Noah

Hypervisor-Based Darwin Subsystem for Linux

Takaya Saeki, Yuichi Nishiwaki

Self Introduction

Noah Developer Team

Takaya Saeki

Yuichi Nishiwaki

They both are graduate students at the University of Tokyo. They are developing Noah in their free time.

Noah was selected as one of MITOH projects, which is a financial assistance program by the government of Japan for outstanding young programmers

Noah

- A middleware that runs unmodified Linux ELF applications on macOS
- Reduce cost of waiting / creating macOS port of Linux apps
- Accomplish it by special hypervisor. Load ELF binary into VM and let it run instead of kernel, trap system calls from it by hypervisor, and translate them to corresponding system calls on macOS.

Noah

- A middleware that runs unmodified Linux ELF applications on macOS
- Reduce cost of waiting / creating macOS port of Linux apps
- Accomplish it by special hypervisor. Load ELF binary into VM and let it run instead of kernel, trap system calls from it by hypervisor, and translate them to corresponding system calls on macOS.

We discuss the architecture in detail later!

Short Demo;

What it looks like

Agenda

- What is Noah (Done!)
- Background
- Noah in Detail
    - Architecture Overview
    - Advantages of Noah Architecture
    - Subsystem Implementation
        - Memory management, VFS, and the other
- Current Implementation Status and Performance
- Related Technologies and Comparison
  (Windows Subsystem for Linux, Linuxulator, and so on)
- Their Possible Impact on Cross Platform Development

Agenda

- What is Noah (Done!)
- **Background**
- Noah in Detail
    - Architecture Overview
    - Advantages of Noah Architecture
    - Subsystem Implementation
        - Memory management, VFS, and the other
- Current Implementation Status and Performance
- Related Technologies and Comparison
  (Windows Subsystem for Linux, Linuxulator, and so on)
- Their Possible Impact on Cross Platform Development

Background

Linux

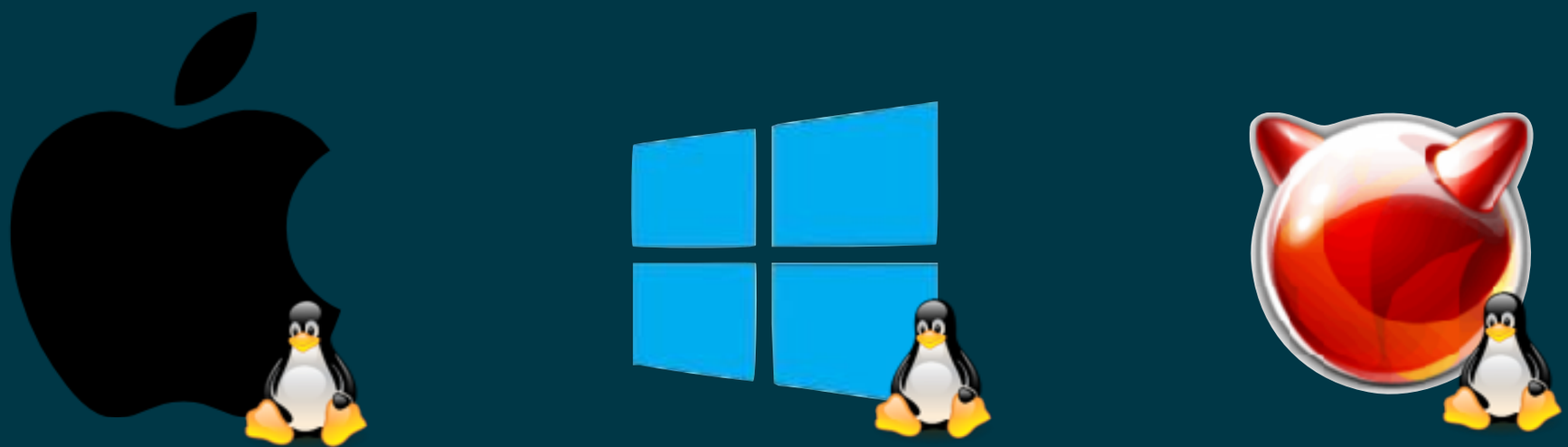
- One of the most important operating systems today
- Most popular OS for WEB servers
- Many apps and middleware come from the Linux ecosystem, and they are ported to other operating systems for developers

Problem: Porting cost

Porting cost

- Many Linux applications are ported to macOS, FreeBSD, and Windows…
- But it takes time and effort!
- Windows decided to have Linux compatibility layer called “Windows Subsystem for Linux” in 2016 to benefit from the Linux ecosystem directly
- FreeBSD also already has Linux compatibility layer

- macOS does not have Linux compatibility layer yet despite its large number of developers
- Noah fills the missing piece!



If major operating systems have Linux compatibility, developers don’t have to port Linux applications nor wait for them to be ported

Architecture Overview of Noah

Architecture Overview

Noah’s architecture consists of three components

1. VT-x Virtual Machines

   They have NO kernel inside them, but directly boot an ELF binary and let it run instead.

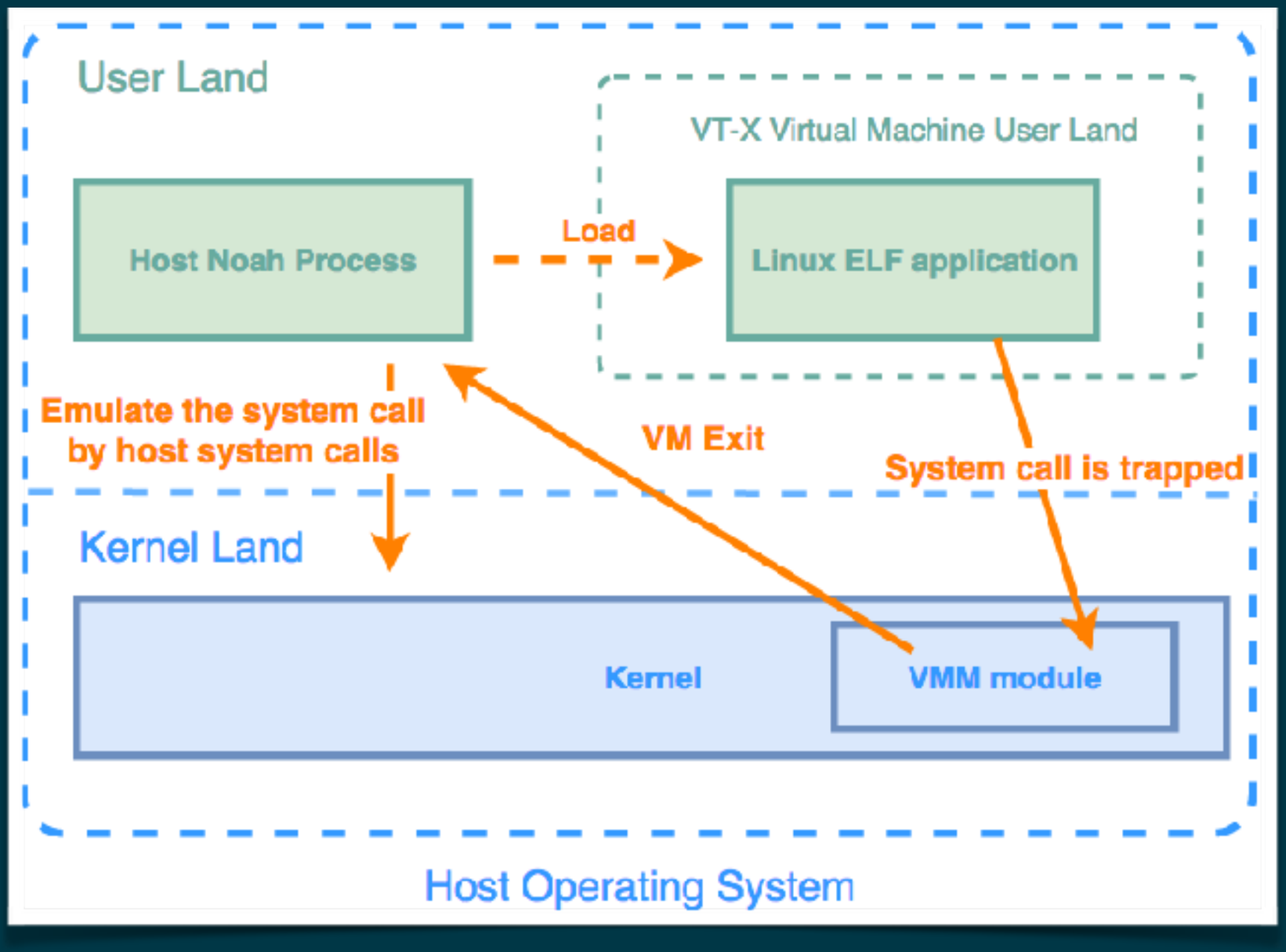
2. Host Noah Processes

   Processes that run on the host OS, which actually work as Linux compatibility layer

3. Virtual Machine Monitor module (VMM).

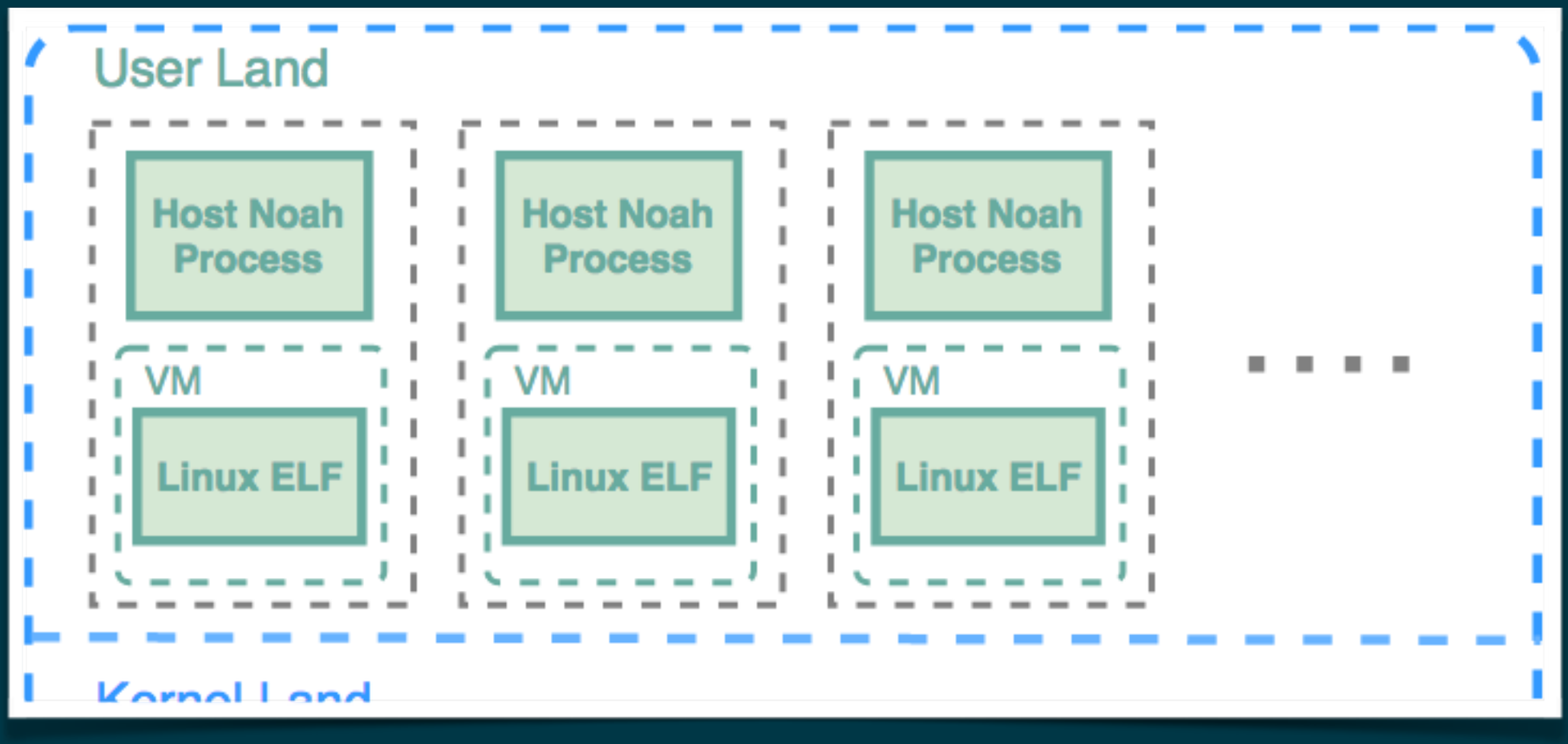
   Actually, not a part of Noah itself, but a kernel API of the host OS for virtualization.
   Apple Hypervisor Framework in macOS, KVM in Linux, for example.

Architecture Overview



1. Host Noah process launches a new VM and loads ELF inside it by ELF loader implemented in the host Noah process
2. The VM runs ELF in its virtualized userland
3. The ELF application calls Linux system calls when running
4. VMM module traps the system call and passes it to the host Noah process
5. Host Noah process emulates the behavior of Linux system call by host OS's system calls

Architecture Overview



A pair of a host Noah process and a VM corresponds to a Linux application. So, when there are multiple Linux applications, there are also multiple pairs of a host Noah process and a VM.

Example1：How “Hello, world” works

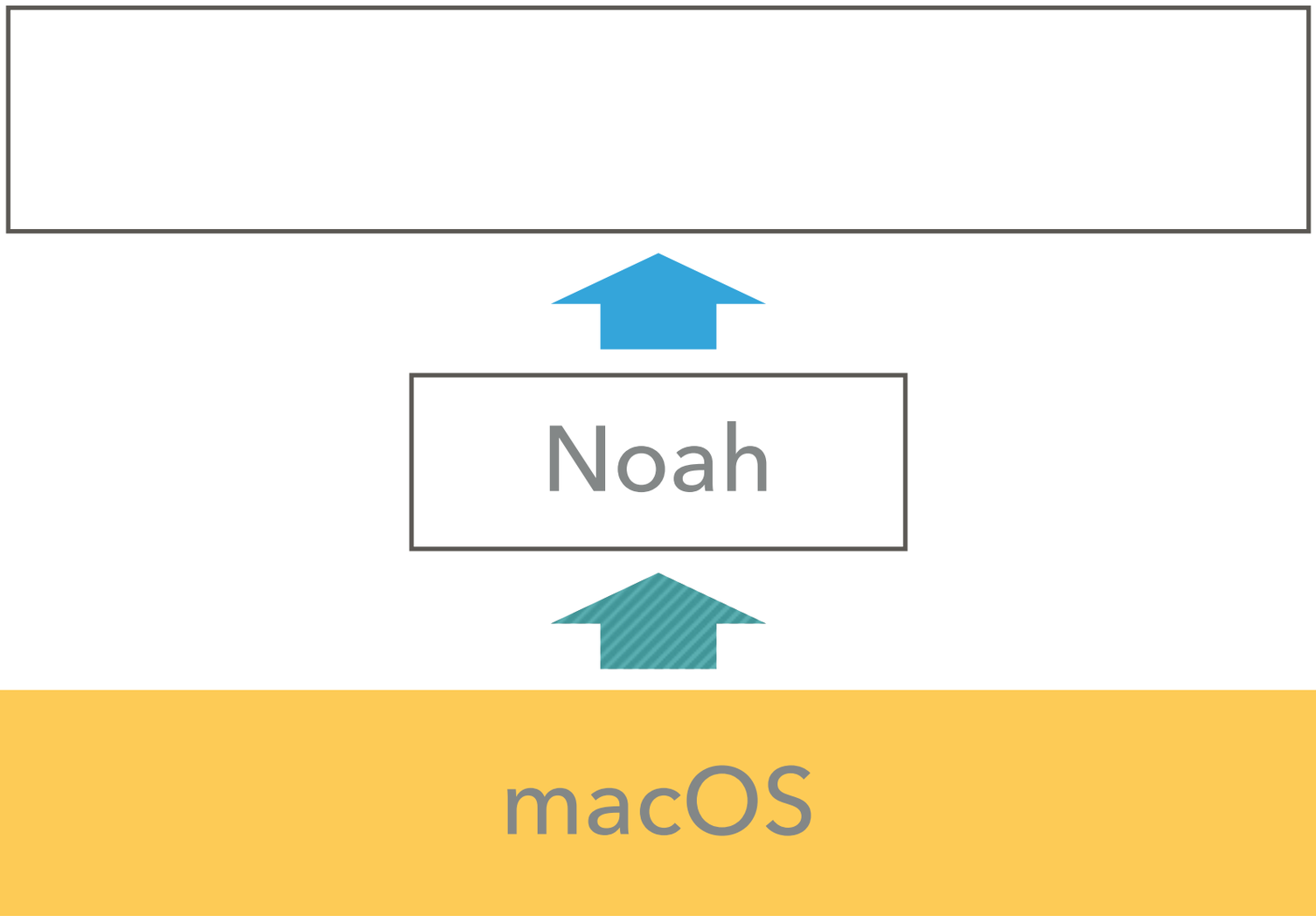
```
$ noah /bin/hello
```

Example1 : How “Hello, world” works

```
$ noah /bin/hello
```

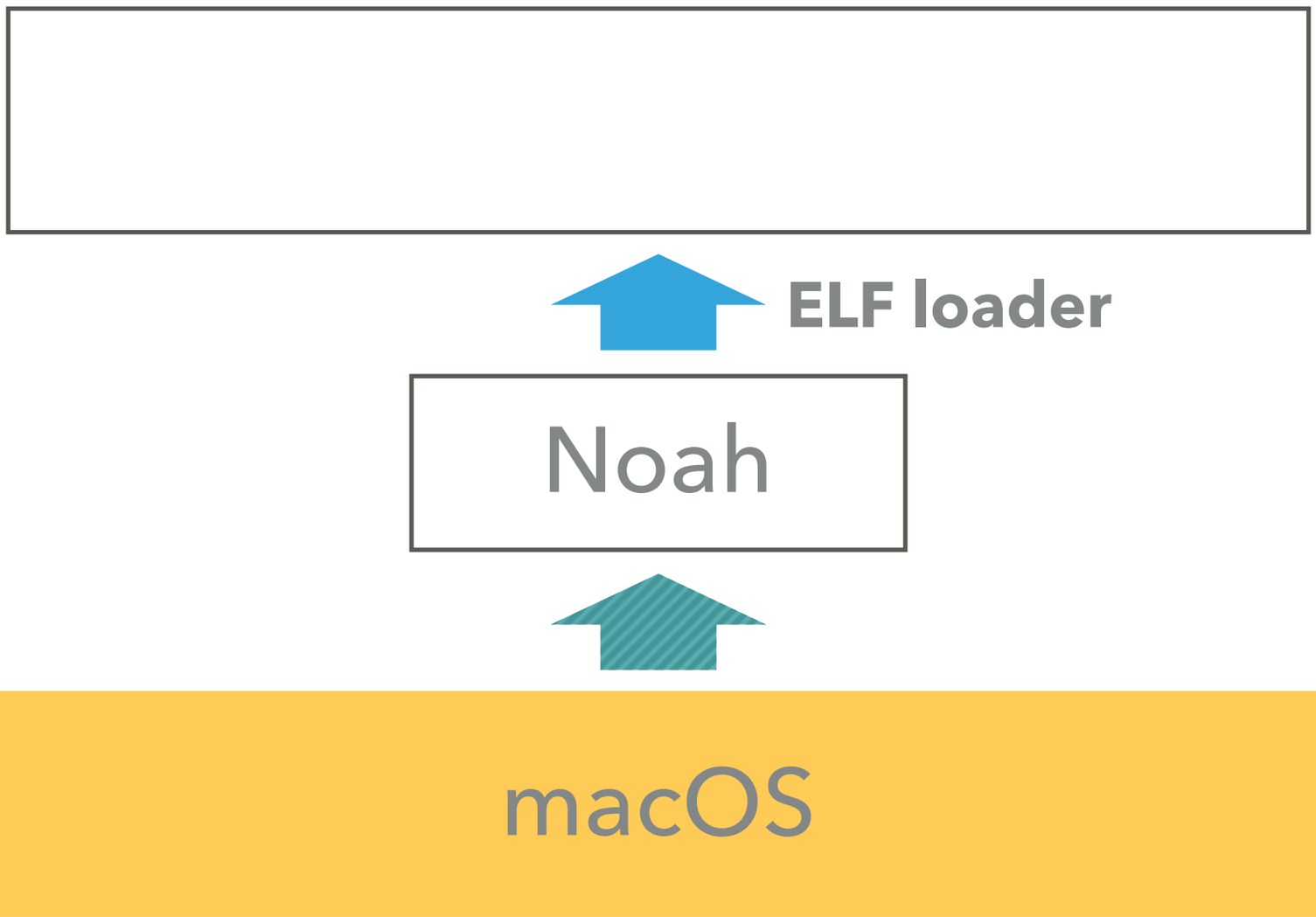
Example1：How “Hello, world” works

```
$ noah /bin/hello
```



