# EECS 482
# Introduction to Operating Systems

## Winter 2018

Baris Kasikci

(Thanks to Harsha V. Madhyastha
for the slides!)

# Address space protection

- Goal: Any process must be unable to access any other process's address space
- Approach:
  - All reads/writes to physical memory are translated based on page table entries
  - Enable protection by ensuring physical pages pointed to by P1's PTEs are disjoint from P2's PTEs
- Requirement: A process must be unable to access the entries in any page table

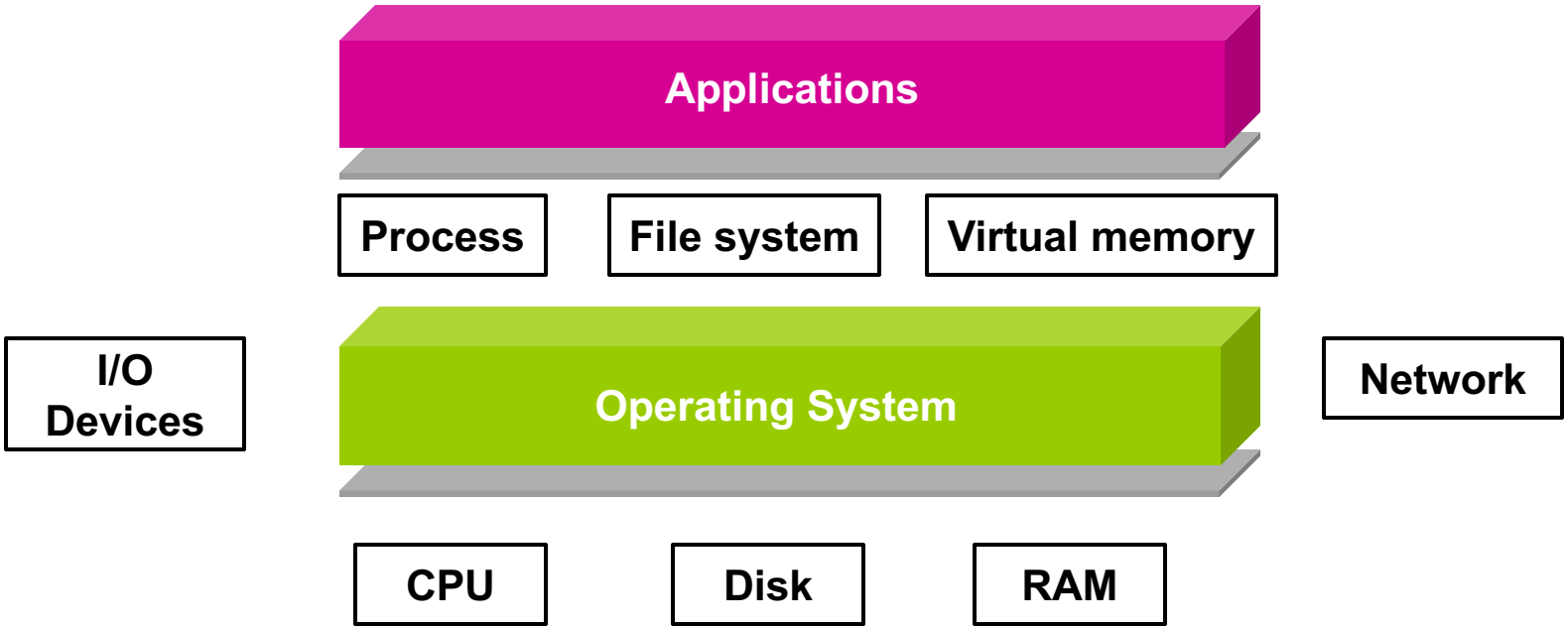
# Page table protection

- Approach:
  - Modifying page table is privileged operation
  - Operation permitted only if mode bit is set
- How to protect mode bit?
  - Set mode bit upon transitions into kernel space (interrupts, faults, and system calls)
  - Transitions defined in interrupt vector table
  - Interrupt vector table initialized by init process
- Protection boils down to control of system init

