

Portability

Application binary interfaces (ABI)

- most Windows laptops, Linux laptops and pre-M1 Macs share the same ISA: [x86_64](#)
- iPhones, Android phones, M1 to M4 Macs share the same ISA: [AArch64](#)

Q: Why, then, do applications need to be recompiled separately for each platform?
e.g. iPhone vs. Android phone

A: Because platforms have different OSs and ABIs.

What is an ABI?

An application binary interfaces (ABI) defines:

- file format for
 - object files
 - dynamically-linked files (shared objects / dll)
 - and executable files
- convention for function calls
- convention for system calls

It is called **binary** because it is related to how the machine code is organised (as opposed to the source code in a programming language)

ABI: function calls (x86_64)

```
#include <stdio.h>

int main()
{
    puts("Hello\n");
    return 0;
}
```

clang / Linux / x86_64

```
main:
    push    rax
    lea    rdi, [rip + .L.str]
    call   puts@PLT
    xor    eax, eax
    pop    rcx
    ret

.L.str:
    .asciz "Hello\n"
```

MSVC / Windows / x86_64

```
_DATA SEGMENT
$SG9391 DB      'Hello', 0aH, 00H
_DATA  ENDS

main   PROC
$LN3:
    sub    rsp, 40
    lea   rcx, OFFSET FLAT:$SG9391
    call  puts
    xor   eax, eax
    add  rsp, 40
    ret  0
main   ENDP
```

ABI: function calls (AArch64)

```
#include <stdio.h>

int main()
{
    puts("Hello\n");
    return 0;
}
```

clang / MacOS / [AArch64](#)

```
main:
    stp    x29, x30, [sp, #-16]!
    mov    x29, sp
    adrp   x0, .L.str
    add    x0, x0, :lo12:.L.str
    bl    puts
    mov    w0, wzr
    ldp    x29, x30, [sp], #16
    ret

.L.str:
    .asciz "Hello\n"
```

MSVC / Windows / [AArch64](#)

```
        IMPORT    |puts|

|main|  PROC
|$LN3|

    stp    fp,lr, [sp, #-0x10]!
    mov    fp, sp
    adrp   x8, |$SG4901|
    add    x0, x8, |$SG4901|
    bl    puts
    mov    w0, #0
    ldp    fp,lr, [sp], #0x10
    ret

        ENDP
```

Try it for yourself:
godbolt.org

Calling convention

```
int function(int a1, int a2, int a3, int a4, int a5, int a6, int a7)
{
    return a1;
}
```

platform	a1	a2	a3	a4	a5	a6	a7	a8	...	return value
AArch64	x0	x1	x2	x3	x4	x5	x6	x7	(stack)	x0
“SysV” x86_64	rdi	rsi	rdx	rcx	r8d	r9d	(stack)	(stack)	(stack)	rax
Windows x86_64	rcx	rdx	r8d	r9d	(stack)	(stack)	(stack)	(stack)	(stack)	rax

Note: floating-point parameters are passed separately

Some specifications

SysV [x86_64](#) ABI: [repo](#), [pdf](#)

[AArch64](#) ABI: [repo](#)

Linux-specific stuff: [documents](#)

Remarks:

- OS vendors may or may not adhere to the ABI spec of the hardware:
 - Microsoft Windows does their own thing on [x86_64](#)
 - MacOS follows [AArch64](#) calling convention,
but uses Mach-O (not ELF) as an object file format
- Some part of the ABI may be defined by OS vendors
(e.g. system call convention)
- The ABI is language-independent,
but the C language (sometimes C++ as well) has a special status
The ABI is defined **in terms of** C function calls and C datastructures.

Portable code

How do we ship code that work across all platforms?

Option 1: interpreters

- use interpreted languages, ship source
 - Python, Javascript, ...
- languages that compile to virtual machine code
 - ship VM code
 - optionally, ship VM interpreter
 - Java, C#

Option 2: multiple compilations

- compile one executable on each platform
- in some cases, cross-compilation is possible
 - MacOS → iOS
 - Linux → Android





What if we cannot (or do not want to) recompile?