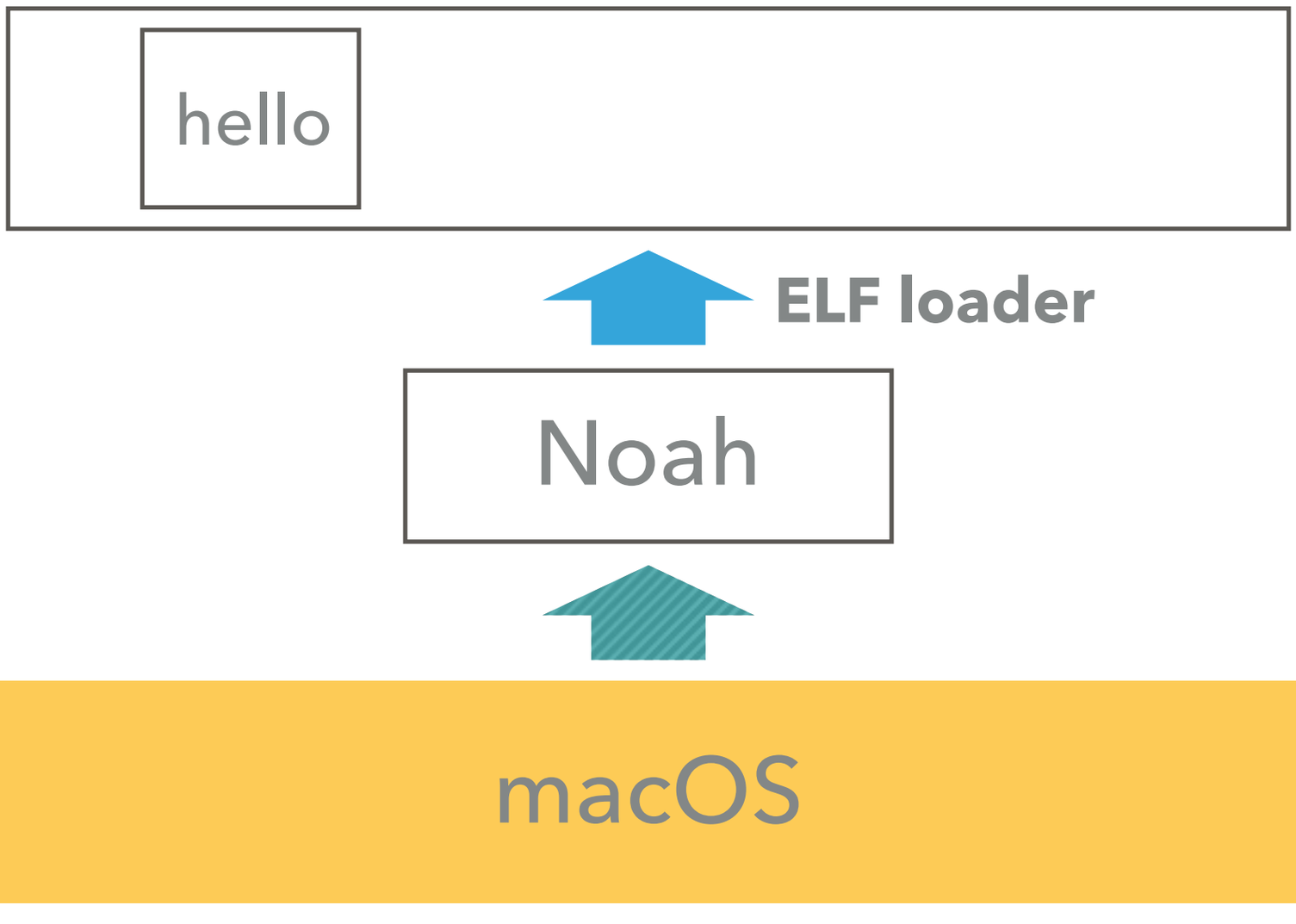
Example1: How “Hello, world” works

```
$ noah /bin/hello
```



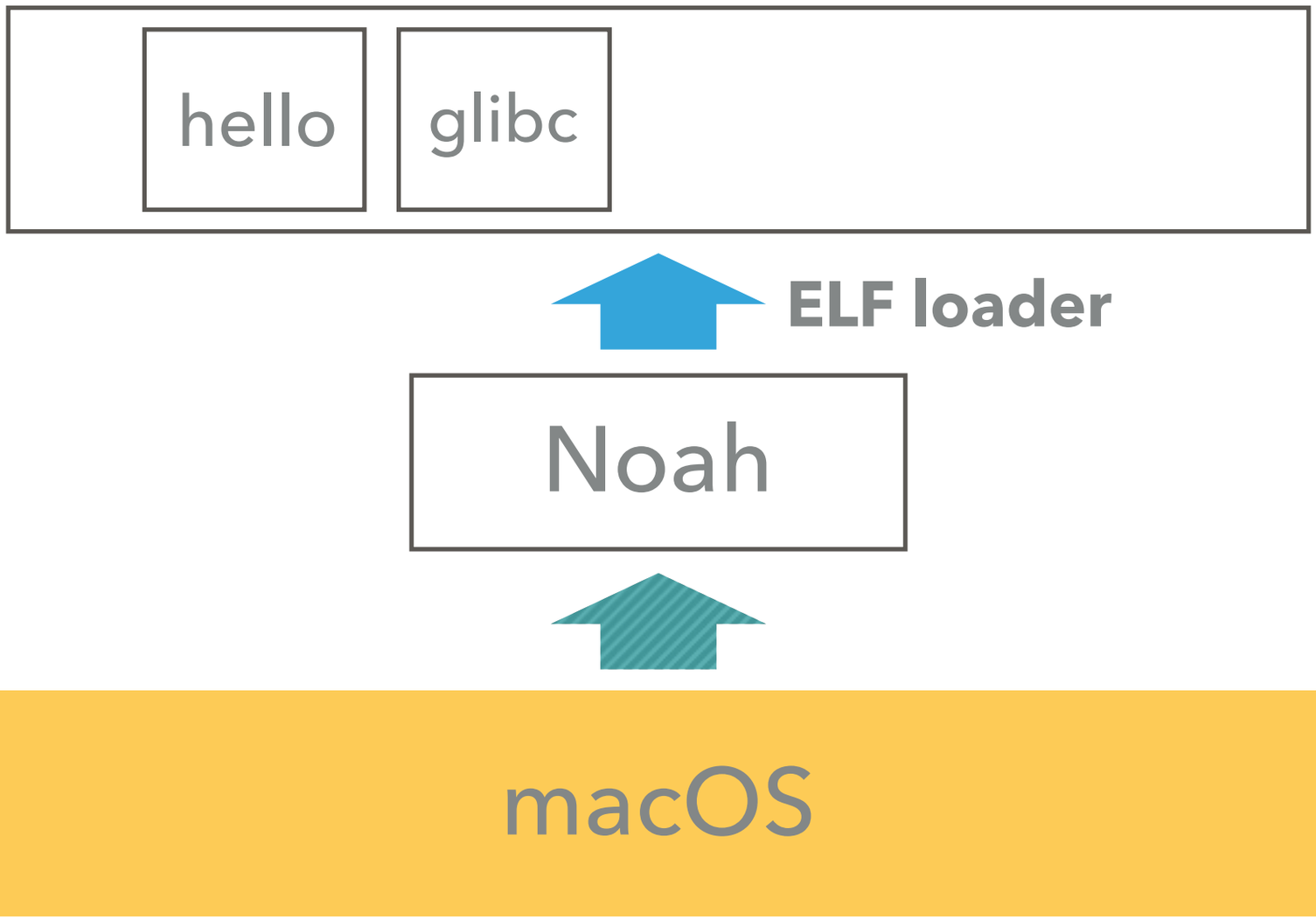
Example1: How “Hello, world” works

```
$ noah /bin/hello
```



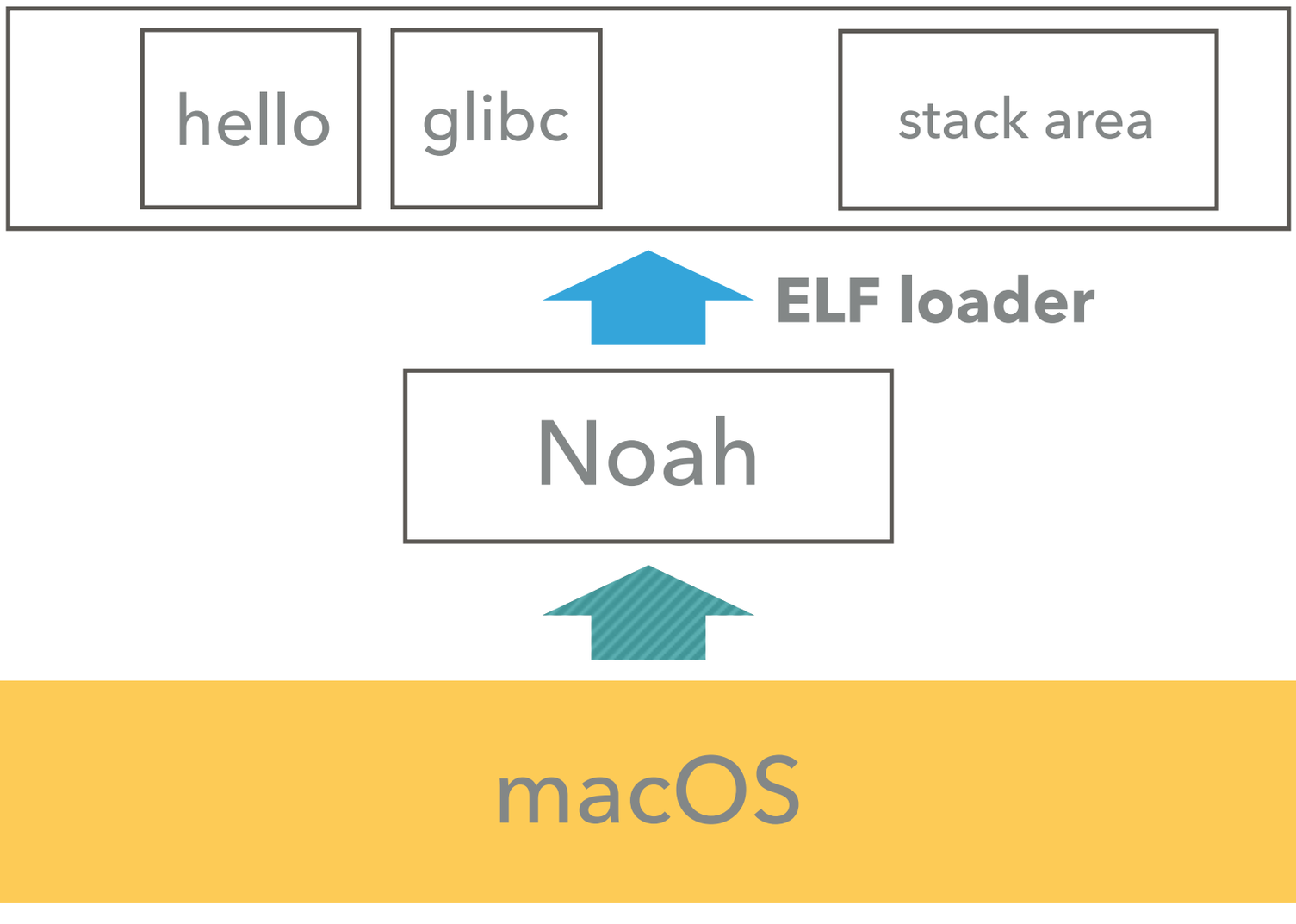
Example1：How “Hello, world” works

```
$ noah /bin/hello
```



Example1 : How “Hello, world” works

```
$ noah /bin/hello
```



Example1：How “Hello, world” works

```
$ noah /bin/hello
```

Run!

Noah

macOS

Example1 : How “Hello, world” works

```
$ noah /bin/hello
```

Example1 : How “Hello, world” works

```
$ noah /bin/hello
```

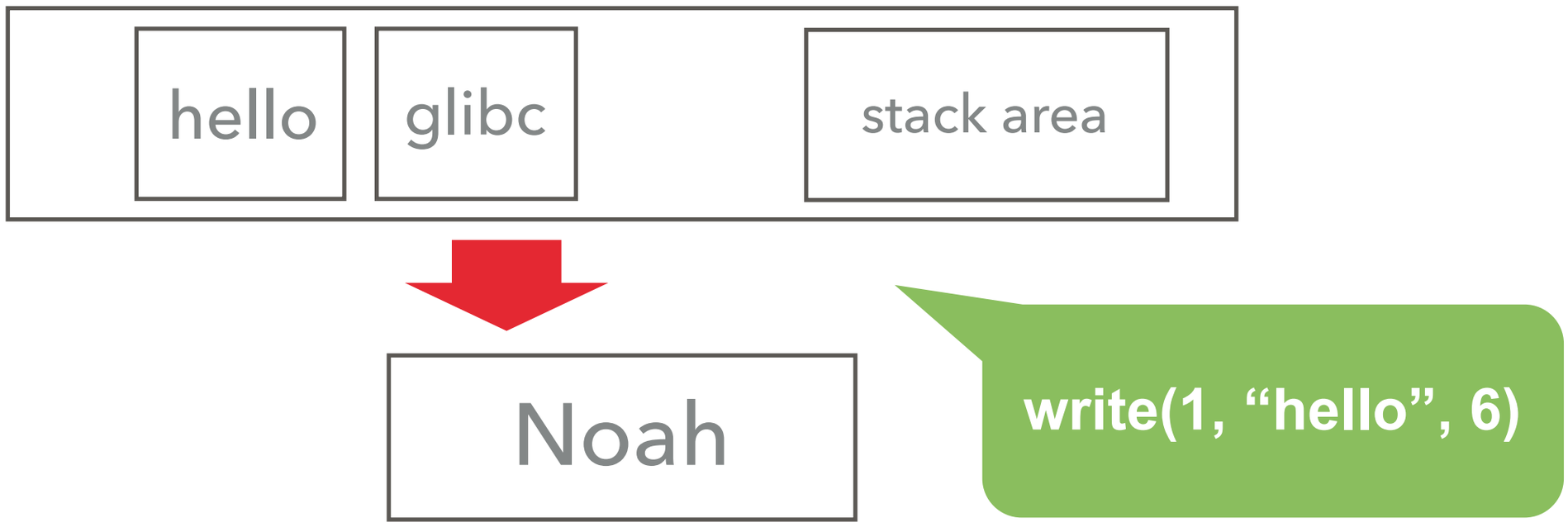
syscall!

Noah

macOS

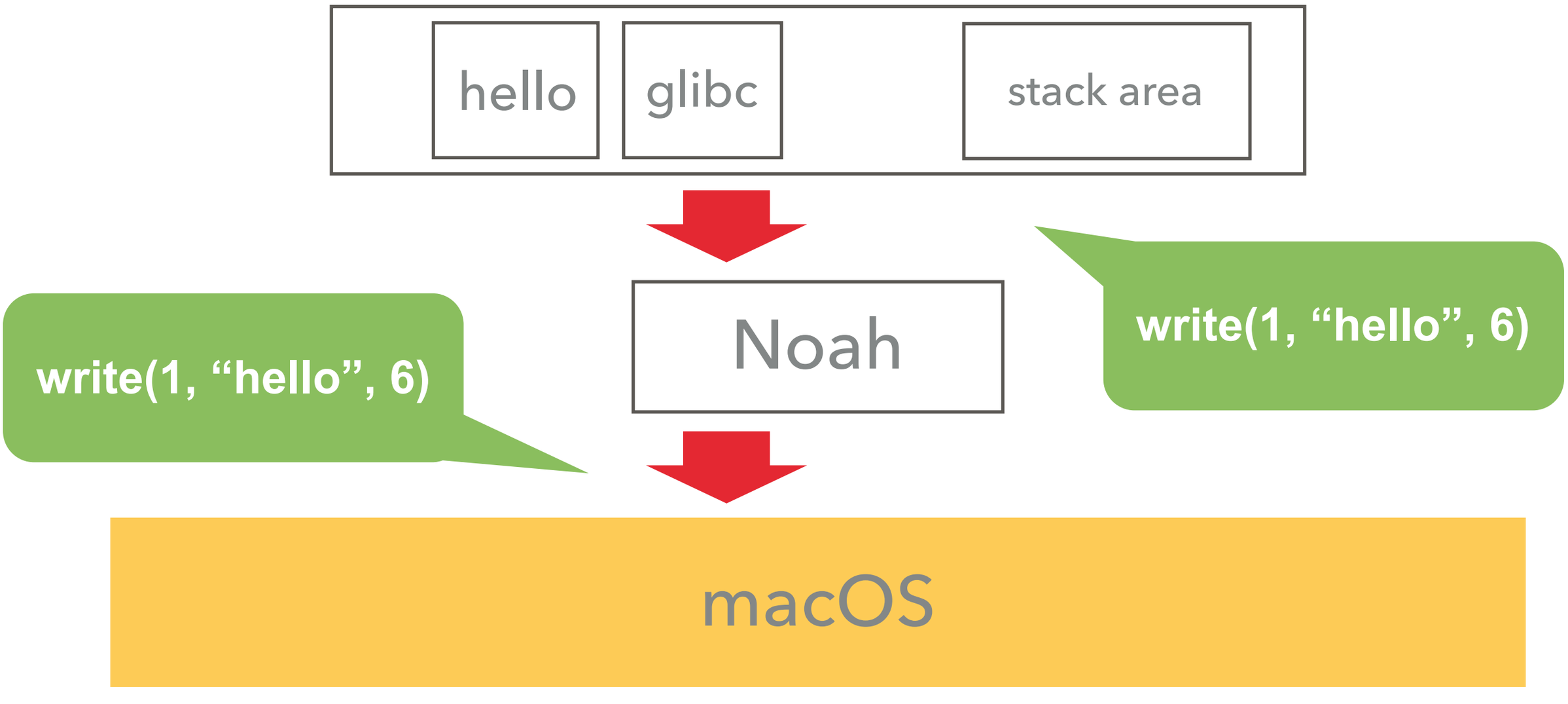
Example1 : How “Hello, world” works

```
$ noah /bin/hello
```