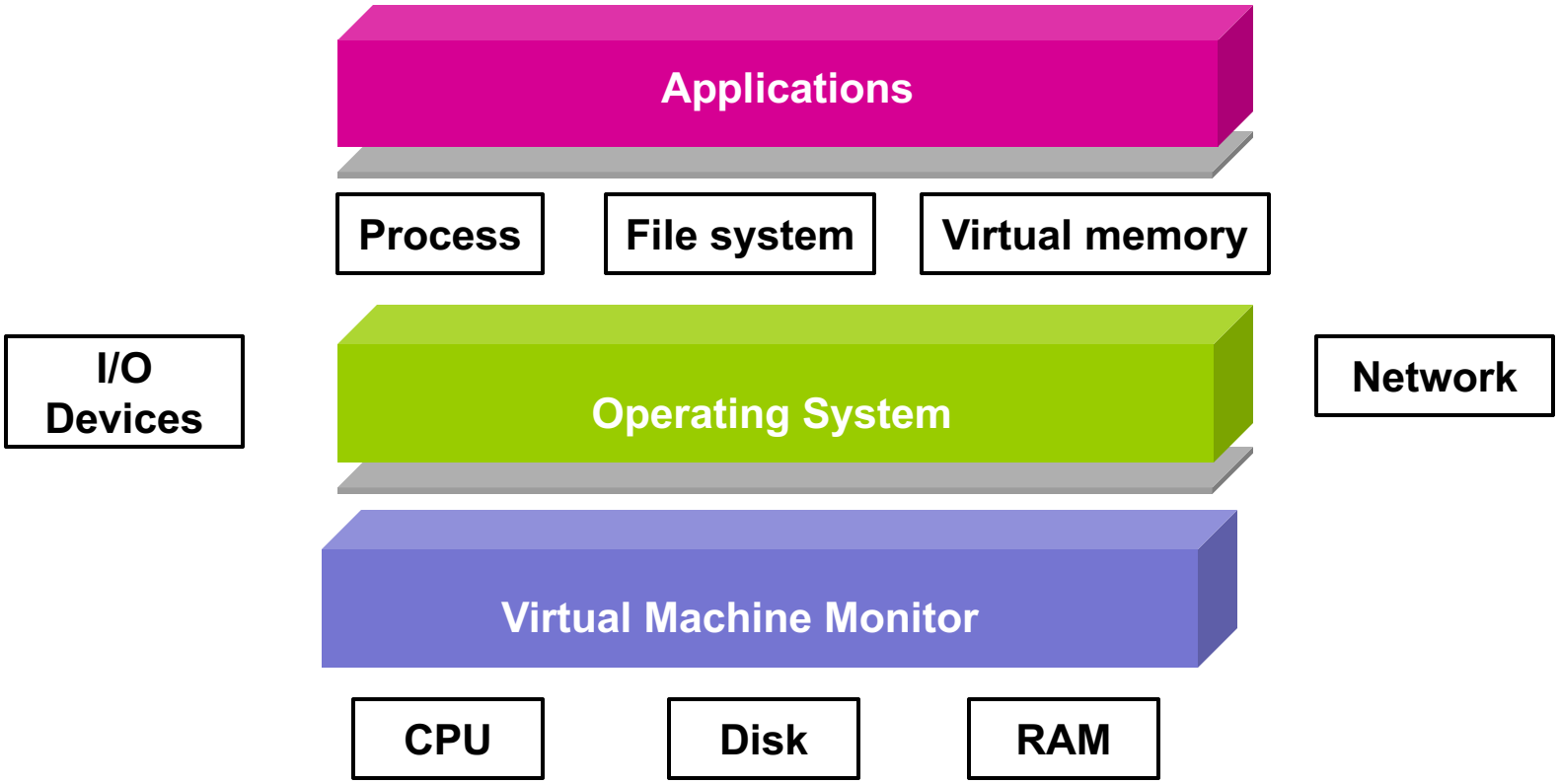
# Summary of OS concepts

- Primary jobs performed by OS:
  - Abstraction
  - Management of shared resources
- Concepts involved:
  - Indirection
  - Concurrency and atomicity
  - Protection
  - Naming
  - Reliability

