember when mine were 7 and 9 the crazy things they did!

Aura	MP3	2,680 KB
Blameless	MP3	3,060 KB
Book the first	MP3	2,526 KB
Botany	MP3	3,002 KB
Mix part 1	MP3	2522 KB
Mix part 2	MP3	3,305 KB
Lostless	MP3	2,833 KB
Locator	MP3	3,381 KB
On your left	MP3	3,219 KB

Thriller starring Emily Miller and Lydia Franci, paints in a future where we've figured out sustainability, yet the question is what to do now with all the abundance. How will we in earth remain in harmony and how will it inspire to help

Mobile applications are usually designed for casual, on-the-go use. They also tend to favor content consumption vs. creation, where touch-input isn't a significant restriction. Otherwise, they are functionally very similar to desktop applications.

Development Snapshot: 2000s

Category	Examples
Programming languages	Objective-C, Swift, C++, Python, JS, Java, Kotlin
Programming paradigm	Object-oriented & functional programming
Editor of choice	Emacs, Vi, Visual Studio
Hardware	IBM PC (and comparables), iPhone, Android phone
Operating System	Windows, Mac OS, iOS, Android, (Chrome, Safari, Firefox)

Modern Applications

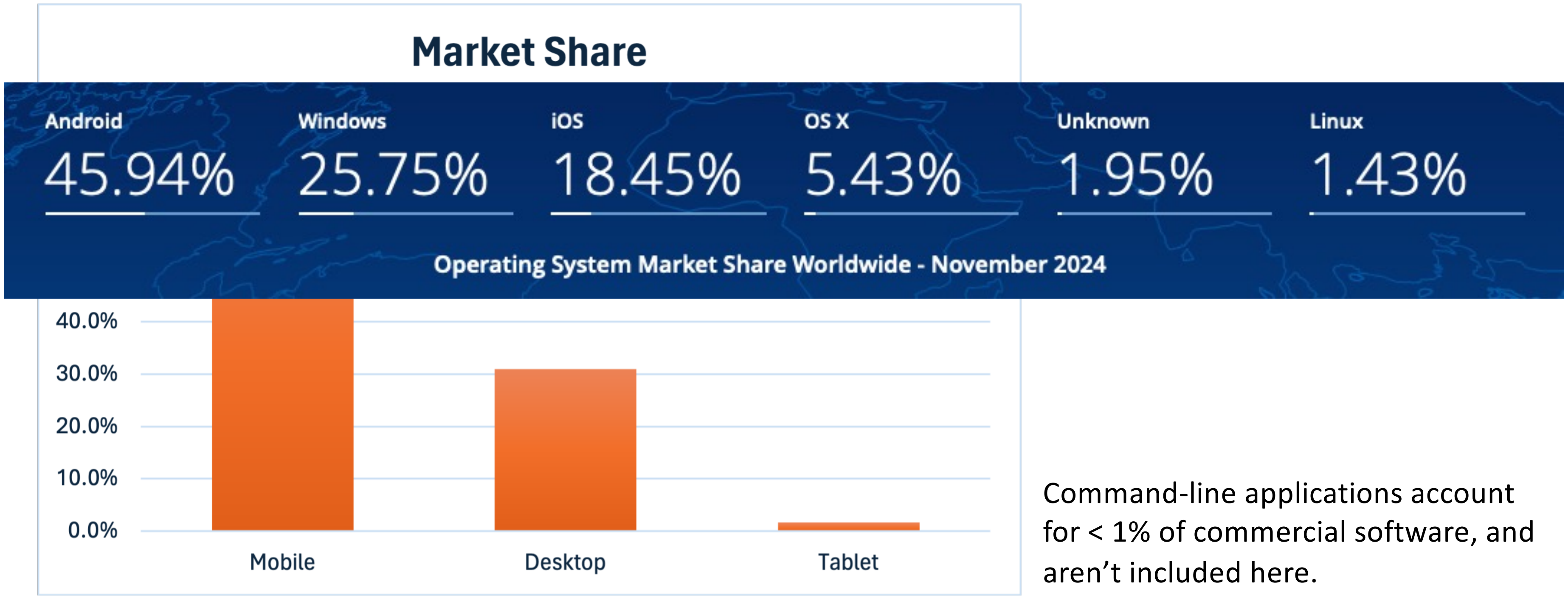
What do modern applications look like? Where do they need to run?

Application styles

All application styles that we explored are still used!