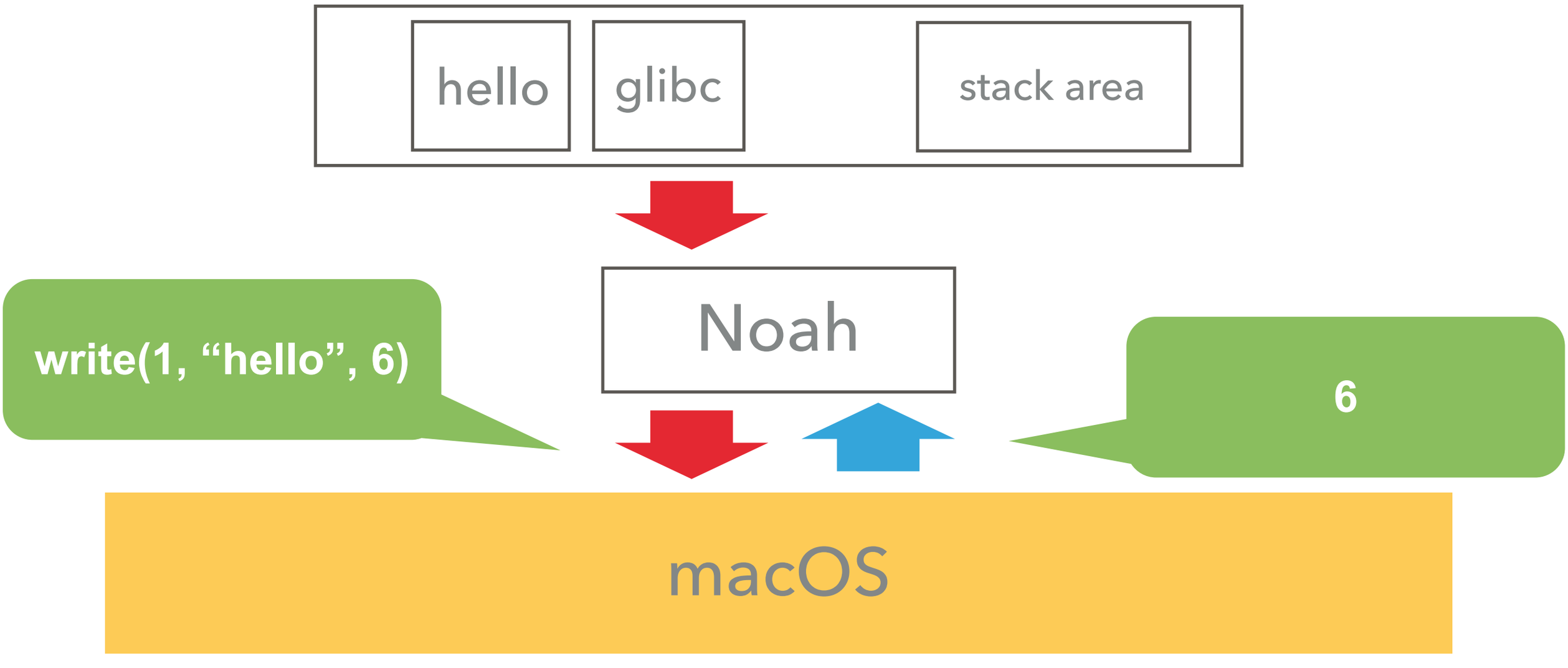
Example1 : How “Hello, world” works

```
$ noah /bin/hello
```



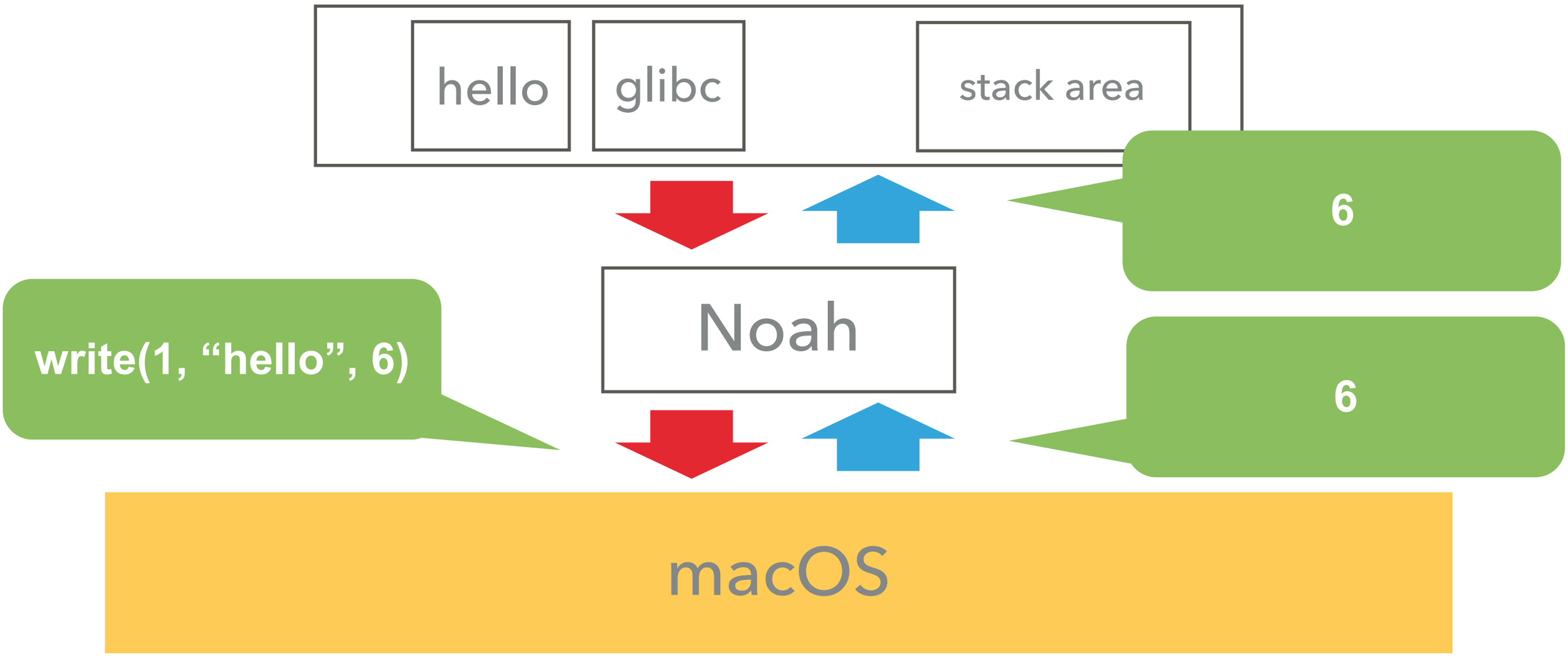
Example1 : How “Hello, world” works

```
$ noah /bin/hello
```



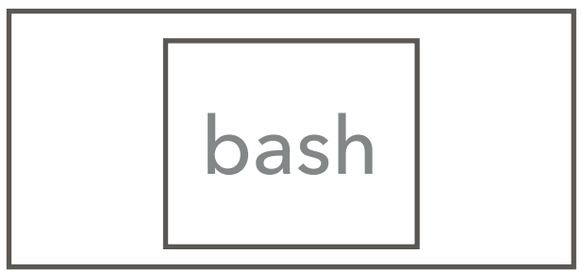
Example1 : How “Hello, world” works

```
$ noah /bin/hello
```



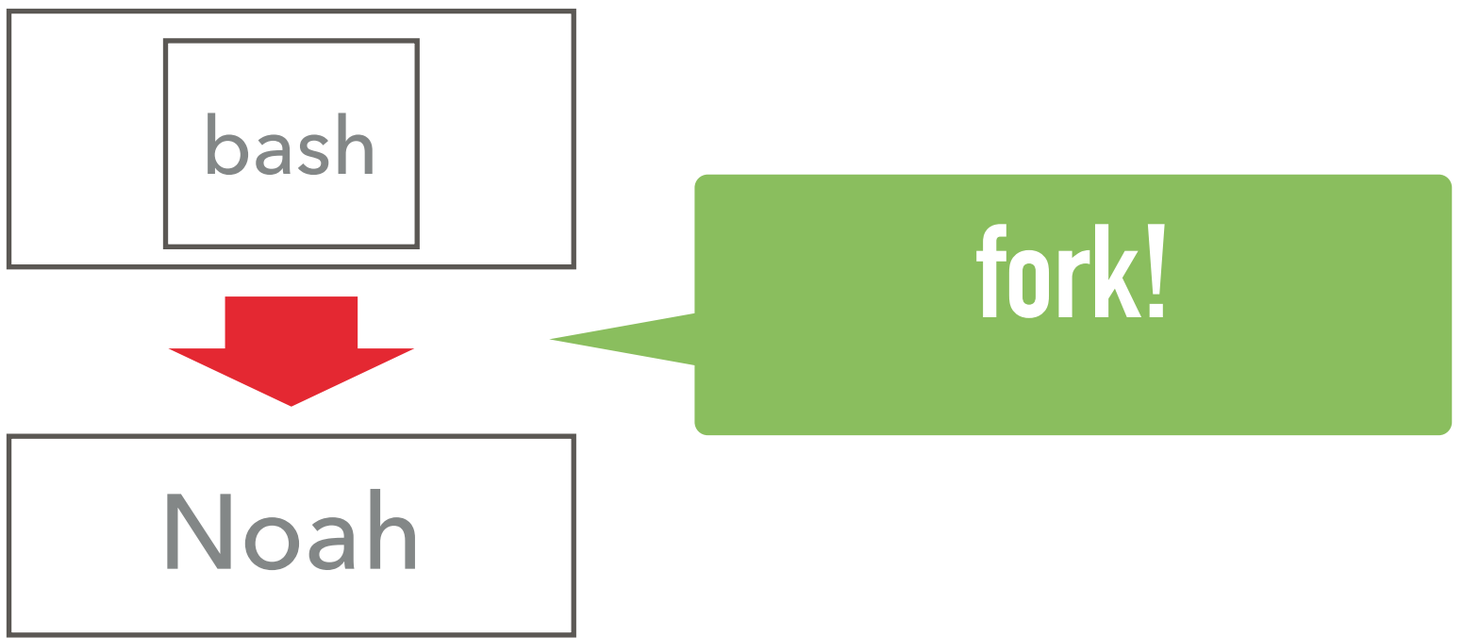
Example2: Interaction between processes

```
$ cat file | grep 2017
```



Example2: Interaction between processes

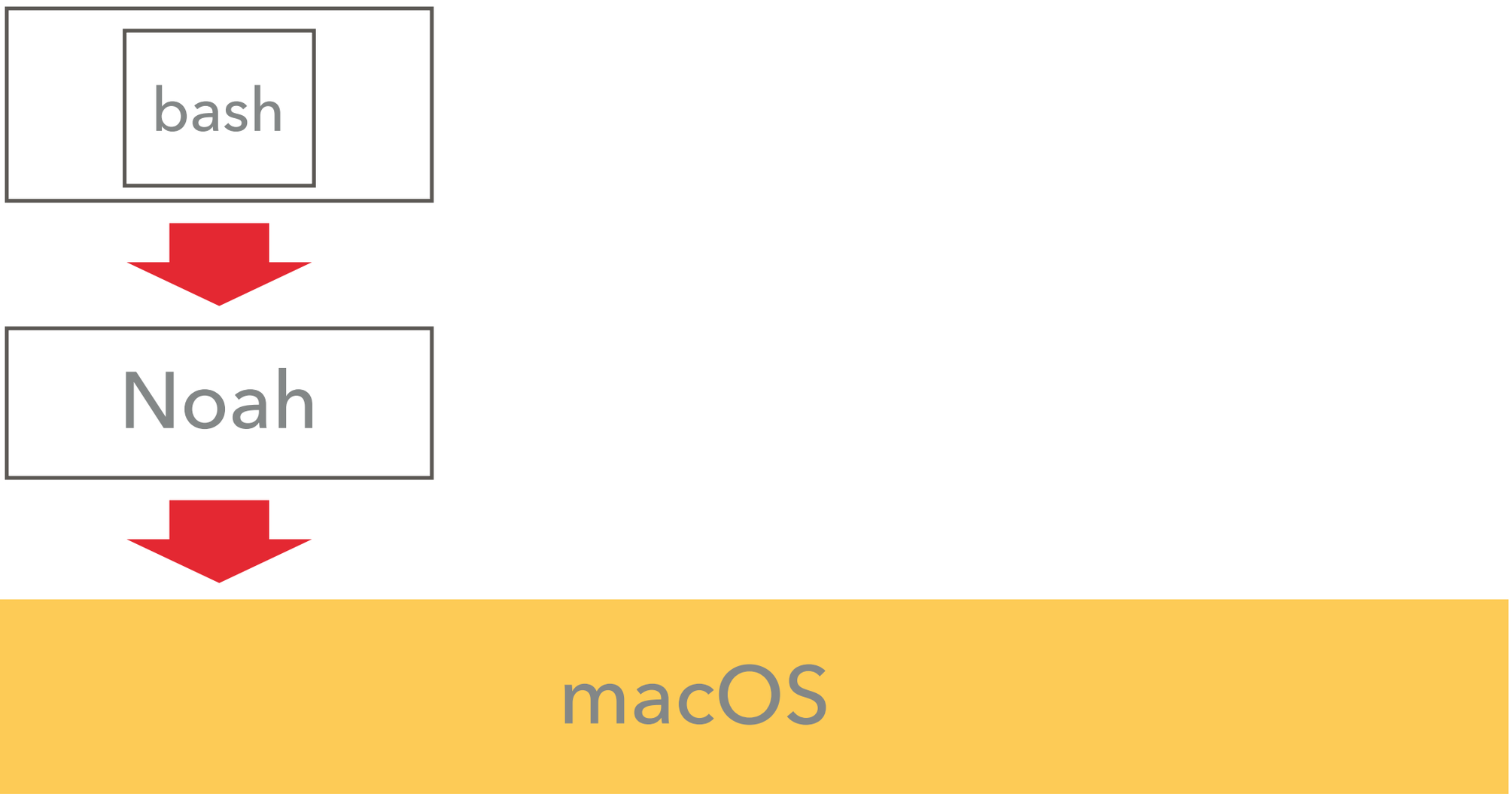
```
$ cat file | grep 2017
```



macOS

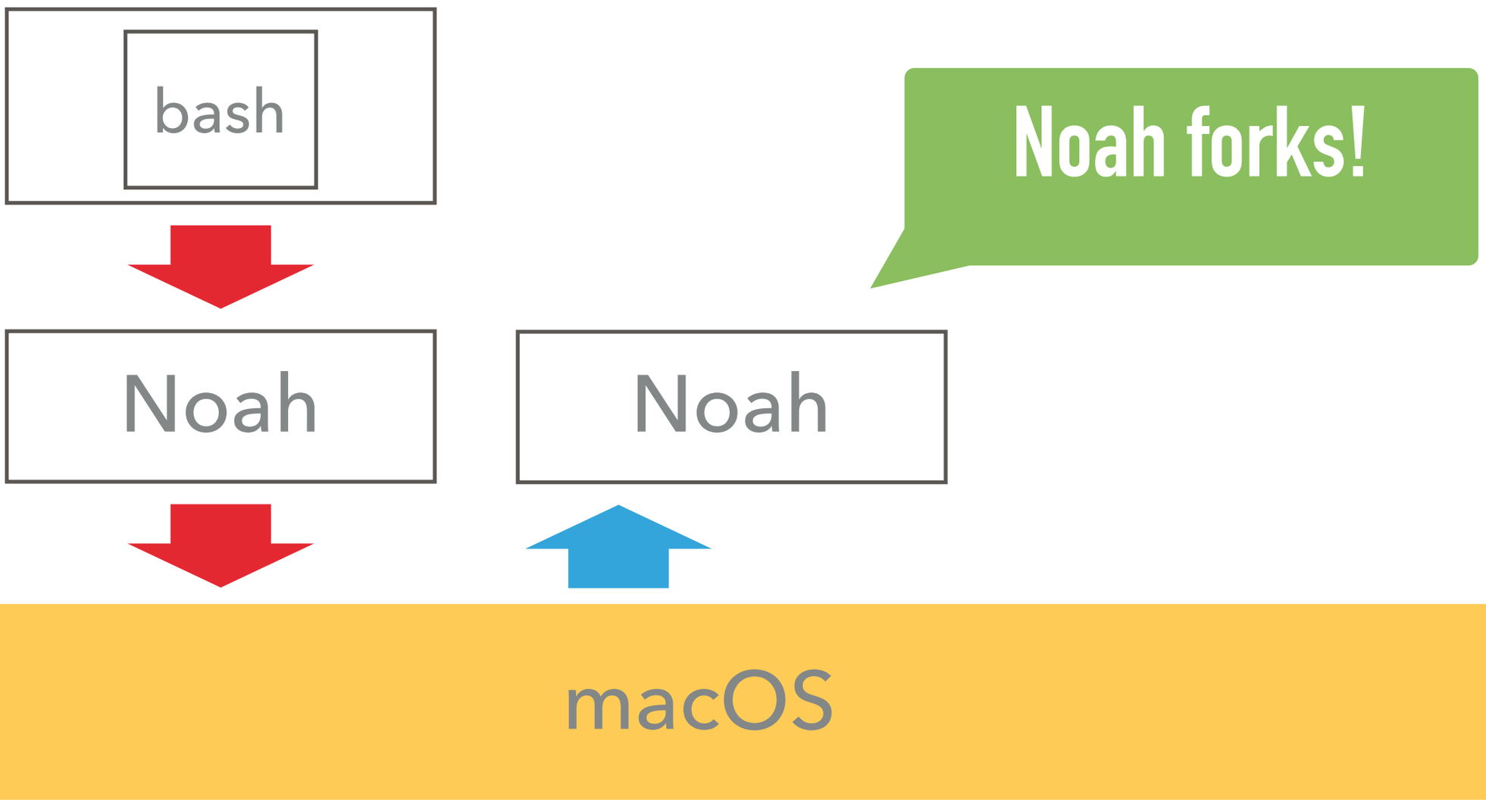
Example2: Interaction between processes

```
$ cat file | grep 2017
```



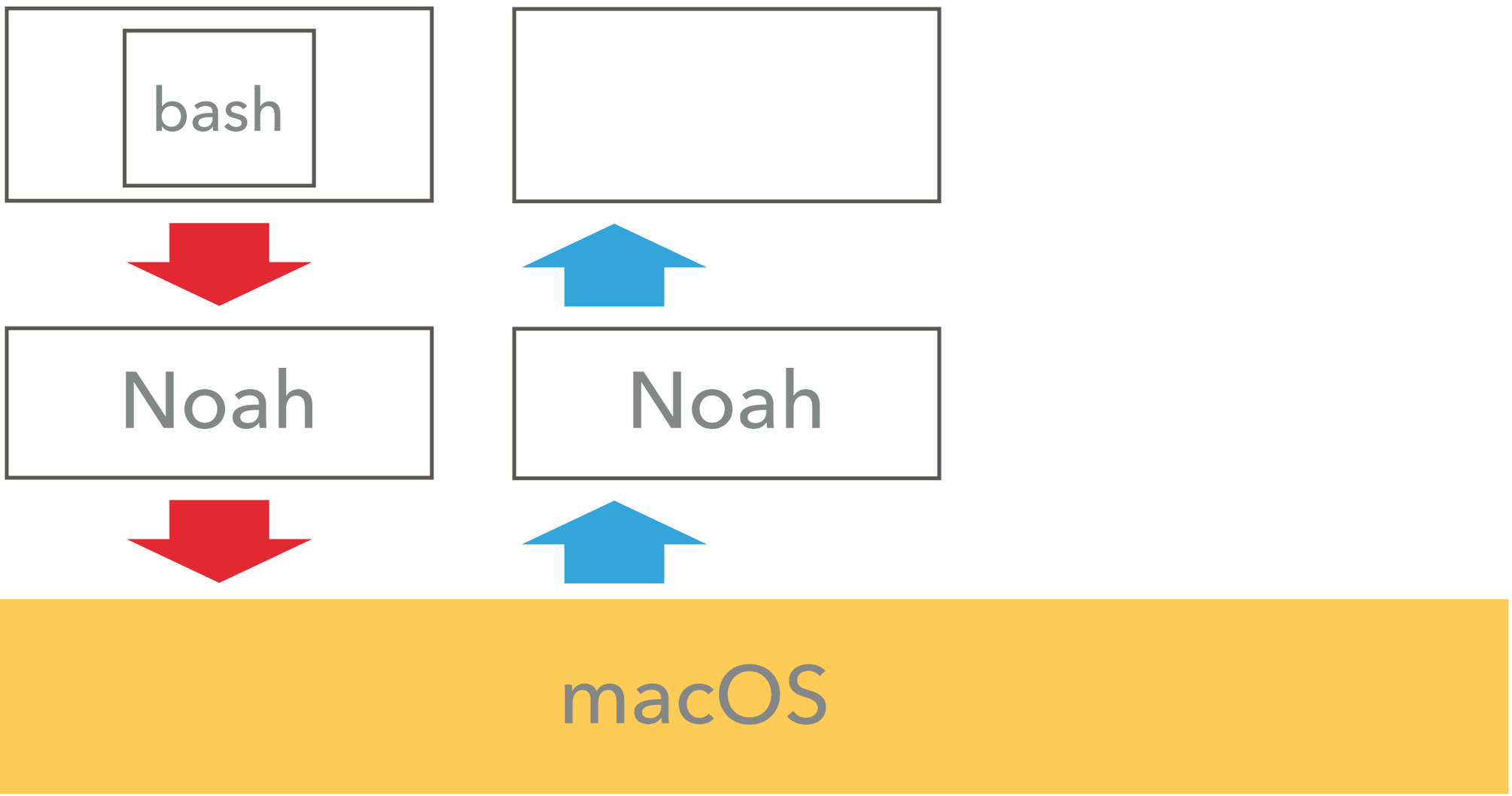
Example2: Interaction between processes

```
$ cat file | grep 2017
```