# OS Abstractions



# Virtual Machine Monitor



# Virtual Machine Monitors

- Virtual Machine Monitors (VMMs) are a hot topic in industry and academia
  - Industry commitment
    - » Software: VMware, Xen, Microsoft Virtual PC
    - » Hardware: Intel VT, AMD-V
      - If Intel and AMD add it to their chips, you know it's serious…
  - Academia: lots of VMM-based projects and papers
- An old idea, actually: developed by IBM in 60s and 70s

# Virtual Machine Monitors

- Today
  - What is a VMM, what problems have to be solved, how to solve them
  - Survey some virtualization systems
  - Briefly outline cool things you can do with virtualization

# What is a VMM?

- OS offers illusion that each process is running on its own computer
- A VMM virtualizes an entire physical machine
  - VMM offers illusion that OS has full control over the hardware
  - VMM "applications" (OSes) run in virtual machines
- Implication: Can run multiple instances of different OSes simultaneously on a machine

# Why do such a crazy thing?

- Resource utilization
  - Machines today are powerful, multiplex their hardware
    - » Example: Cloud services
  - Migrate VMs across machines without shutdown
- Software use and development
  - Can run multiple OSes simultaneously
    - » No need to dual boot
  - Can do system (e.g., OS) development at user-level
- Many other cool applications
  - Debugging, emulation, security, fault tolerance, …

# Example of Cool VMM Tricks

- How to experiment with apps, protocols, and systems on future hardware?
- Time dilation
  - VMM slows timer interrupt to make other hardware (CPU, disk, network) appear faster to OS and apps
  - Example:
    - » OS reads 10 Gb of data from network in 1 second, but thinks only 0.1 second has elapsed, giving illusion of 100 Gbps of bandwidth
    - » But, applications run 10x slower

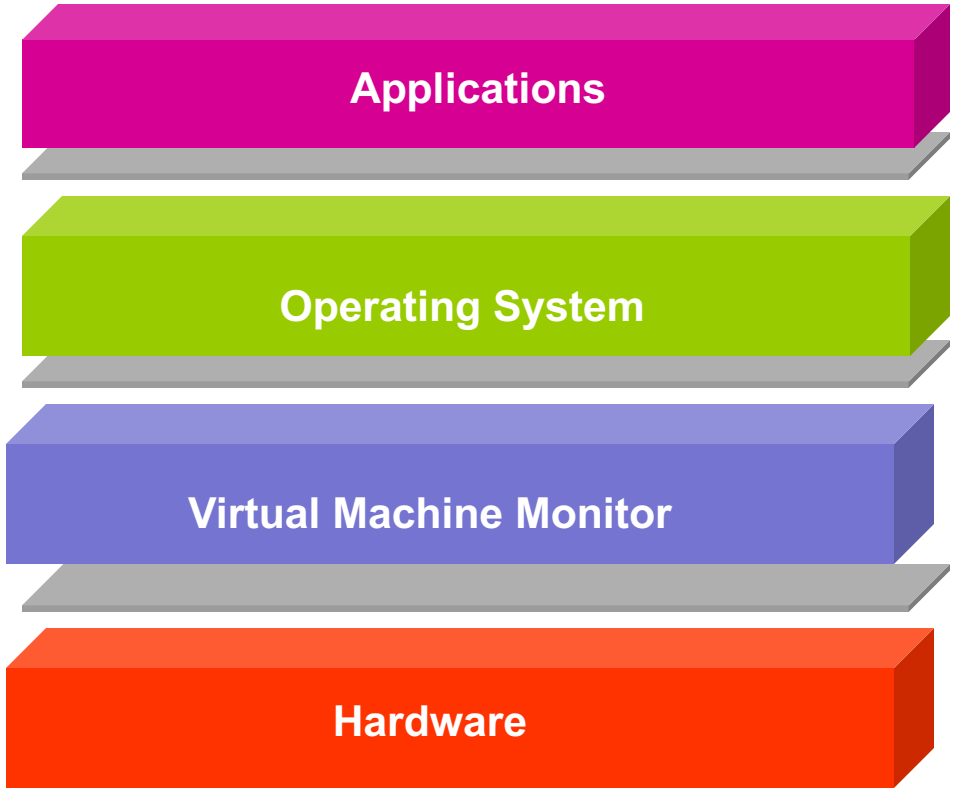
# VMM Requirements

- Fidelity
  - OSes and applications work without modification
    - » (although we may modify the OS a bit)
- Isolation
  - VMM protects resources and VMs from each other
- Performance
  - VMM is another layer of software → overhead
    - » As with OS, want to minimize this overhead