Option 3: Translation

Use case: same OS, different ISA

- Translation is a form of compilation
- From machine code
- To machine code (of a different ISA)

Example: Apple Rosetta 2 translates `x86_64` into `AArch64`

Activity Monitor								
All Processes								
ⓧ ⓘ ⋮ CPU Memory Energy Disk Network 🔍 Search								
Process Name ^	% CPU	CPU Time	Threads	Idle Wake Ups	Architecture	% GPU	GPU Time	PID
accountsd	0.0	2:04.61	3	0	Apple	0.0	0.00	876
 Activity Mo...	5.9	34.51	5	2	Apple	0.0	0.00	95250
adb	0.2	56.96	5	1	Intel	0.0	0.00	27487
 AdGuard Ad...	0.0	1.36	4	0	Intel	0.0	0.00	45039
 AdGuard Sa...	0.0	1.21	4	0	Intel	0.0	0.00	45040
AdGuardFor...	0.0	0.08	5	0	Intel	0.0	0.00	45045
adid	0.0	4.20	2	0	Apple	0.0	0.00	902
adprivacyd	0.0	1.97	2	0	Apple	0.0	0.00	92215
 AirPlayUIAg...	0.0	1.04	3	0	Apple	0.0	0.00	7740
AirPlayXPC...	0.0	39.62	5	1	Apple	0.0	0.00	534
airportd	0.0	5:32.57	11	0	Apple	0.0	0.00	596
akd	0.0	0.51	2	0	Apple	0.0	0.00	31868
akd	0.0	7.36	3	1	Apple	0.0	0.00	901
AlertNotific...	0.0	0.01	2	0	Apple	0.0	0.00	60628
amfid	0.0	7.40	2	0	Apple	0.0	0.00	535
AMPDevice...	0.0	0.53	4	0	Apple	0.0	0.00	7571
AMPLibrary...	0.0	9.33	4	0	Apple	0.0	0.00	10224
amsaccount...	0.0	8.01	4	0	Apple	0.0	0.00	7745

Option 4: Compatibility layers

Use case: different OSs, same ISA

- add OS support for a foreign ABI
 - foreign file formats (for objects, DLLs and executables)
 - foreign convention for system calls
- add libraries for foreign ABI
 - foreign convention for function calls

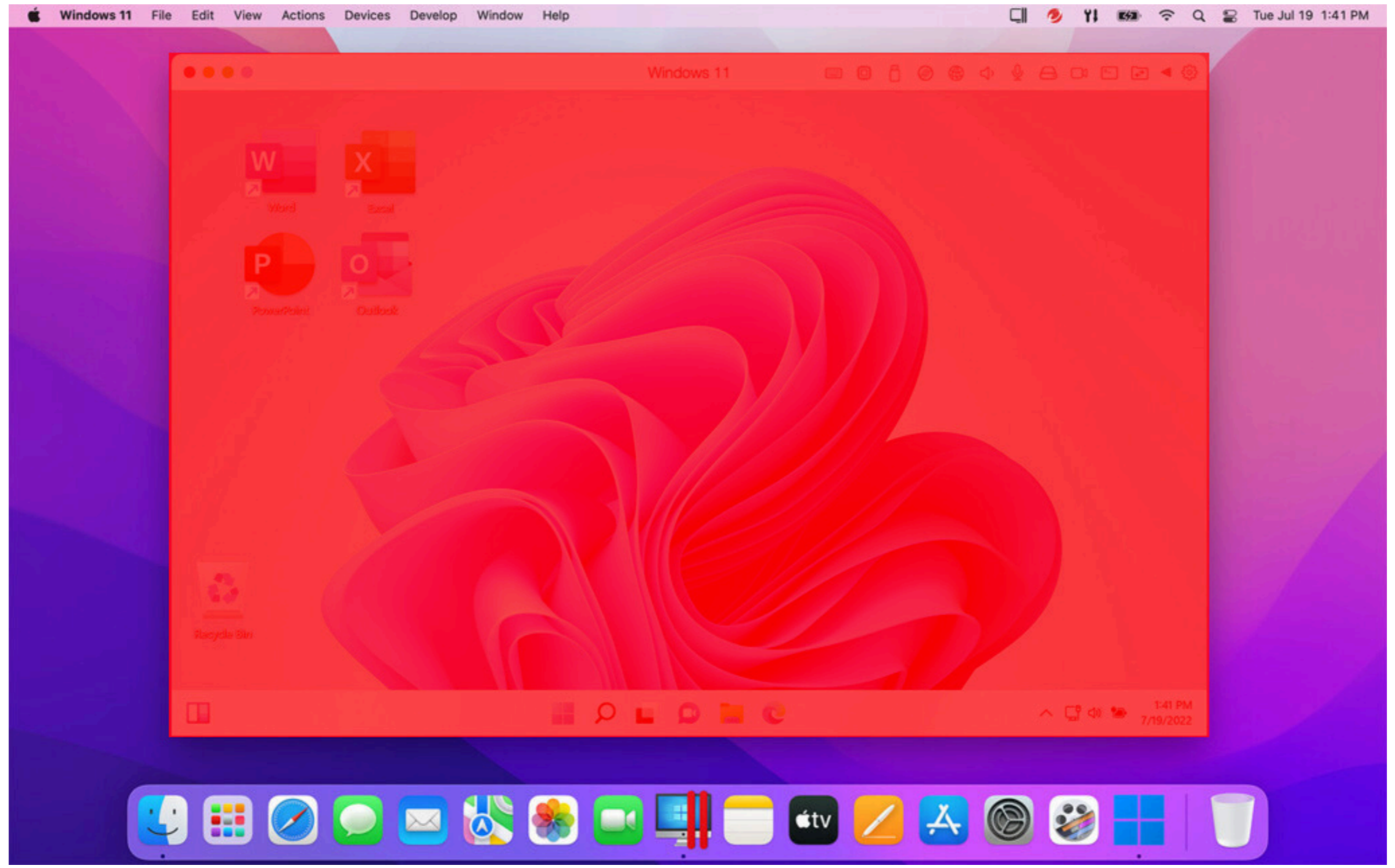
Examples:

- Wine allows running Windows apps on Linux.
- WSLv1 allows running Linux apps on Windows.

Option 5: emulation

- an emulator is an **interpreter** for machine code (e.g. QEmu)
- much slower than running the code
- JIT can mitigate slowness, to some extent
- typically, a full-blown **operating system runs inside** the **interpreter**!



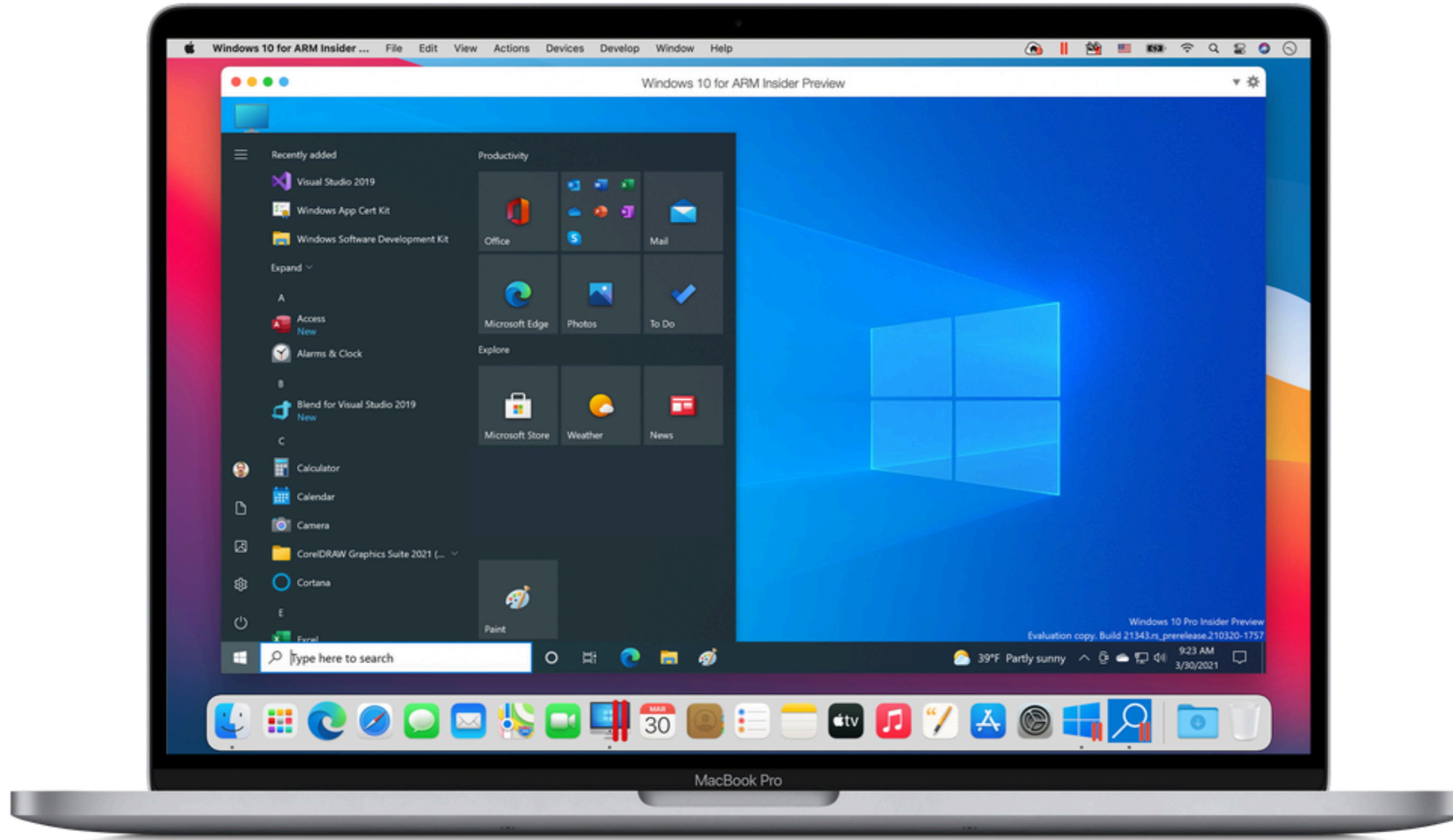




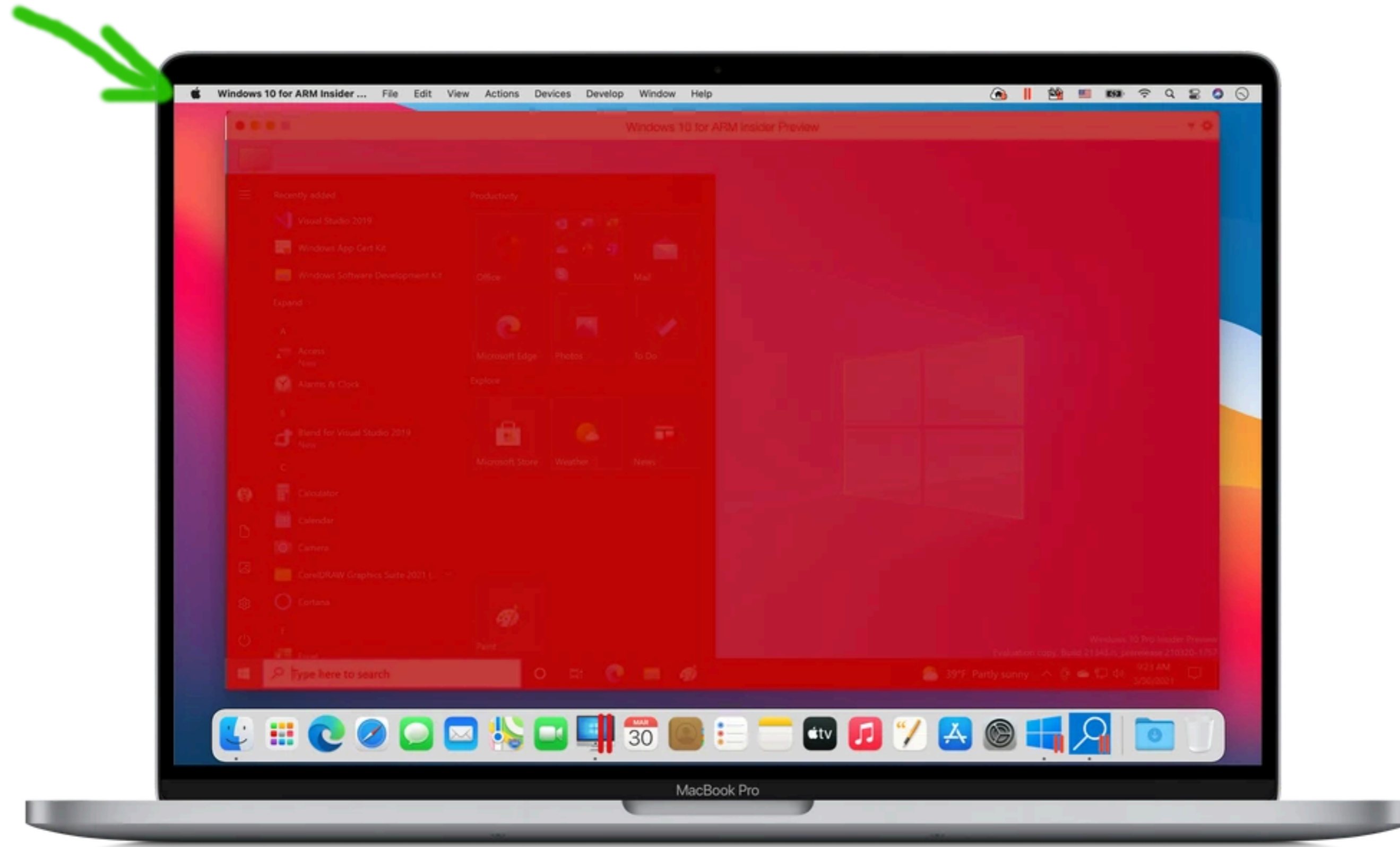
Option 6: virtualization

- virtualization is essentially hardware-assisted emulation
(e.g. Xen, KVM, VirtualBox, VMWare, Apple Parallels, WSLv2)
- virtualized software must target the **same ISA** as hardware
- like emulation, runs a full-blown **operating system**