Example2: Interaction between processes

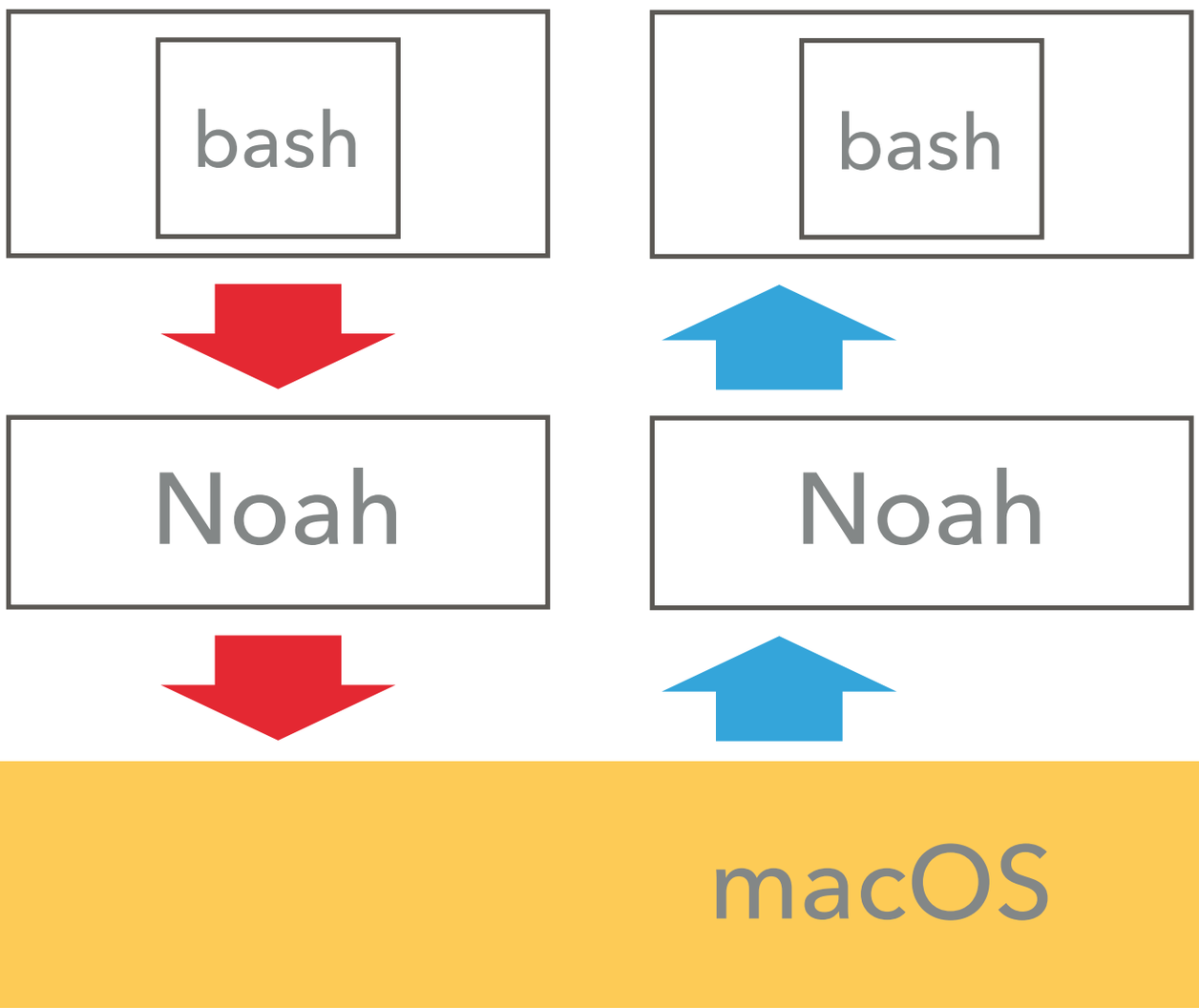
```
$ cat file | grep 2017
```



Example2: Interaction between p

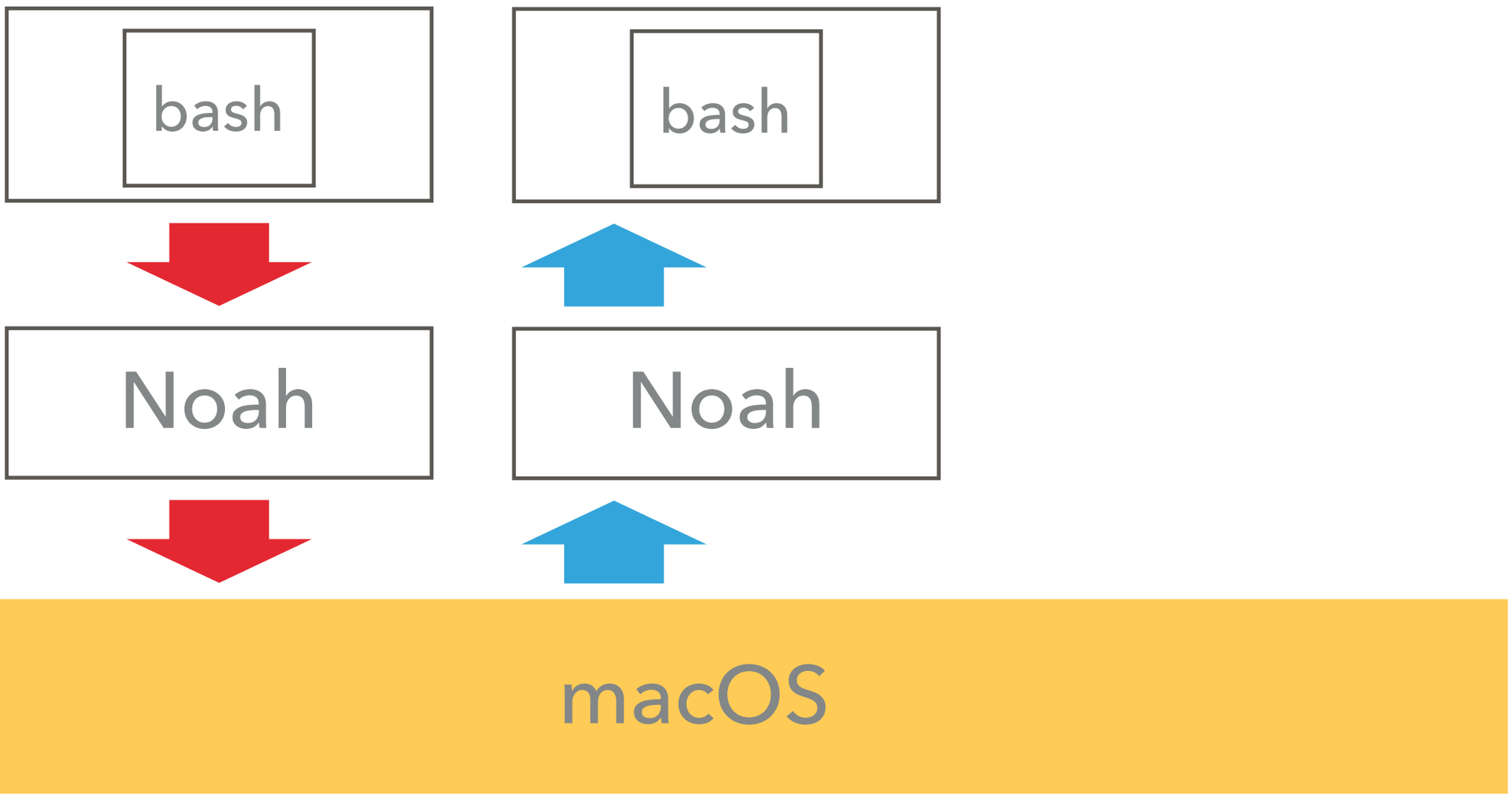
```
$ cat file | grep 201
```

Clone the VM state



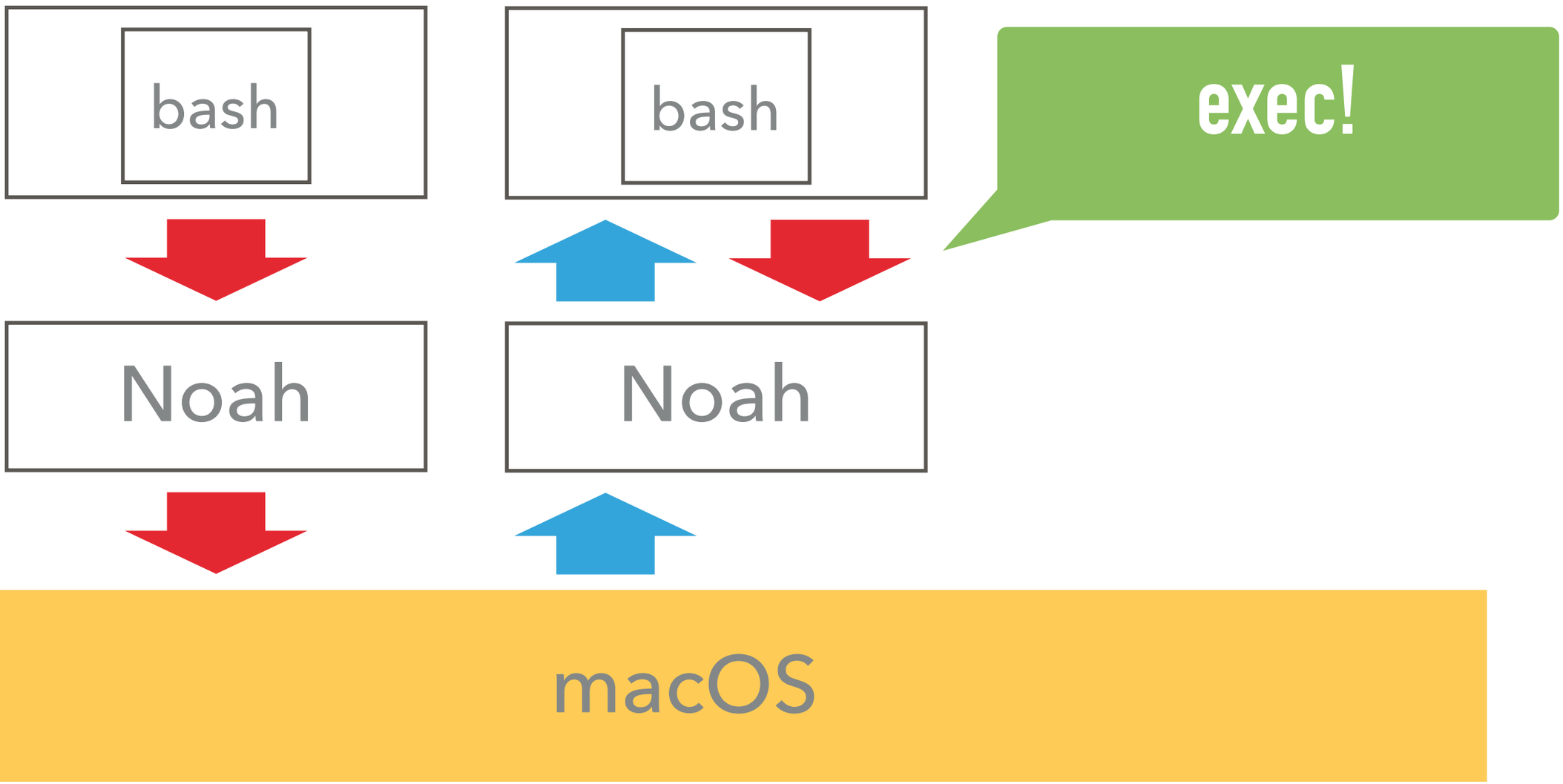
Example2: Interaction between processes

```
$ cat file | grep 2017
```



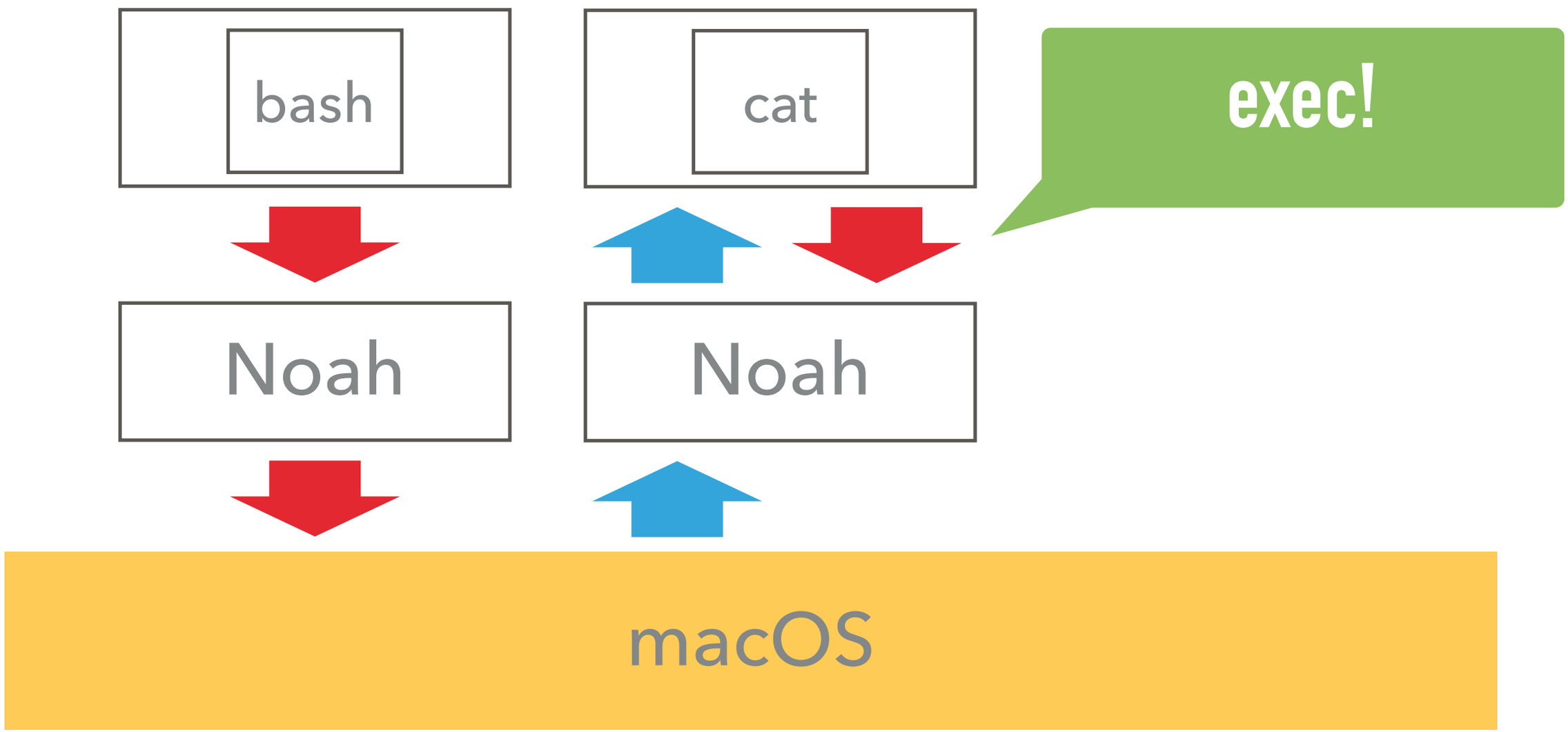
Example2: Interaction between processes

```
$ cat file | grep 2017
```



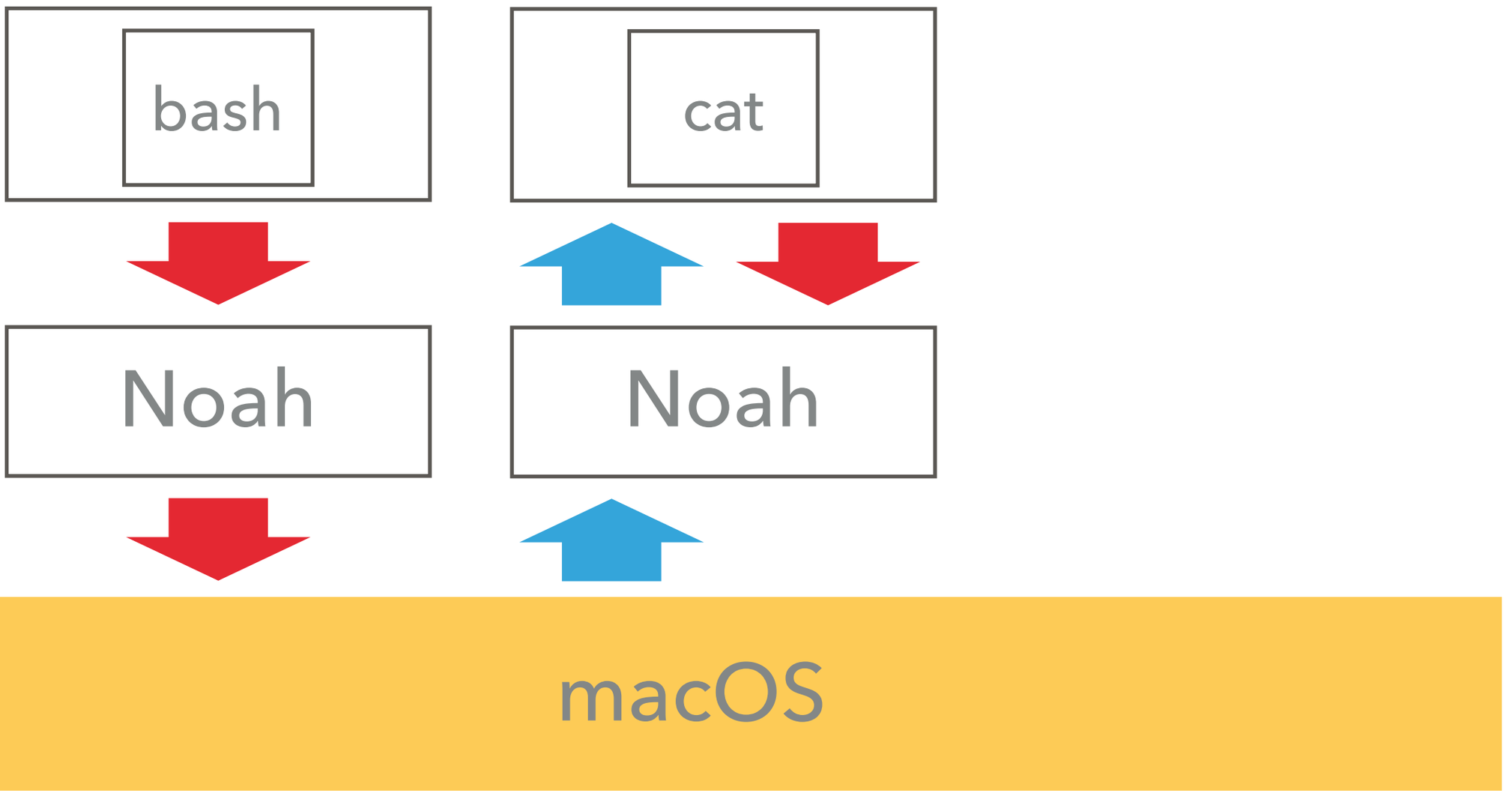
Example2: Interaction between processes

```
$ cat file | grep 2017
```