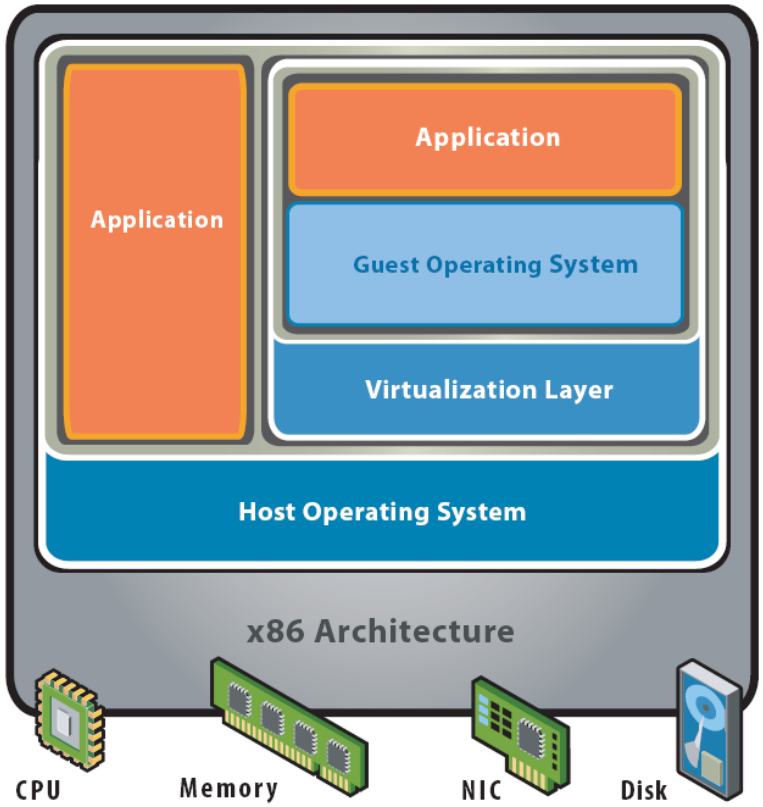
# VMware

- VMware workstation uses hosted model
  - VMM runs unprivileged, installed on base OS
  - Relies upon base OS for device functionality
- VMware ESX server uses hypervisor model
  - VMM runs directly on hardware
- VMware uses software virtualization
  - Dynamic binary rewriting translates code executed in VM
  - Rewrite only privileged instructions to reduce overhead