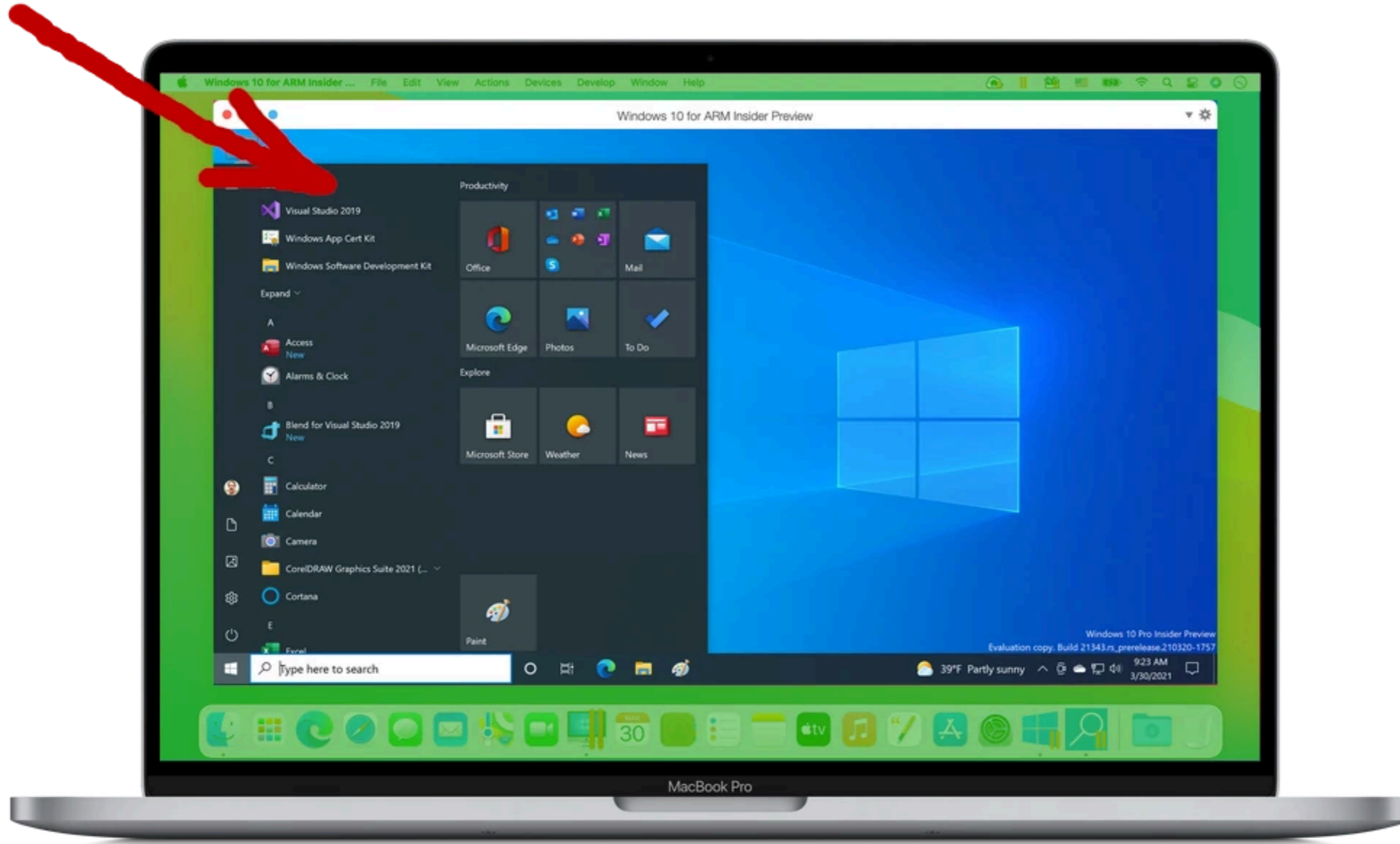
Example: Apple Parallels



Host OS



Guest OS



Definitions

- The *hypervisor* is the software that manages the **guest OS**.
- It can be the **host OS** itself (“Type 1”: Xen, KVM)
- It can be a process within the **host OS** (“Type 2”: Apple Parallels)

Virtualization mainly deals with security:

Let **guest OSs** believe they have direct access to hardware...

... but every hardware access is tightly controlled by the **hypervisor**

Virtualization is the main technology enabling “cloud computing”.

- Amazon Web Services runs **Xen**
- Google Cloud Platform runs **KVM**
- Customers rent a virtual machine in a datacenter
 - They can connect (remotely) to this machine
 - It runs their (**guest**) OS of choice
 - It acts as if it was physical hardware

Option 7: containers

Use case: Same ISA, same kernel, different OS.

- Containers are a lightweight form of virtualization.
- The **host**'s kernel also acts as a kernel for the **guest**.
- Mainly: filesystems, libraries and applications are separated.

Examples:

- A Debian Linux **guest** on a Fedora Linux **host**
- A Debian 11 Linux **guest** on a Debian 12 **host**
- A Debian 12 **guest** with specific libraries installed, on a Debian 12 **host**

Application programming interfaces (API)

Definition

An API defines how a library (or any other service) is to be used.

Library API

```
FILE *fopen(const char *path, const char *mode);
```

```
open(file, mode='r', buffering=- 1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
```

Web API

```
GET https://www.google.com/search?q=<query>
```

Example:

```
google-chrome https://www.google.com/search?q=Software%20Engineering
```

```
GET https://cloudflare.com/cdn-cgi/trace
```