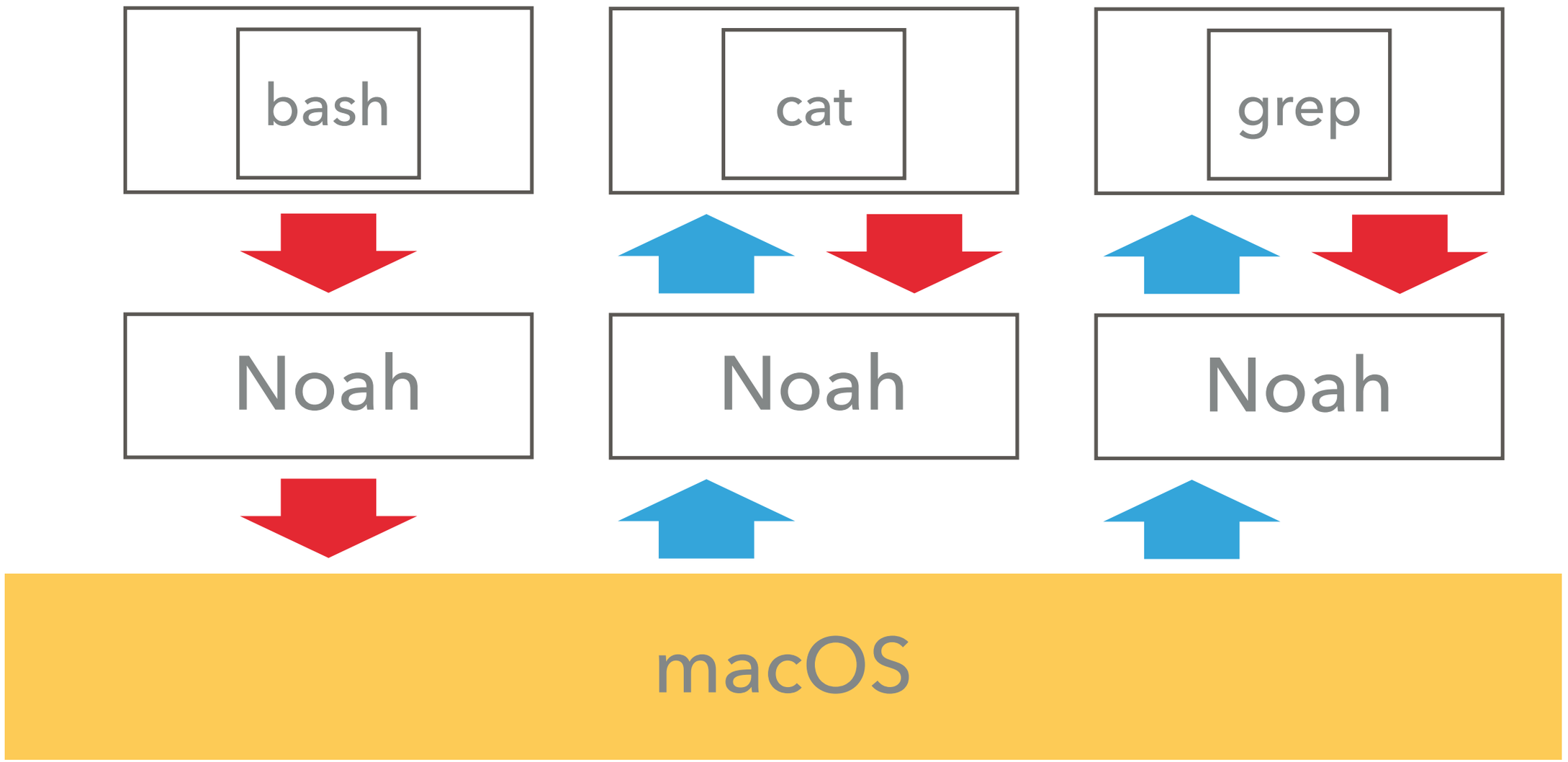
Example2: Interaction between processes

```
$ cat file | grep 2017
```



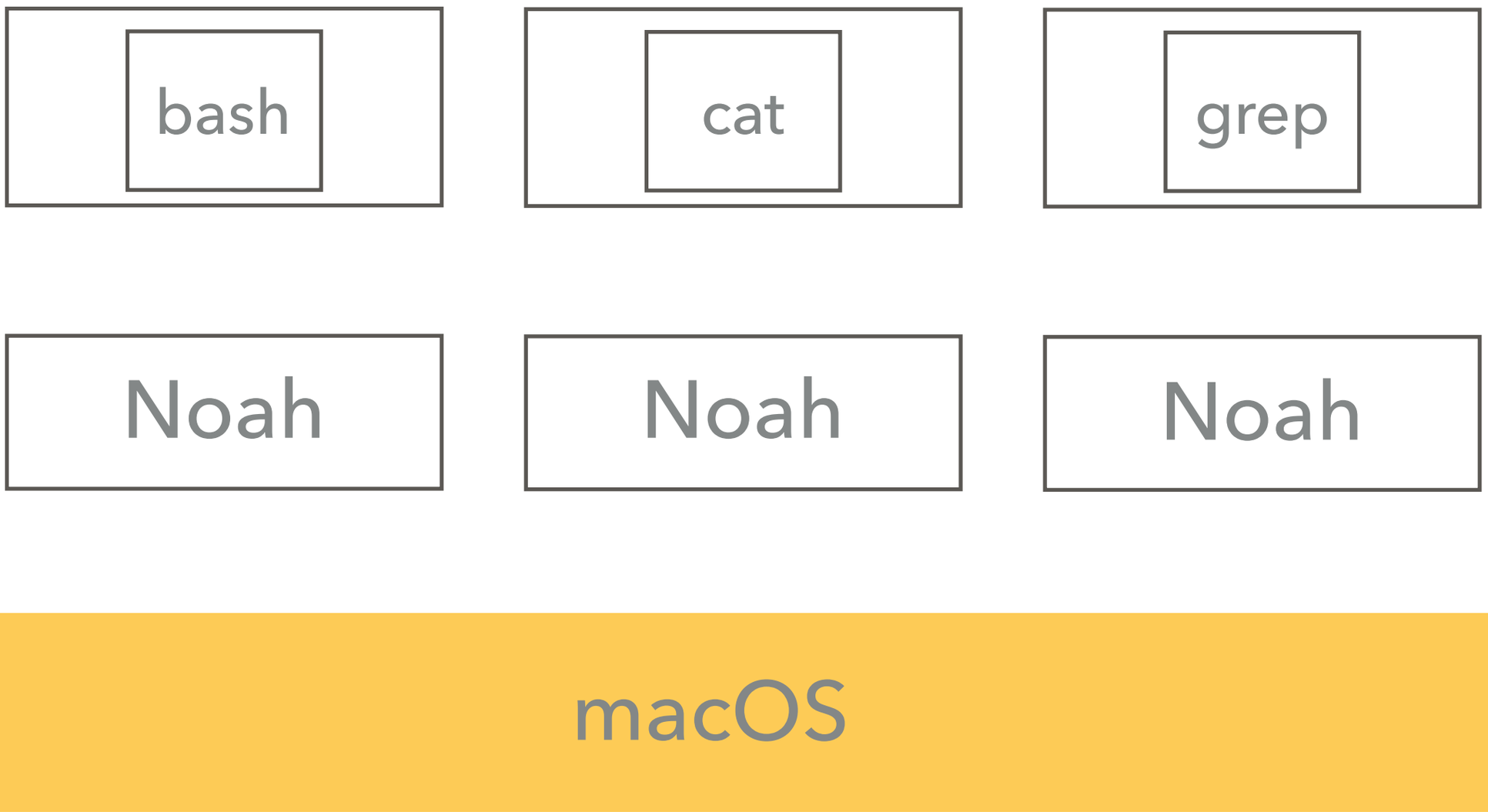
Example2: Interaction between processes

```
$ cat file | grep 2017
```



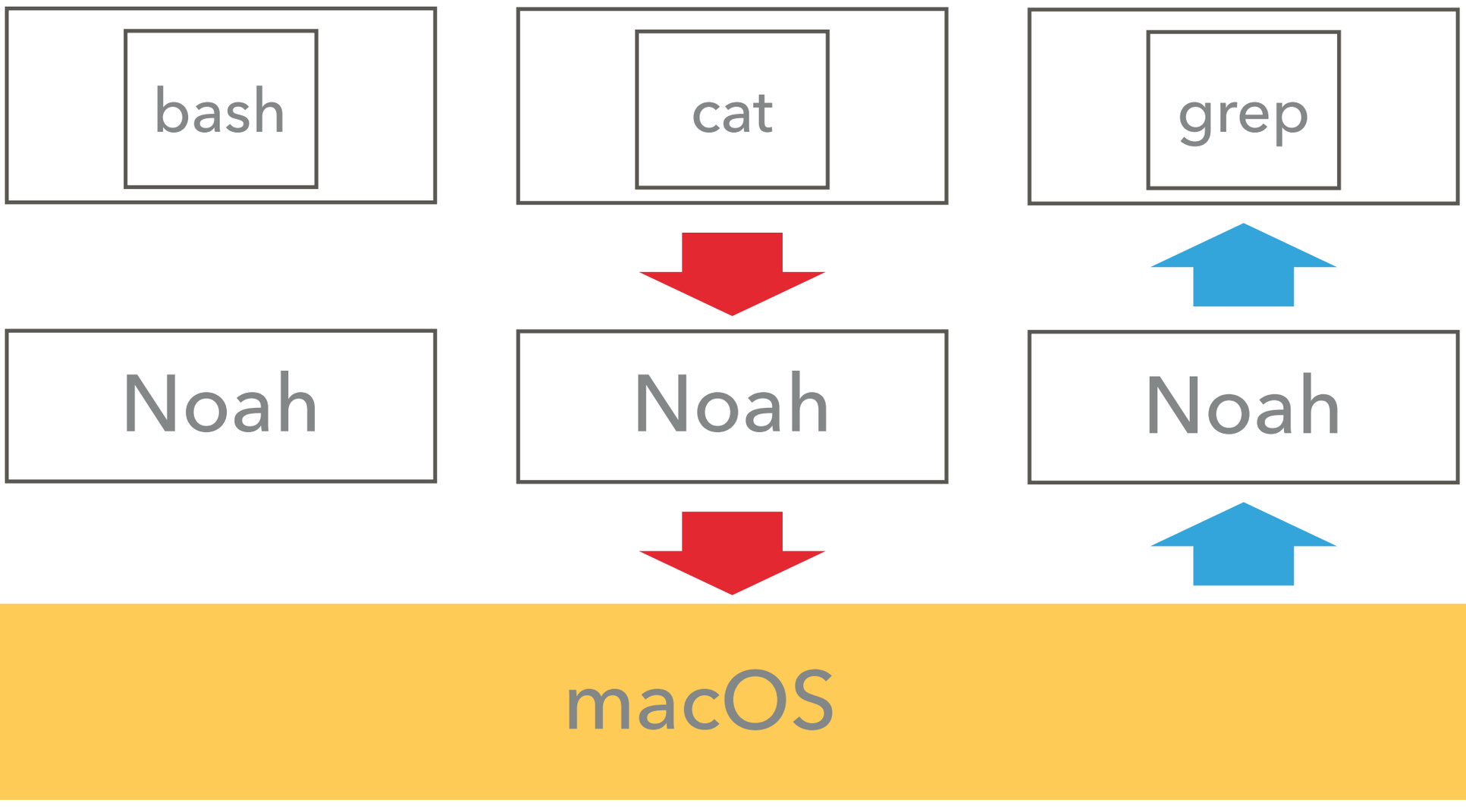
Example2: Interaction between processes

```
$ cat file | grep 2017
```



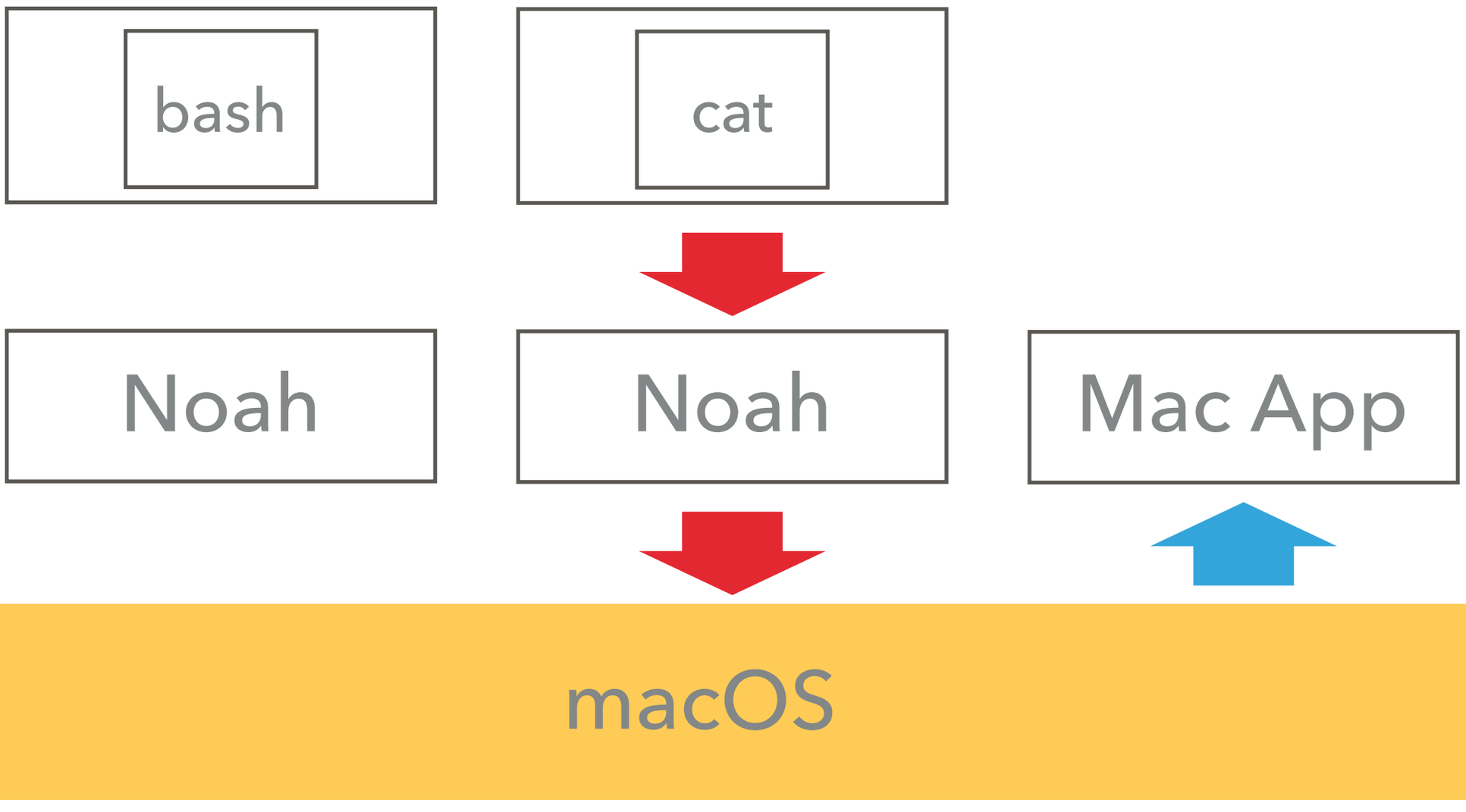
Example2: Interaction between processes

```
$ cat file | grep 2017
```



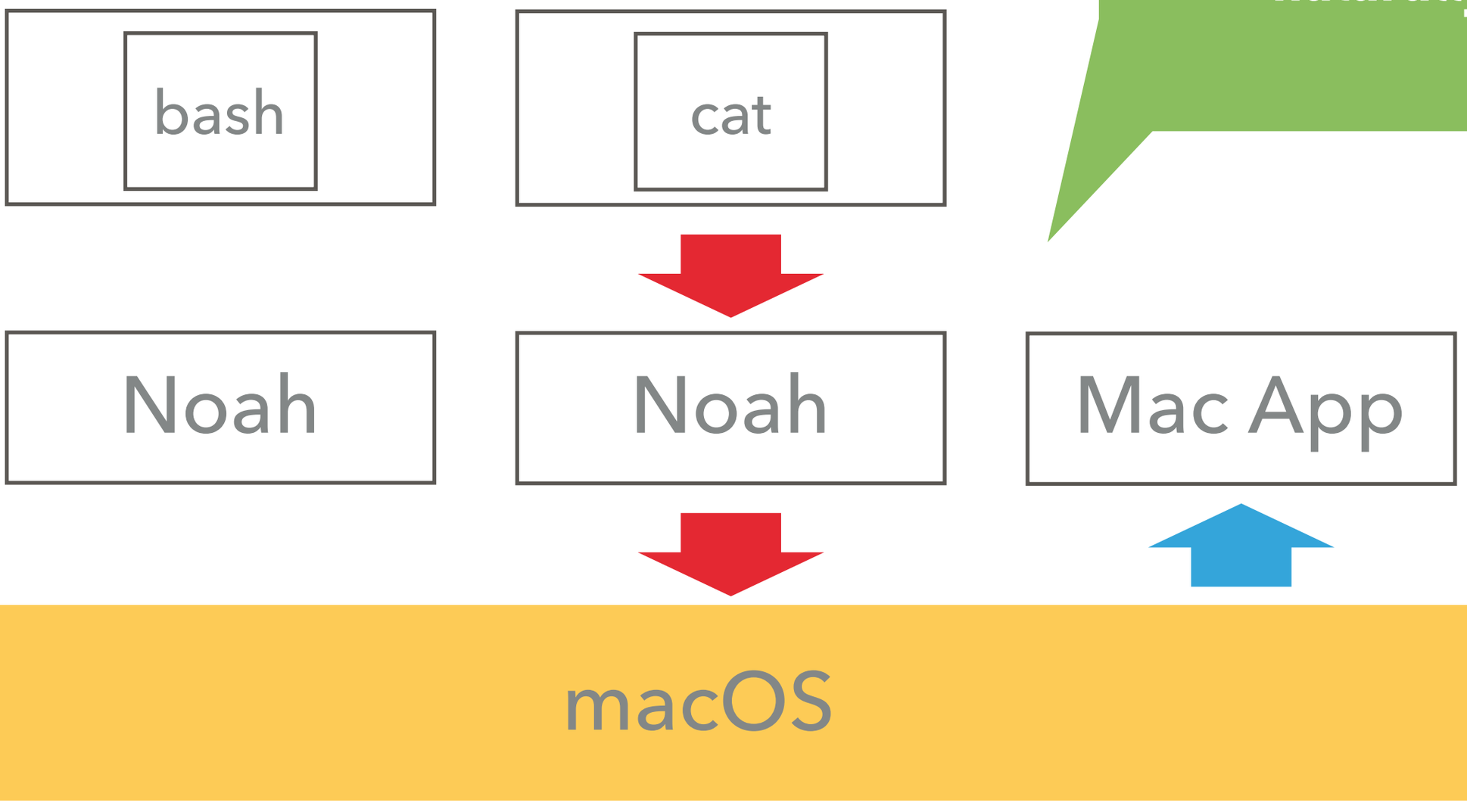
Example2: Interaction between processes

```
$ cat file | grep 2017
```



Example2: Interaction between processe

```
$ cat file | grep 2017
```



Linux and macOS applications can communicate naturally

Advantages of Noah Architecture

Unique Characteristics

1. All syscall translation is done in user land instead of kernel land
    - Still, any sensitive events are trappable with VT-x
2. Launch as many VMs as virtual Linux processes
    - No kernel running inside VMs
3. Virtualization is per syscall, not per device I/O
    - No need to worry about hardware device emulation

Advantages of Noah Architecture

1. Robustness

   Bugs in Noah never cause kernel panic because Noah is just an ordinary userland program (let's google "WSL bluescreen" now).