1. **Console application**: Applications that are launched from a command-line, in a shell.
2. **Desktop application**: Graphical applications, launched within a graphical desktop operating system (Windows, macOS, Linux).
3. **Mobile application**: Graphical applications, hosted on a mobile operating system (iOS, Android). Interactions optimized for small form-factor, leveraging multi-touch input.
4. **Web applications**. Launched from within a web browser. Optimized for reading short-medium block text, but capable of running apps.

Which is the most popular?

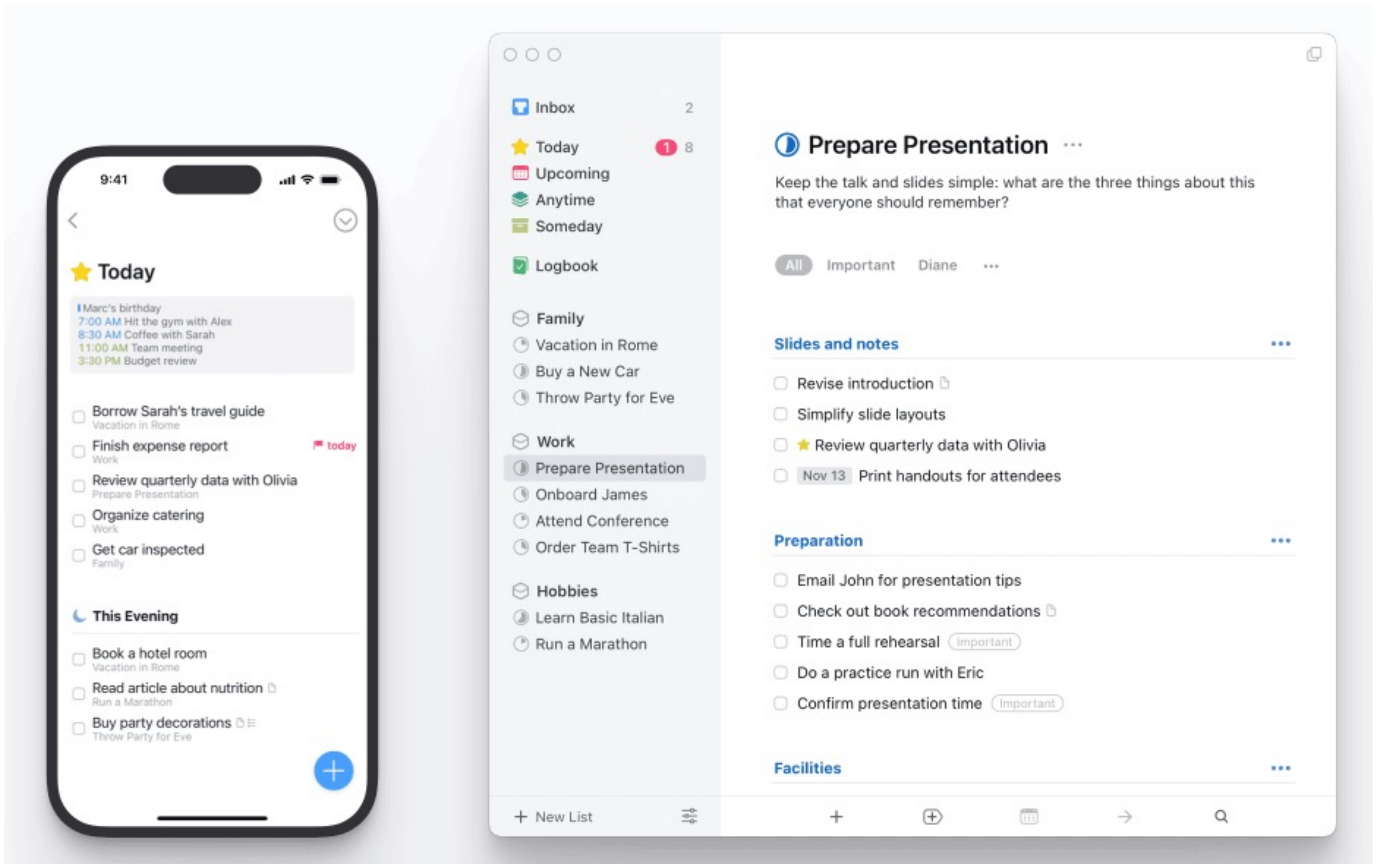


Command-line applications account for < 1% of commercial software, and aren’t included here.

What styles should we care about?

Realistically:

- **#1 Mobile applications**: Graphical applications running on a smartphone operating system (iOS, Android)
- **#2 Desktop applications**: Graphical applications, launched within a graphical desktop operating system (Windows, macOS, Linux).
- **#3. Everything else**: Kiosks, In-car “infotainment”, Watch etc. based on your specific application.