# Vmware Hypervisor Model



# VMware Hosted Architecture

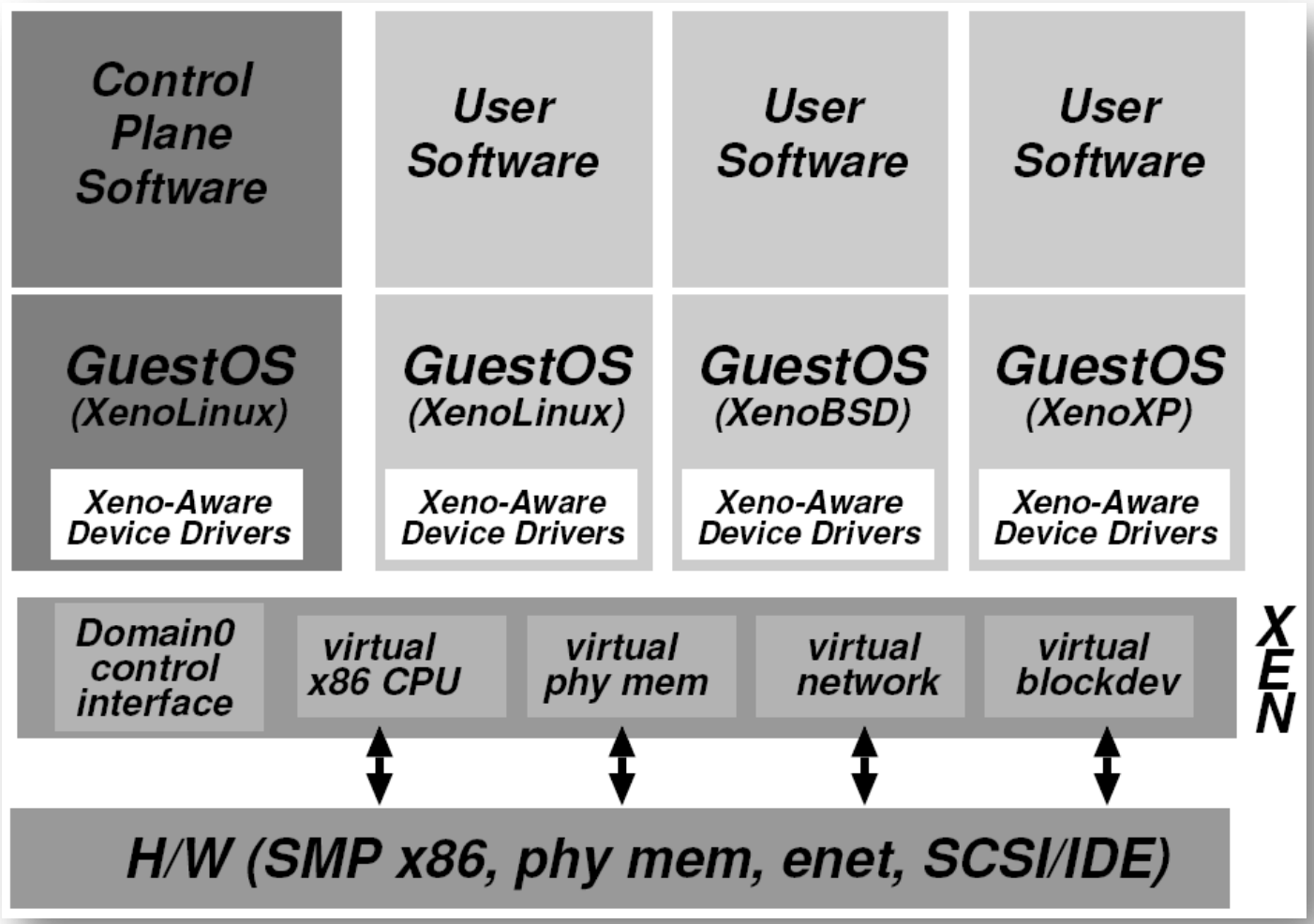



Hosted Architecture

# Xen

- Early versions use "paravirtualization"
  - Fancy word for "we have to modify & recompile OS"
- Xen hypervisor (VMM)
  - VMM runs at privilege, VMs (domains) run unprivileged
  - Trusted OS (Linux) runs in own domain (Domain0), manages privileged operations
- Most recent version does not require OS mods
  - Because of Intel/AMD hardware support
- Commercialized via XenSource, but also open source