2. Portability

   The architecture is independent from host OS's architecture. Syscall calling convention, memory layout, page fault handling rules, …etc are all configurable.

3. Smooth interaction with host OS

   Linux process runs as if it is the host OS's process. Resources such as memory, network, and so on are managed by host OS. No need to worry about the amount of virtual memory allocation like full virtual machines.

Agenda

- ~~What is Noah~~
- ~~Background~~
- ~~Noah in Detail~~
  - ~~Architecture Overview~~
  - ~~Advantages of Noah Architecture~~
  - **Subsystem Implementation**
    - Memory management, VFS, and the other
- Current Implementation Status and Performance
- Related Technologies and Comparison
  (Windows Subsystem for Linux, Linuxulator, and so on)
- Their Possible Impact on Cross Platform Development

Subsystem Implementation

Noah Subsystems

Noah consists of subsystems such as memory management, IPC, or file system just like a real kernel.

Needs ingenuity to implement some of them because of the nature of Noah’s architecture.

Noah Subsystems

Today we explain two subsystems in detail.

1. Memory management
2. Virtual file system

Memory Management

Memory Management

- Since a Linux ELF binary runs inside a VM, Noah must manage address translation between the VM memory space and the host memory space

- It gives us copy on write ability, efficient exec implementation, but also some difficulty