Example:

```
curl -4s "https://cloudflare.com/cdn-cgi/trace"
```

```
PUT https://api.cloudflare.com/client/v4/zones/{zone_idenfier}/dns_records/{identfier}
```

Example:

```
curl --request PUT \  
  --url https://api.cloudflare.com/client/v4/zones/zone_idenfier/dns_records/identfier \  
  --header 'Content-Type: application/json' \  
  --header 'X-Auth-Email: ' \  
  --data '{  
    "content": "198.51.100.4",  
    "name": "example.com",  
    "proxied": false,  
    "type": "A",  
    "comment": "Domain verification record",  
    "tags": [  
      "owner:dns-team"  
    ],  
    "ttl": 3600  
  }'
```

APIs and portability

- many APIs are cross-platform
 - C standard library
 - Almost all Python modules
 - Qt, Electron, Flutter, ... (frameworks for GUI applications)
 - WEB APIs only depend on an internet connection
- some are specific to a platform
 - Windows UI Library, MacOS Cocoa

Dependencies

- your code requires libA version ≥ 1.1 , lib B version ≥ 4.5
 - lib B version 4.5 requires libX version 2.0 and libA version 0.8
 - lib B version 4.7 requires libX version 2.0 and libA version 1.1
 - lib B version 4.6 requires libX version 2.0 and libA version 2.0
 - lib X version 2.0 requires libA version ≤ 1.9

How do we install all this?

Which version do we install?

Package managers

Package managers solve this problem for you.

They can solve it...

- at the OS level:
 - MacOS: `brew install <package>`
 - Debian/Ubuntu Linux: `apt-get install <package>`
 - Fedora/Suse Linux: `dnf install <package>`
- at the language level:
 - Python: `pip install <module>`
 - JavaScript/Node: `npm install <package>`
 - Rust: `cargo install <crate>`

Limitations

- package selection may be limited (packaging is labor-intensive)
- security and trust