# Xen Architecture



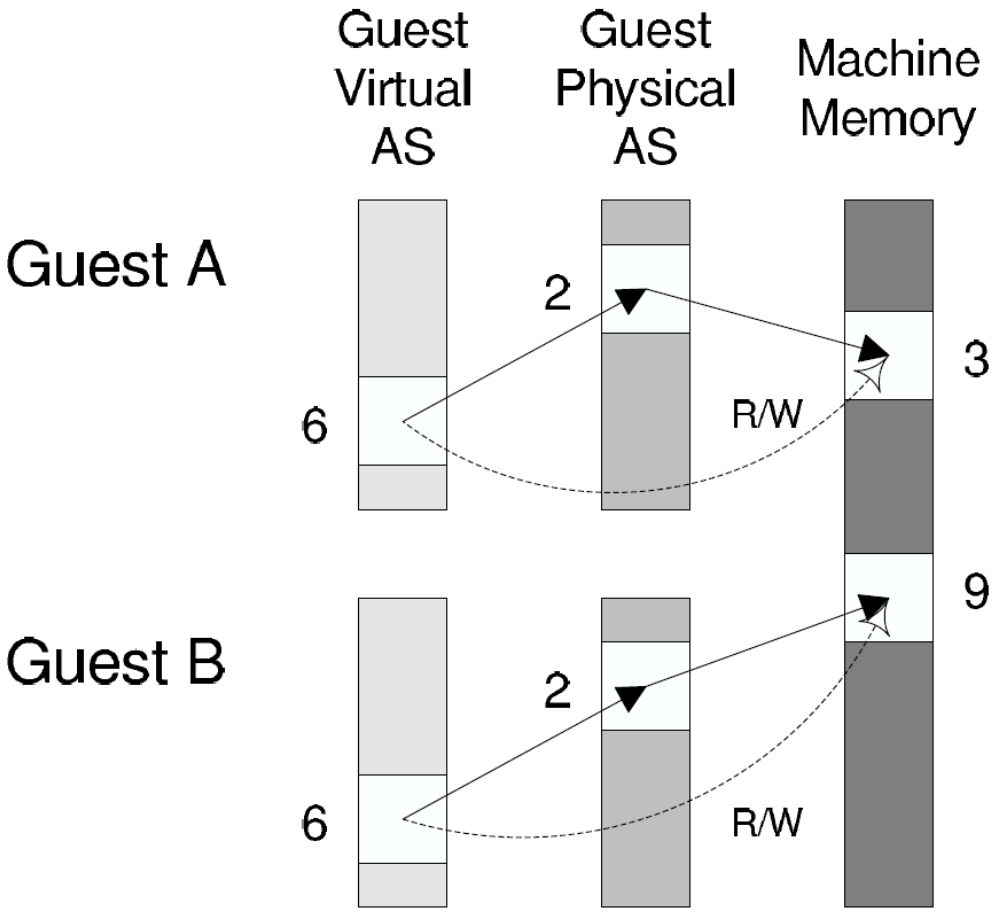
# What needs to be virtualized?

- Exactly what you would expect
  - CPU
  - Events
  - Memory
  - I/O devices
- Isn't this just duplicating OS functionality?
  - Yes and no
  - Approaches will be similar to what OS does
    - » Simpler functionality (VMM much smaller than OS)
  - But implements a different abstraction
    - » Hardware interface vs. OS interface

# Virtualizing Memory

- OS assumes full control over memory
- But VMM partitions memory among VMs
  - VMM needs to control mappings for isolation
    - » OS can only map to a physical page given to it by VMM
- Solution: Need MMU support to handle two-levels of page tables

# Shadow Page Tables



# Hardware Support

- Intel and AMD implement virtualization support in their latest x86 chips (Intel VT-x, AMD-V)
  - Goal is to fully virtualize architecture
  - Transparent trap-and-emulate approach now feasible
  - Echoes hardware support originally implemented by IBM
- Execution model
  - New execution mode: guest mode
    - » Direct execution of guest OS code, including privileged insts
  - Virtual machine control block (VMCB)
    - » Controls what operations trap, records info to handle traps in VMM
  - New instruction `vmrun` enters guest mode, runs VM code
  - When VM traps, CPU executes new `exit` instruction
  - Enters VMM, which emulates operation

# Hardware Support (2)

- Memory
  - Intel extended page tables (EPT), AMD nested page tables (NPT)
  - Original page tables map virtual to (guest) physical pages
    - » Managed by OS in VM, backwards-compatible
    - » No need to trap to VMM when OS updates its page tables
  - New tables map physical to machine pages
    - » Managed by VMM
  - Tagged TLB w/ virtual process identifiers (VPIDs)