Memory Translation

LINUX APPLICATION

Guest

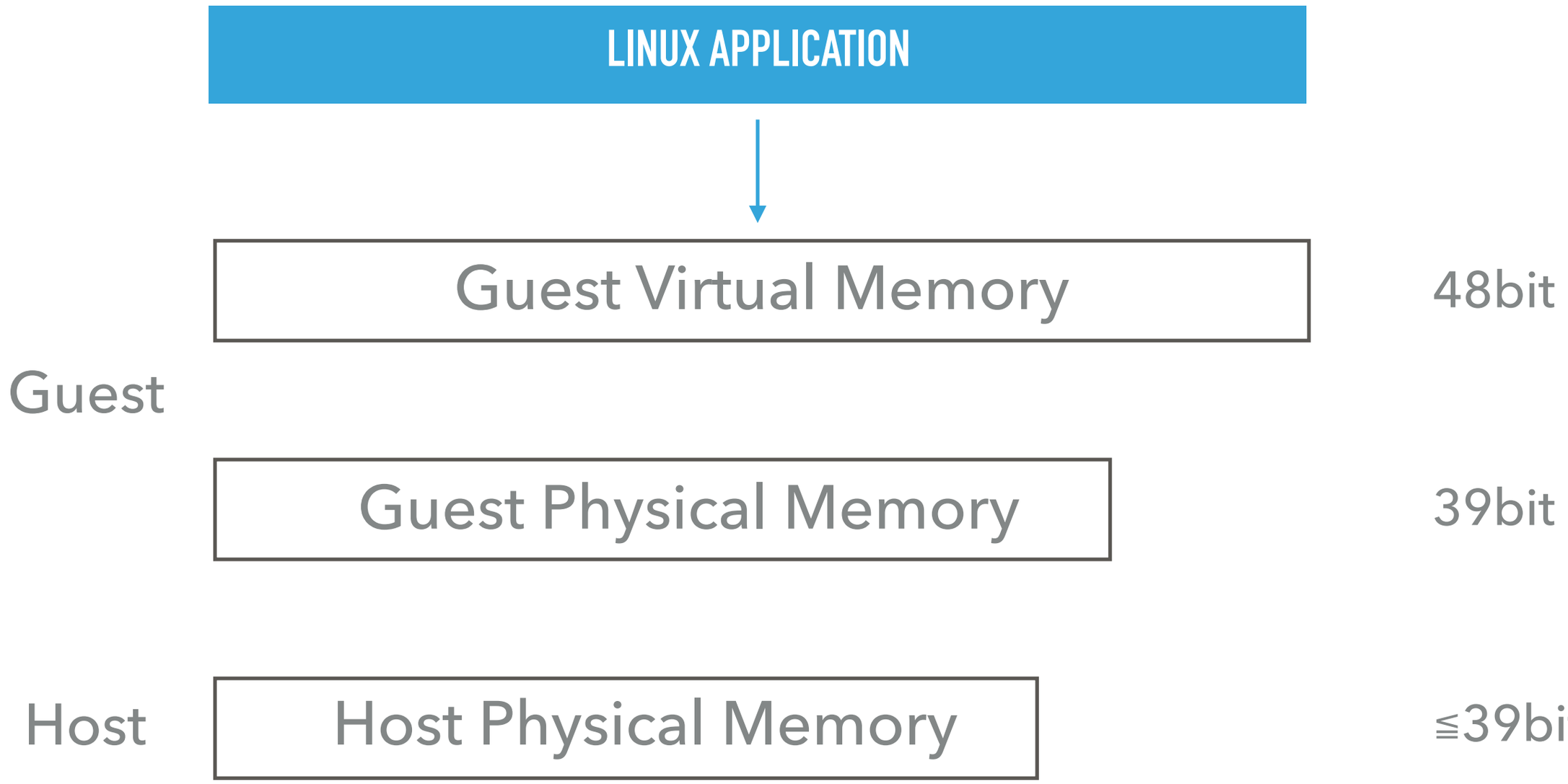
Guest Virtual Memory — 48bit

Guest Physical Memory — 39bit

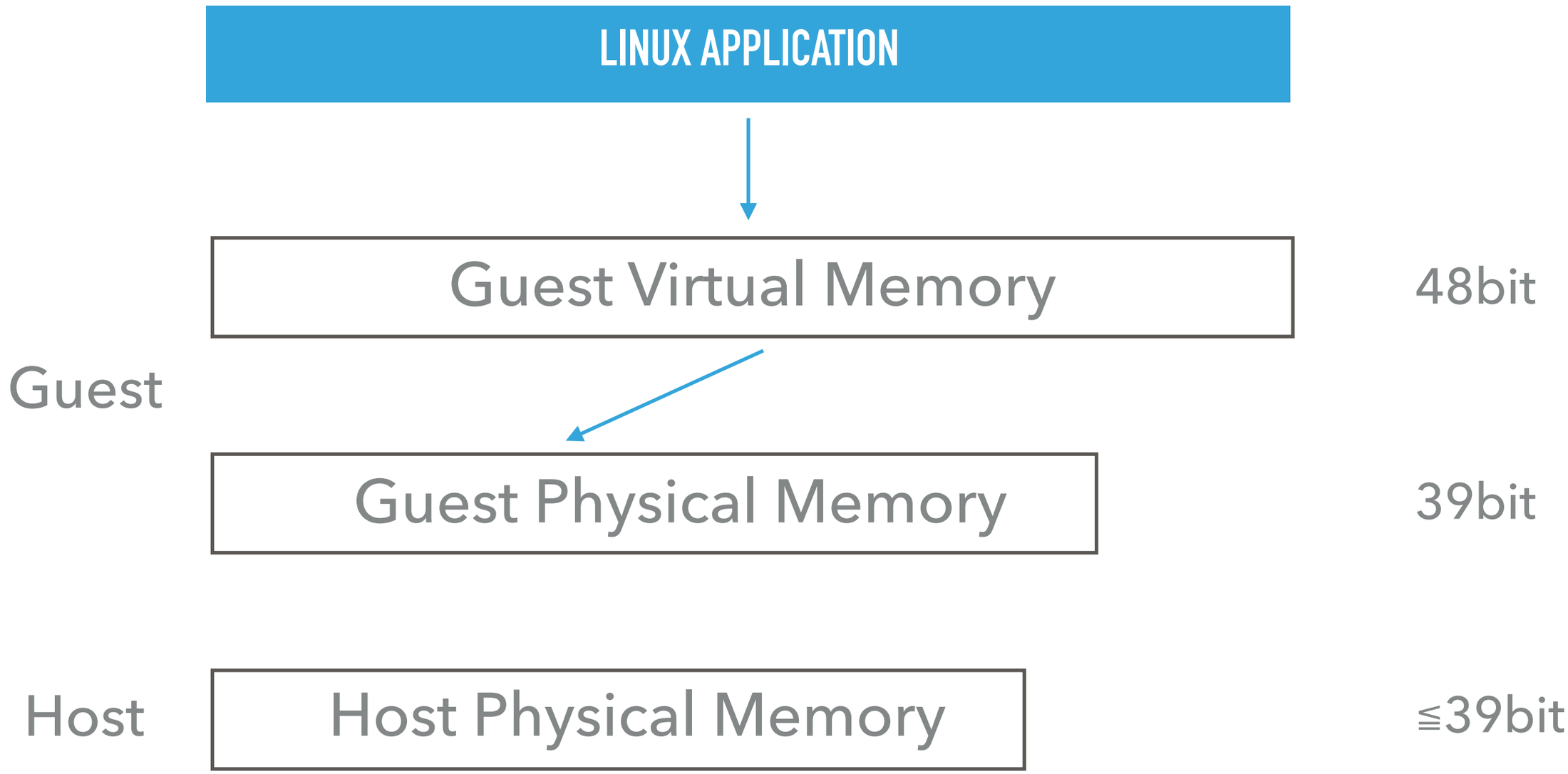
Host

Host Physical Memory — ≦39bit

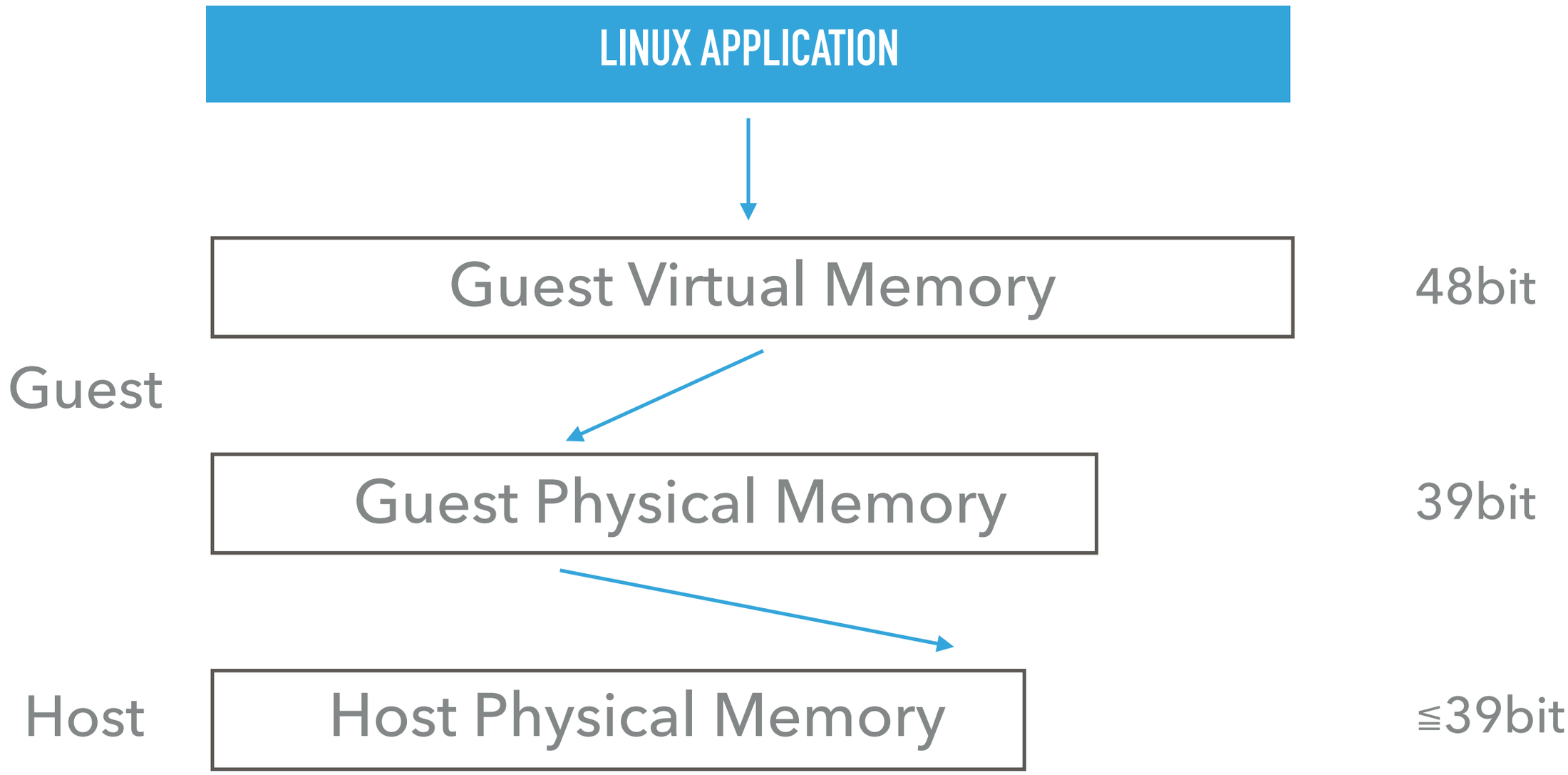
Memory Translation



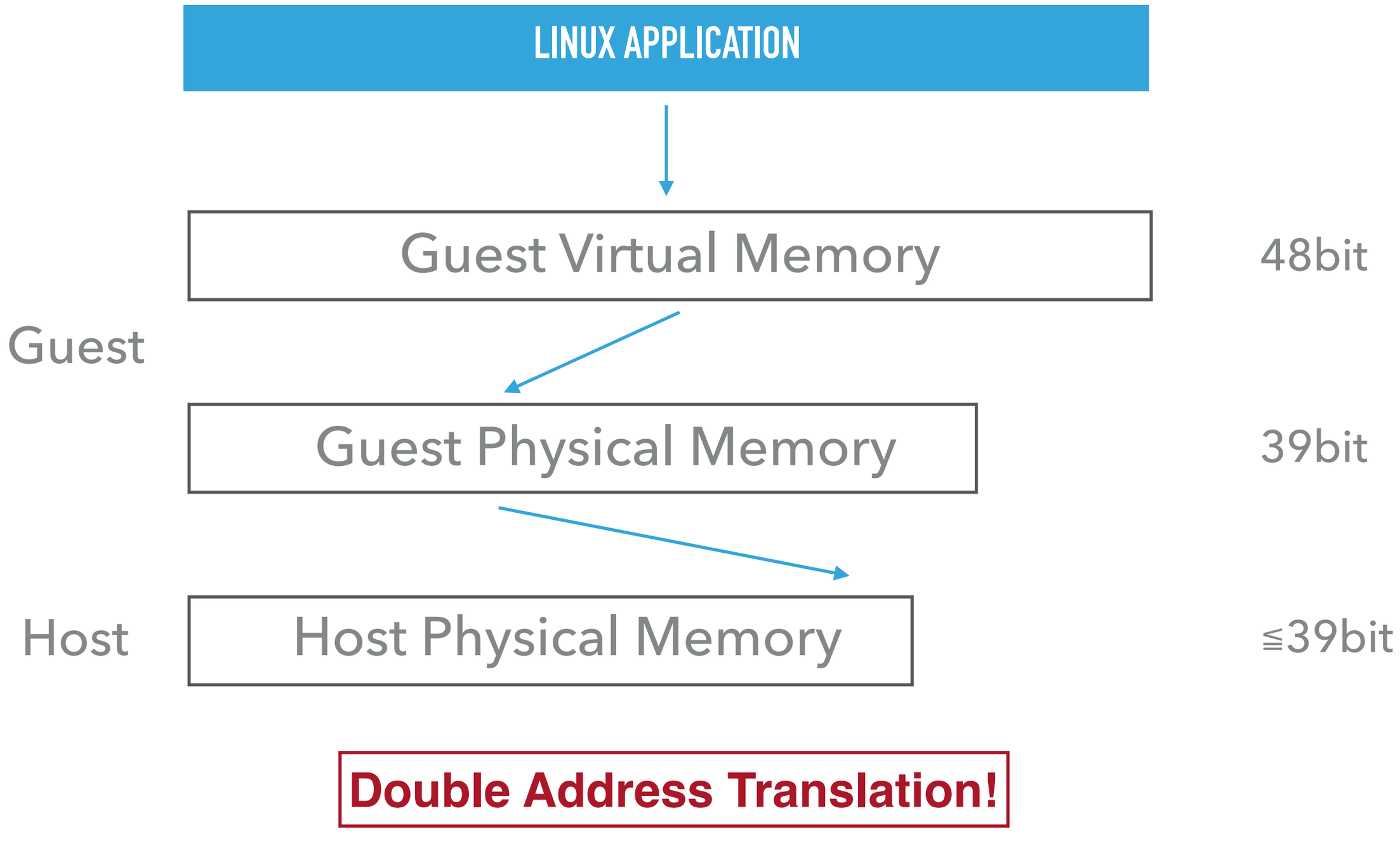
Memory Translation



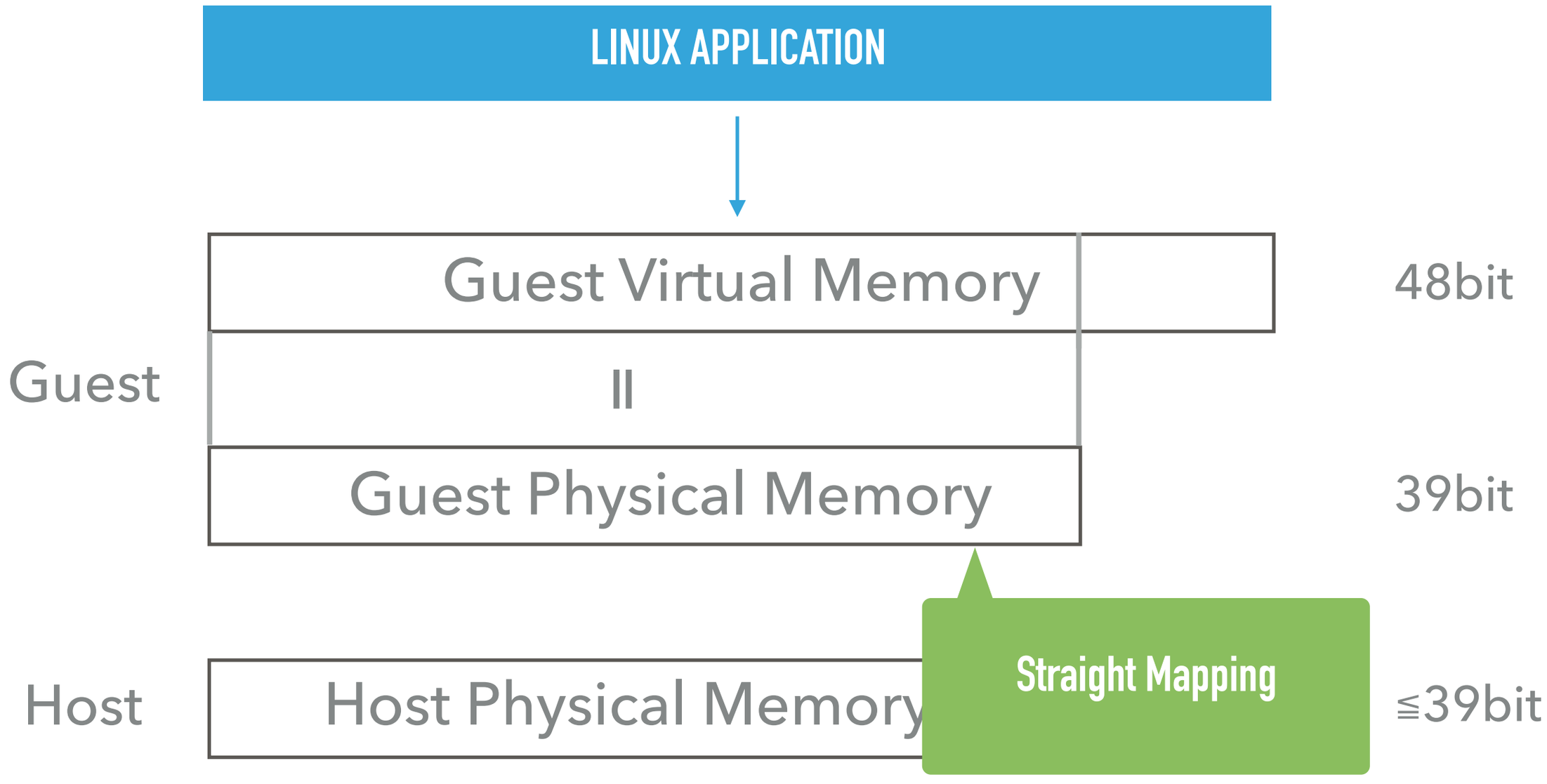
Memory Translation



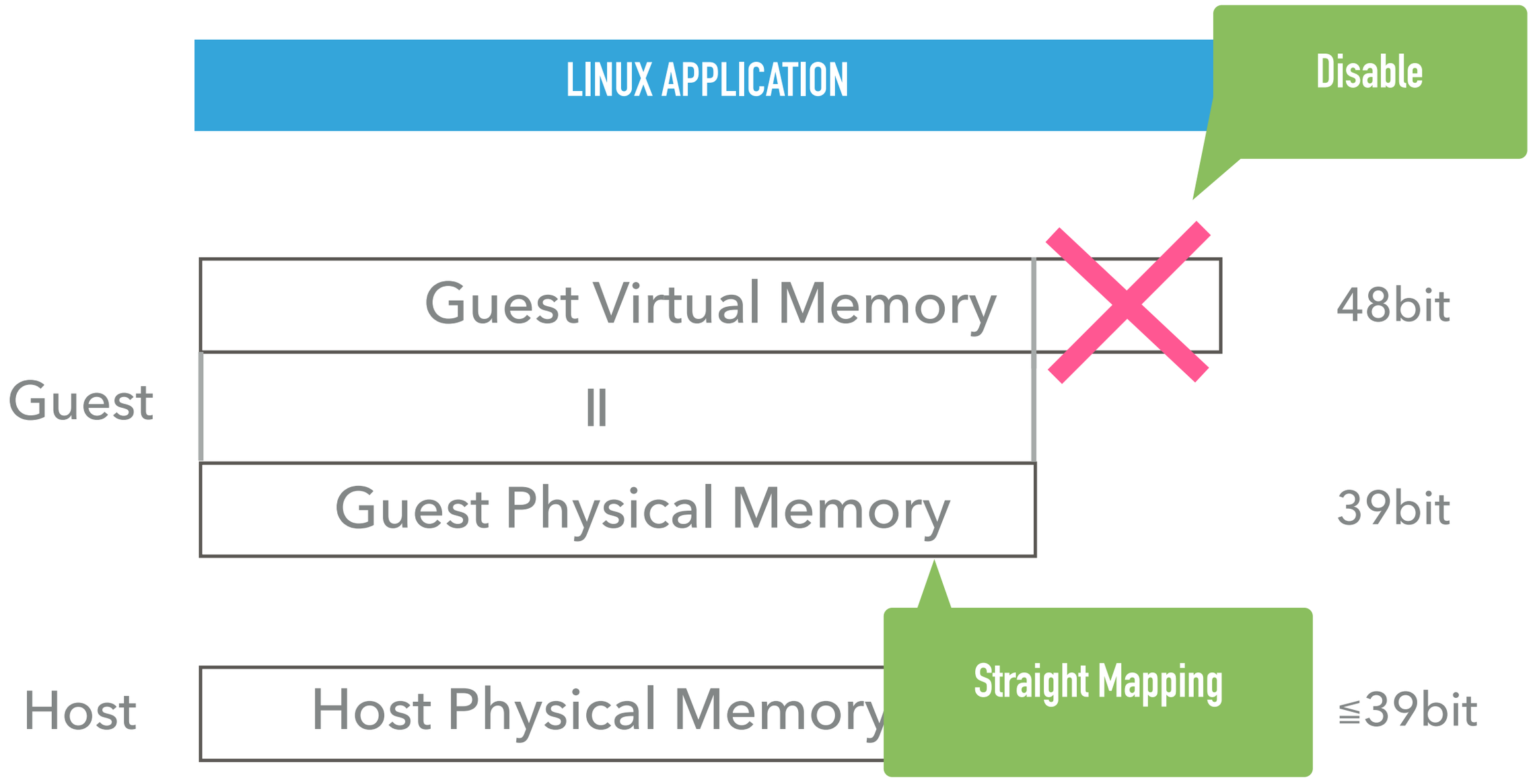
Memory Translation



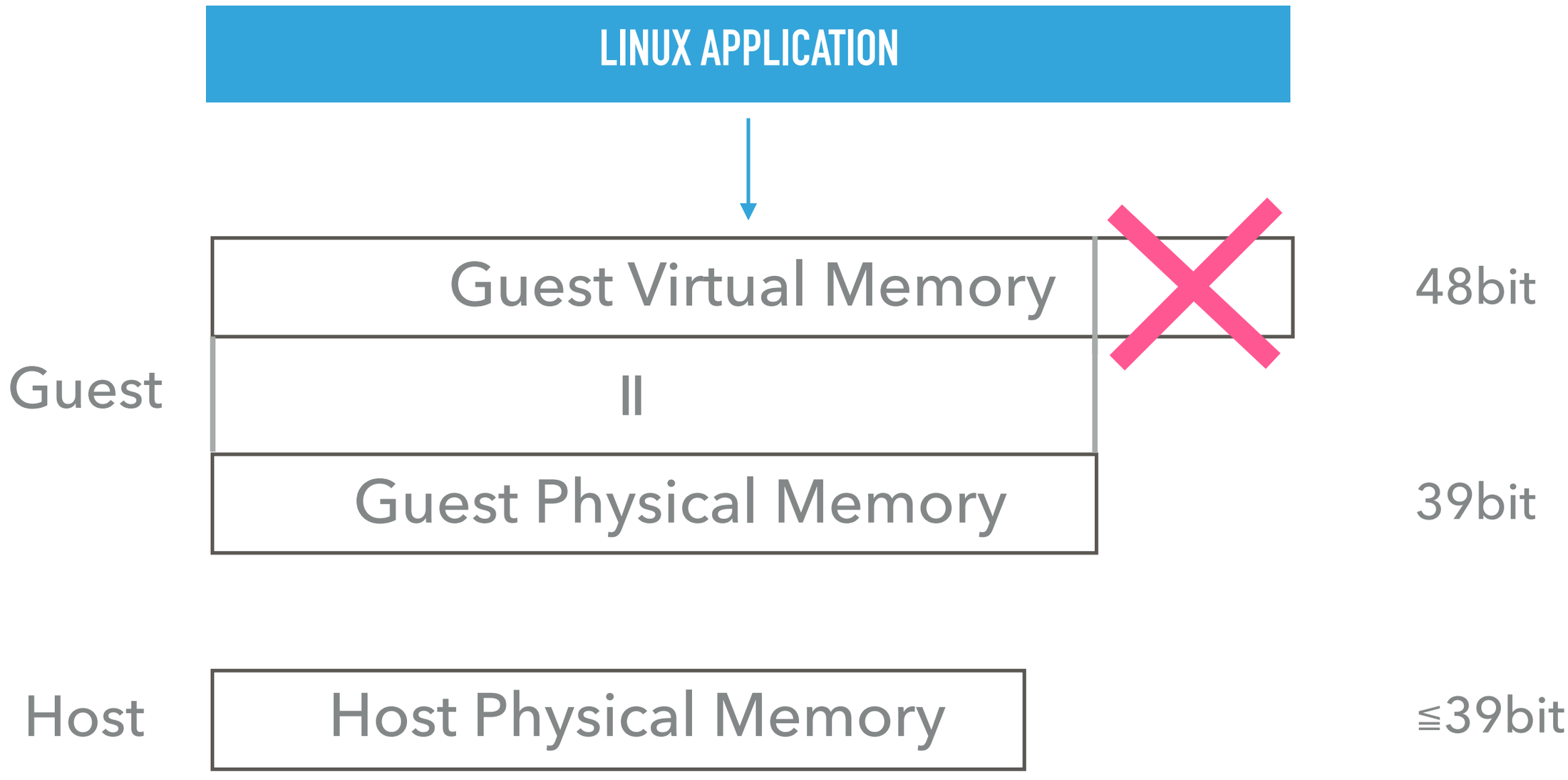
Memory Translation



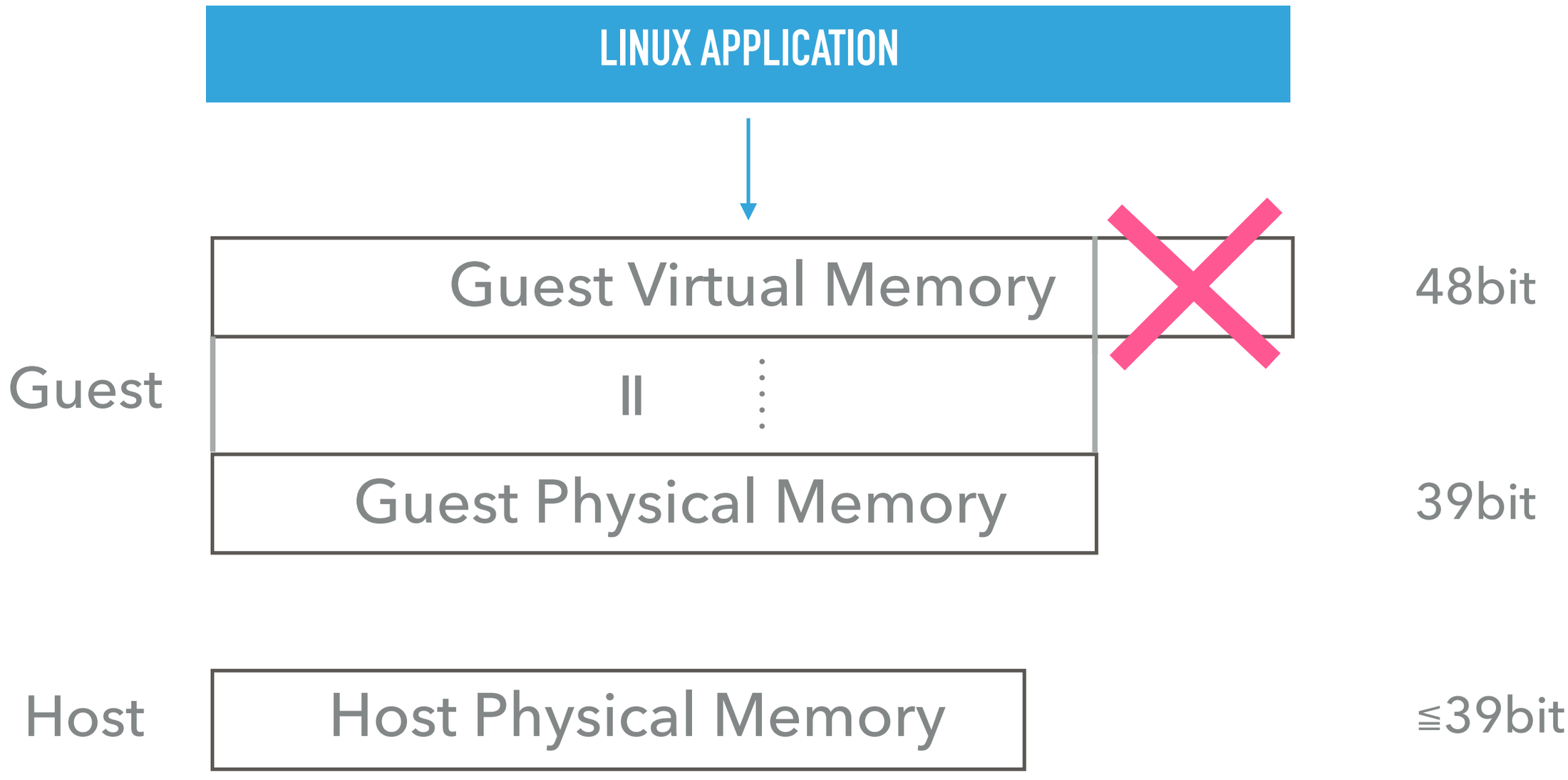
Memory Translation



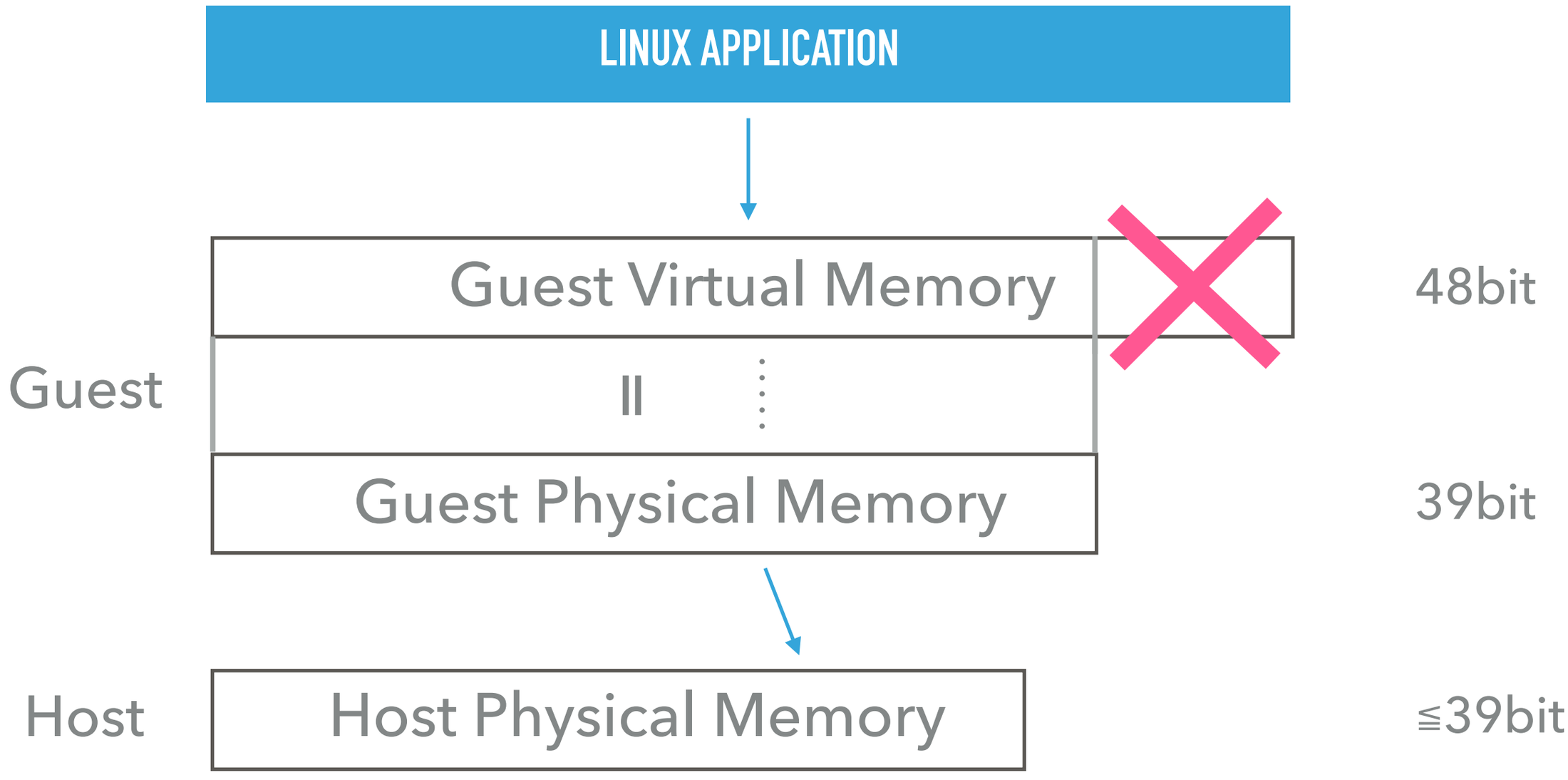
Memory Translation



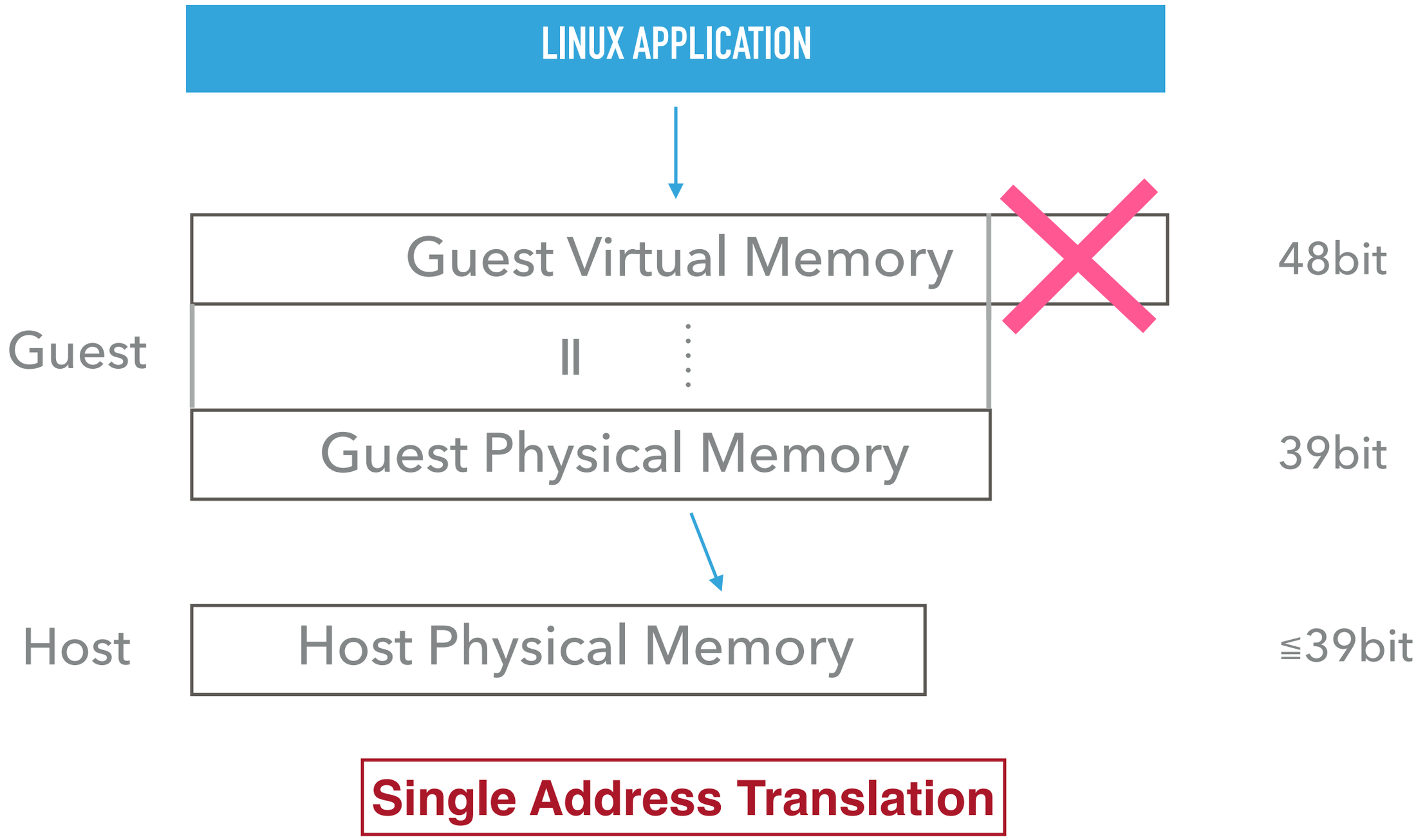
Memory Translation



Memory Translation



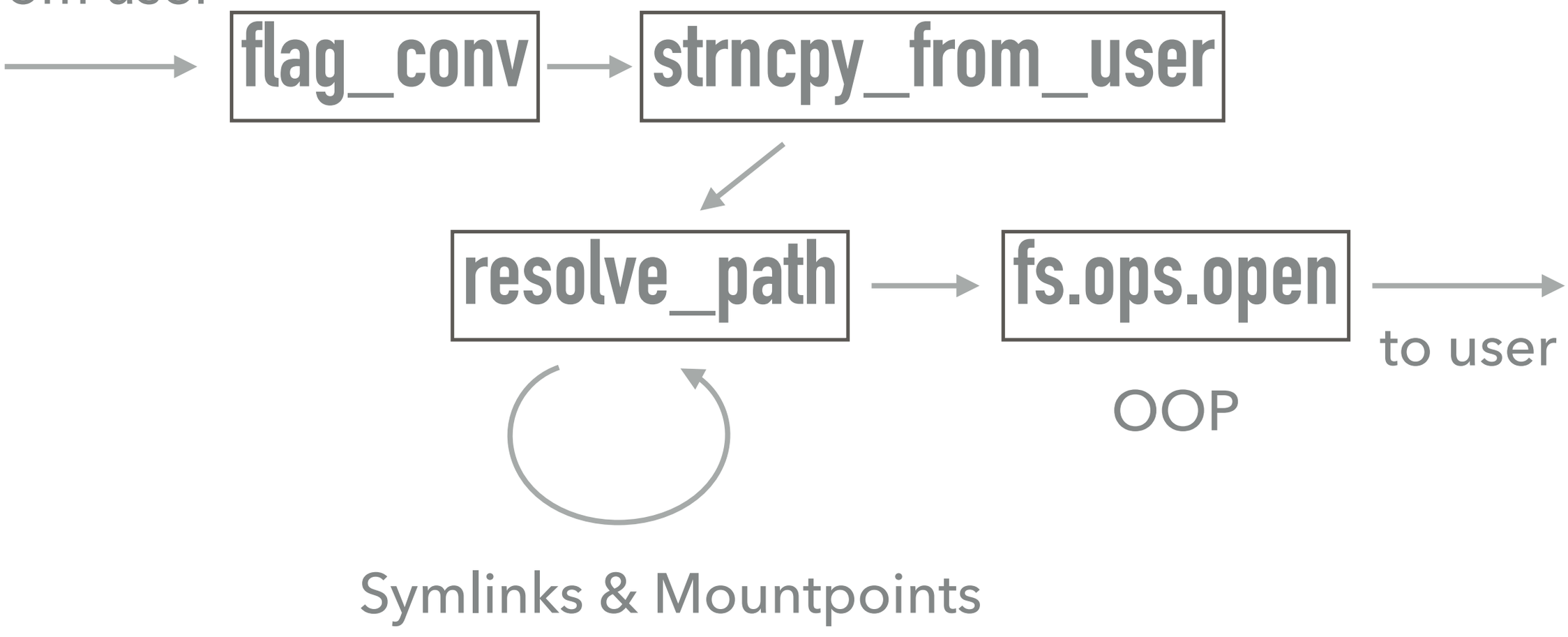
Memory Translation



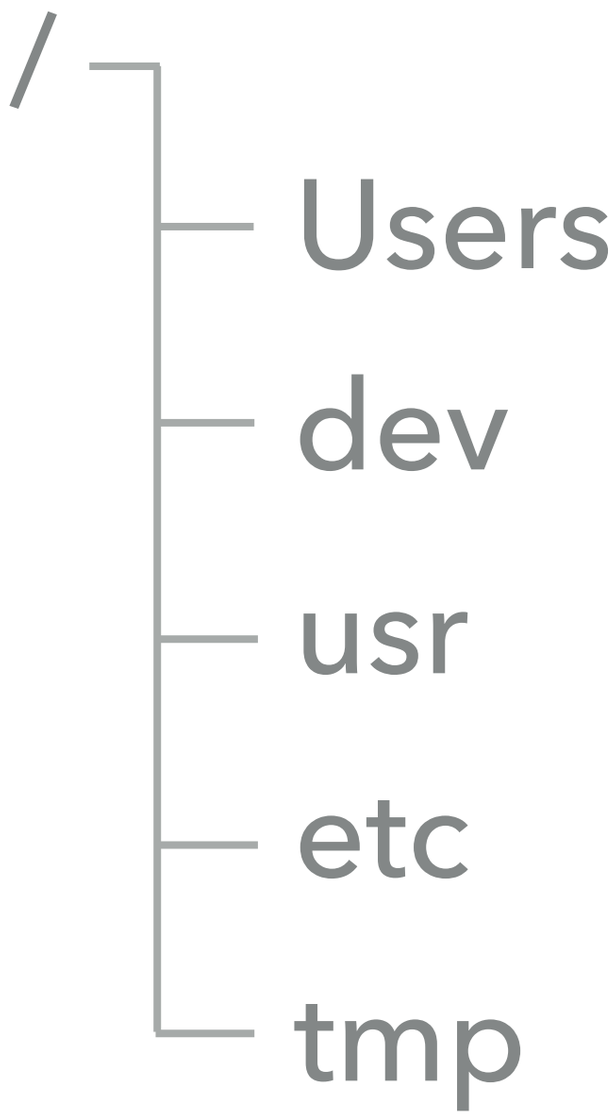
Virtual File System

Virtual File System

- open system call

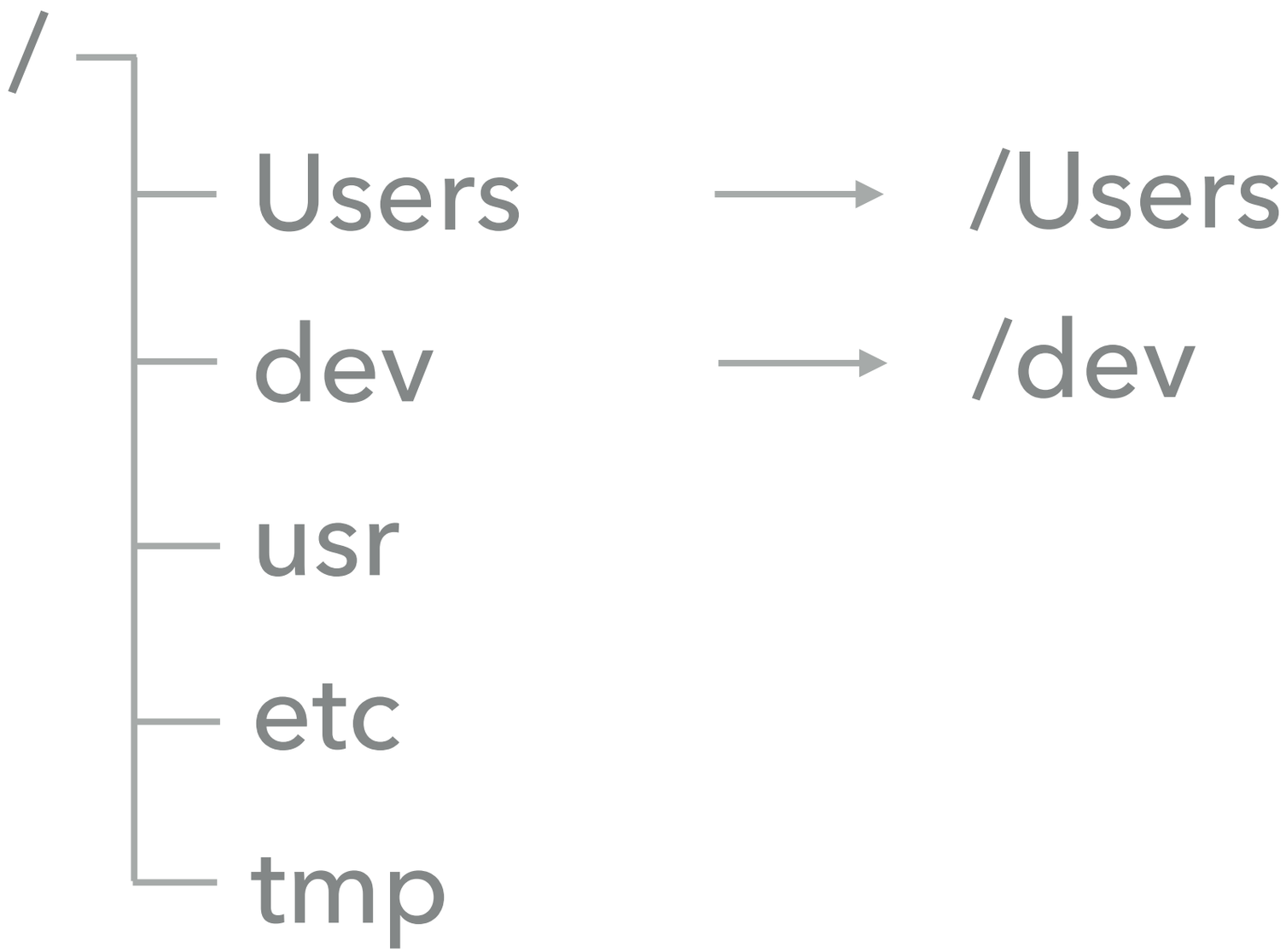


Virtual File System



Virtual File System

Virtual File System



Virtual File System

Virtual File System

Virtual File System

Other System Calls

Just call macOS's one

getuid　　getpid

getgid　　getpgid

alarm　　time

semget

Need conversion

futex	emulate with conditional value
socket	integrate with VFS
sigaction	create signal frame inside VM
gettid	generate from thread-id

Agenda

- ~~What is Noah~~
- ~~Background~~
- ~~Noah in Detail~~
  - ~~Architecture Overview~~
  - ~~Advantages of Noah Architecture~~
  - ~~Subsystem Implementation~~
    - ~~Memory management, VFS, and the other~~
- Current Implementation Status and Performance
- **Related Technologies and Comparison**
  (Windows Subsystem for Linux, Linuxulator, and so on)
- Their Possible Impact on Cross Platform Development

Current Implementation Status of Noah

Current Implementation Status of Noah

- Still in development
- Currently capable of running
  - apt-get, pacman (Not all subcommands are supported yet)
  - vim, gcc, make
  - Ruby
  - Binutils, ls, cat, …
  - X applications; xeyes, xclock, xfwrite, doom 3, …
  - sudo, curl, nc, man, …