Today

Marc's birthday
7:00 AM Hit the gym with Alex
8:30 AM Coffee with Sarah
11:00 AM Team meeting
3:30 PM Budget review

- Borrow Sarah's travel guide — Vacation in Rome
- Finish expense report — Work — today
- Review quarterly data with Olivia — Prepare Presentation
- Organize catering — Work
- Get car inspected — Family

This Evening

- Book a hotel room — Vacation in Rome
- Read article about nutrition — Run a Marathon
- Buy party decorations — Throw Party for Eve

Prepare Presentation

Keep the talk and slides simple: what are the three things about this that everyone should remember?

Slides and notes

- Revise introduction
- Simplify slide layouts
- Review quarterly data with Olivia
- Nov 13 Print handouts for attendees

Preparation

- Email John for presentation tips
- Check out book recommendations
- Time a full rehearsal — Important
- Do a practice run with Eric
- Confirm presentation time — Important

Facilities

Things 4 running on iOS and Desktop. A popular decision of "which platform to support" is "all of them!".

Technology stack

What technologies are suitable for building applications?

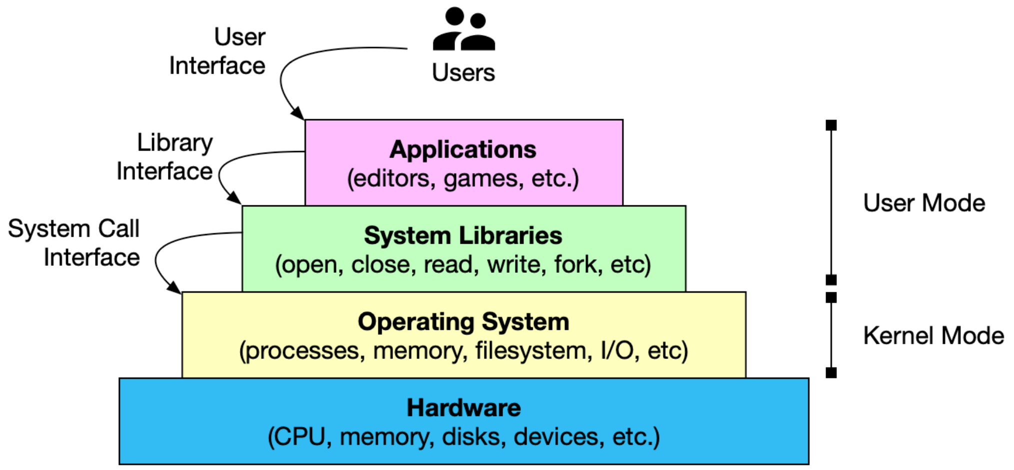
What is a technology stack?

Technology stack == all the technologies that work together to deliver the functionality that you need as a software developer.

Typically includes:

- Operating system + system libraries
- Programming languages + other tools
- Any other libraries that provide specialized services

The role of the operating system



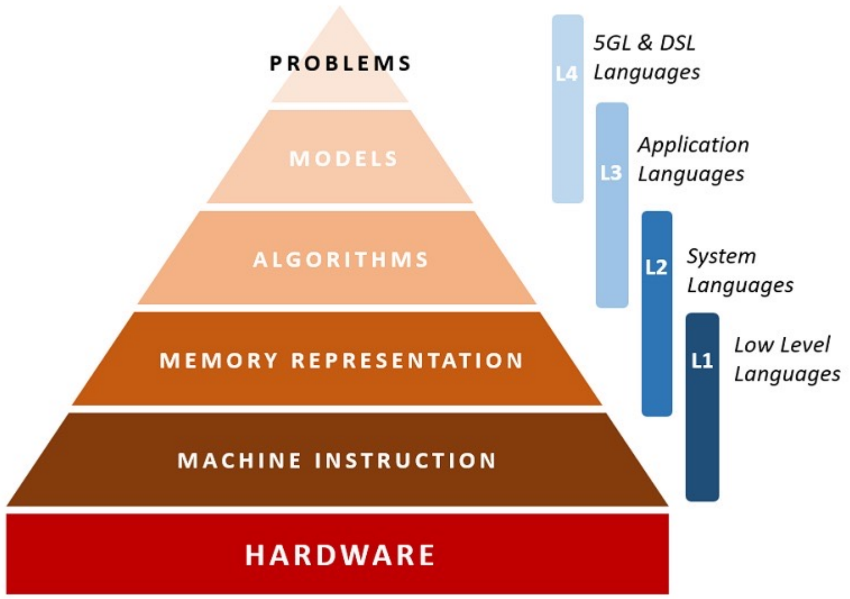
- The operating system sits between the hardware and the application.
- OS system libraries expose complex functionality to your application.
- Other frameworks e.g., graphics can be included at the library level.
- Compilers need to be able to access and import libraries.