- The most easiest way to build Linux kernel on macOS is to use Noah!

Performance

- Performance data will be made public in the presentation since it contains unpublished materials

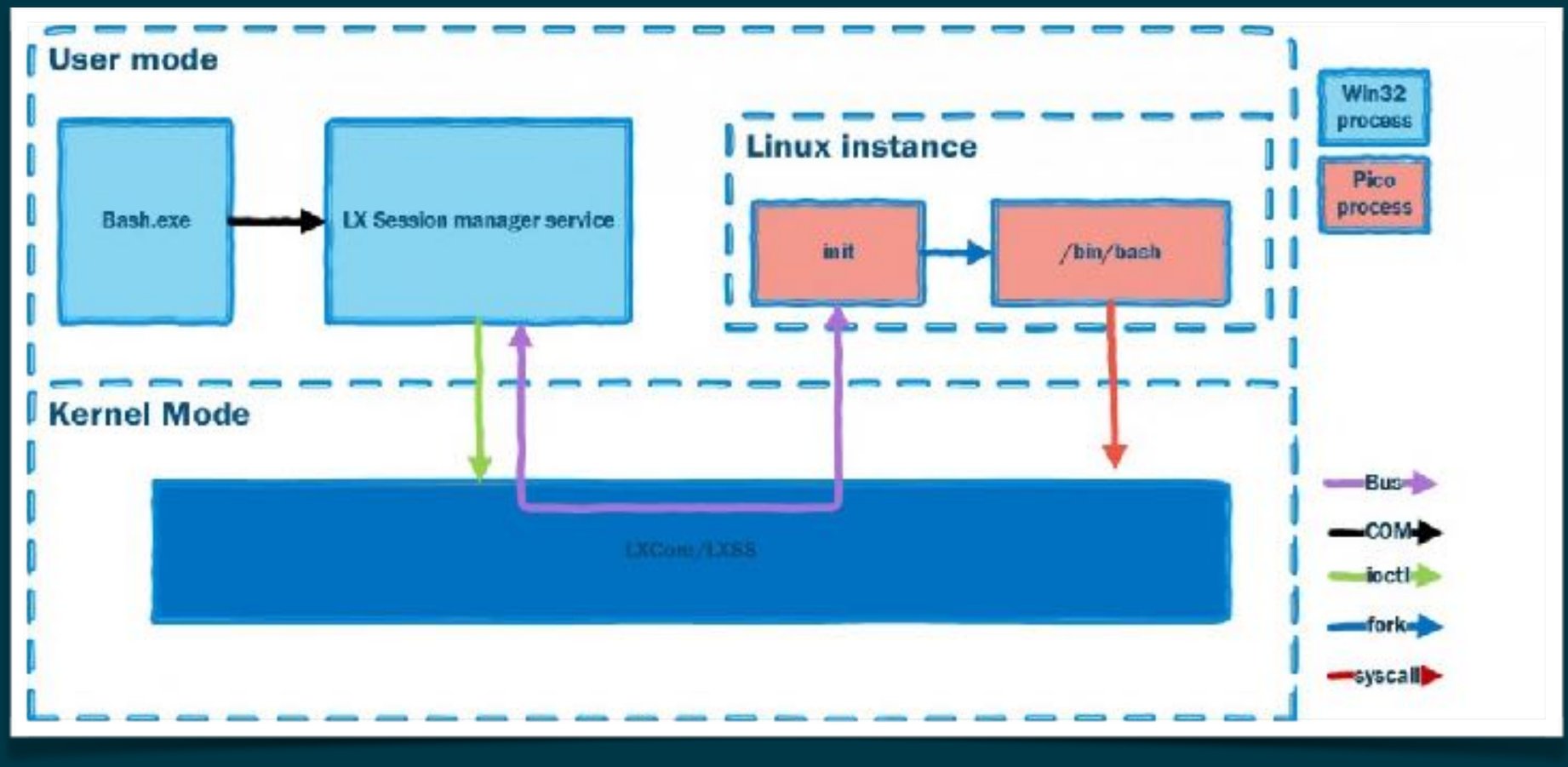
Related Technologies

Linux Compatibility Layers

- OS Built-in
  - Windows Subsystem for Linux
    - FreeBSD’s Linuxulator
- Third Party Middleware
  - Foreign LINUX

Windows Subsystem for Linux (WSL)

- Built-in Linux compatibility layer for Windows 10 by Microsoft
- Picoprocess contains a Linux ELF binary. Kernel drivers (LXCore and LXSS) handle system calls from it in kernel mode



Quoted from https://blogs.msdn.microsoft.com/wsl/2016/04/22/windows-subsystem-for-linux-overview/

Linuxulator

- Built-in Linux compatibility layer for FreeBSD
- FreeBSD has a loader for Linux ELF and implementations of Linux system calls
- Similar approach to WSL's LXSS / LXCore (Linuxulator is older, though)

Foreign Linux

- https://github.com/wishstudio/flinux

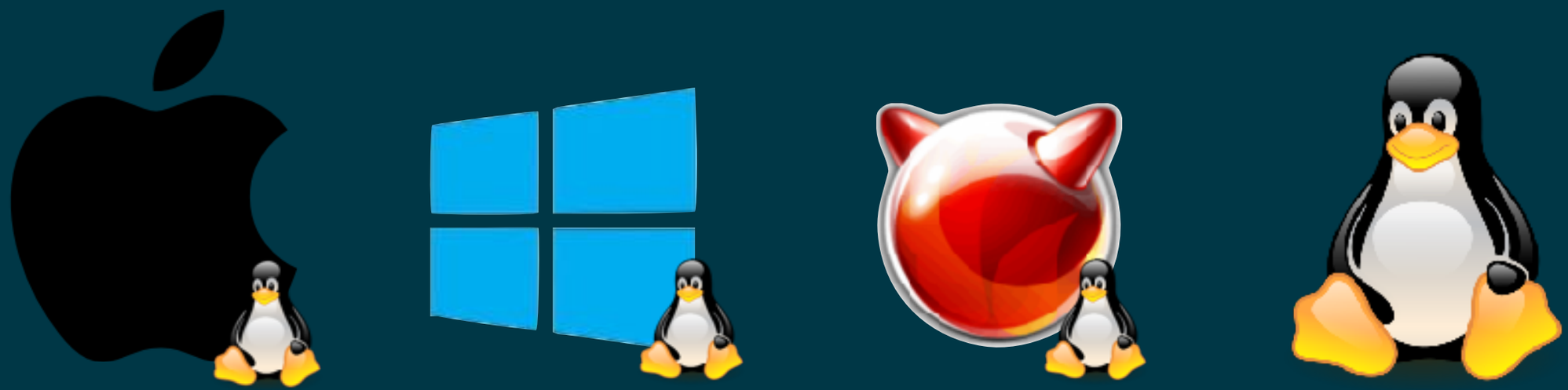
- Run unmodified Linux applications on Windows by dynamic binary translation and system call emulation

- It seems that the overhead of dynamic binary translation is a bit heavy, however…

Comparison

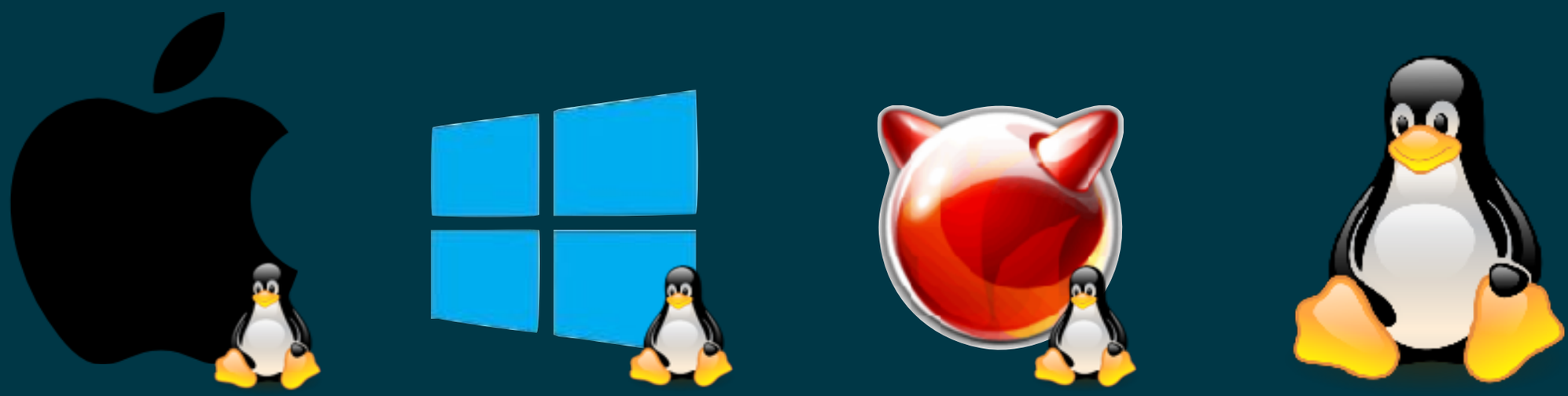
	OS	Binary Compatibility	Portable	No Kernel Modification	Smooth Interaction	Reasonable Overhead
Noah	macOS	✔	✔	✔	✔	✔
WSL	Windows	✔	✗	✗	✔	✔
Linuxulator	FreeBSD	✔	✗	✗	✔	✔
Foreign LINUX	Windows	✔	✔	✔	✔	✗
Full VM	Any	✔	✔	✔	✗	✔ *with processor hardware virtualization

Future Cross Platform Development



Now major four operating systems have Linux compatibility

- Linux could be regarded as “standard” like POSIX
- In the future, once you write a Linux application, it simply could run on any platforms
- Noah could help it!



Summary

- We introduced Noah, which is a middleware that runs unmodified Linux ELF applications on macOS
- Noah adopts a new hypervisor based approach with many merits
- Now four major operating systems have Linux compatibility. In the future, your Linux applications could run anywhere!