Programming languages

Your choice of programming language needs to align with the software that you are developing.

We can (simplistically) divide programming languages into two categories: **low-level** and **high-level** languages.

Early assumptions in programming languages is that we were move away from low-level languages, and this has *sort-of* happened…



Low-level languages are suitable when you are concerned with the performance of your software, and when you need to control resources.

- They are often used for **systems programming** i.e., delivering code that runs as fast as possible and uses as little memory as possible.
- Examples of systems languages include C, C++, and Rust.
- Appropriate domains include device drivers, and game engines.

High-level languages are suitable when you are concerned with the speed of development, extensibility, and the robustness of your solution.

- **Applications programming** leans heavily on high-level languages, trading some performance for desirable programming language features.
- Examples of application languages include Swift, Kotlin, Go, and Dart.
- Appropriate domains include web, mobile/desktop applications.

What does a high-level language provide?

Automatic memory management

Automatic memory allocation/deallocation (via ref-counting, or GC). This eliminates the risk of accessing uninitialized memory.

Type inference

Type inference across the type system.

NULL safety

A type system that prevents NULL errors.

Concurrency

More control over how async code executes.

Broader programming models

Mix of functional, object-oriented paradigms.

Features	**Low-level**	**High-level**
Memory management	manual	garbage collection
Type system	static	static or dynamic
Runtime	fast	medium
Executables	small, fast	large, slow
Portability	low	high

These features contribute to our goal of building robust, extensible, reusable software.

Libraries

You could, in theory, access a lot of your operating system functionality through system calls, but this would be highly platform-specific code, and your app would not be portable across operating systems. We often use libraries, which tend to abstract away small OS and hardware differences.

Library origin		
Programming language	stdlib, stdio	Libraries included in the programming language; guaranteed to exist everywhere
OS vendor	Win32, Cocoa	Libraries/Frameworks provided by the vendor to support syscalls/low-level access. Includes common functionality like graphics, networking.
Third-party	OpenGL, OpenCV	Libaries provided by other parties to allow additional support beyond what the OS vendors provide. e.g., database access, computer vision.

OS + Language + Libraries

Here’s the set of common technology stacks that are used, based on platform and style of application. Note that this is not complete but reflects common technologies.

Platform	OS	Programming Languages	Libraries?
Desktop	Windows (Microsoft)	C#	.NET, UWP (Maui)
	macOS (Apple)	Swift, Objective-C	Cocoa, UIKit, SwiftUI
	Linux (Many)	C, C++	GTK
Mobile	Android (Google)	Kotlin	Android SDK, Compose
	iOS (Apple)	Swift, Objective-C	UIKit, SwiftUI

^ Languages & Libraries are often vendor-specific !

Why not build web applications?

It’s possible to build *practically anything* as a web application.

There are many *fantastic* applications written for the web e.g., Gmail, Netflix, D&D Beyond.

Why not build web apps? Native applications are more capable!

- Native applications tend to be faster.
- Native applications can do more. We aren’t restricted by a browser’s security model.
- Native applications can work offline/handle network issues.
- We can customize interaction. e.g., multiple windows open; keyboard shortcuts.

Examples of native/web applications

Native technologies

- IntelliJ IDEA IDEA, Kotlin, C++ compilers, Python interpreter.
- MS PowerPoint, MS Word, MS Excel.
- Apple Notes, Reminders.
- Forklift file browser.

Web technologies

- Banking application
- Healthcare portal
- VS Code, Discord, Slack (”disguised” as native apps)

What **are** *we* doing?

- We’re going to build native applications using the Kotlin toolchain:
  - **Native support for Android**. If you want to build a mobile application, this is the main (native) language for the most popular mobile OS. Win win.
  - **Cross-platform support for Desktop**. You can also build reasonably good desktop applications for Windows, Linux and macOS with Kotlin/JVM. Libraries aren’t as well developed as Kotlin/Android but certainly “good enough” for this course and for *most* applications you will build.
- Kotlin is one of many “modern” application languages (along with Swift, Go, Dart), with features that make it extremely useful for building applications.
- We’ll discuss Kotlin in detail and introduce **supporting libraries** to extend its functionality i.e., graphics, user interfaces, databases and so on.