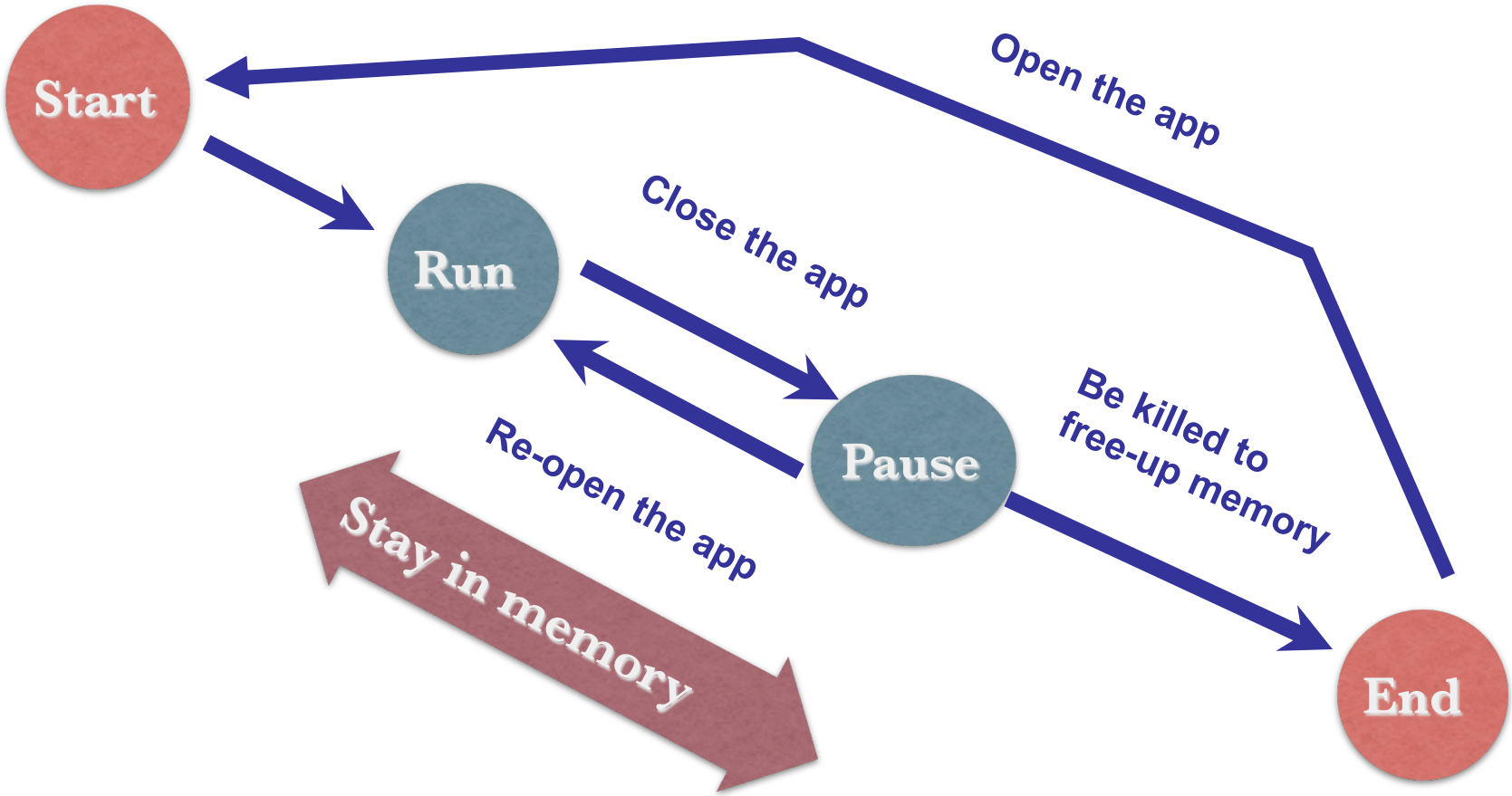
# Dalvik VM

- Dalvik is a Java Virtual Machine
  - Designed to run with limited system resources
- Android applications are written in Java
- Every application runs in its own JVM

# Process management

- What's the difference between mobile app cycle and desktop app cycle?
  - One app has focus from the user at a time
- Two key principles
  - Android usually does not delete an app's state even if you close it
  - Android kills apps when memory usage is too high, but it saves app state for quick restart later on

# Application Life cycle



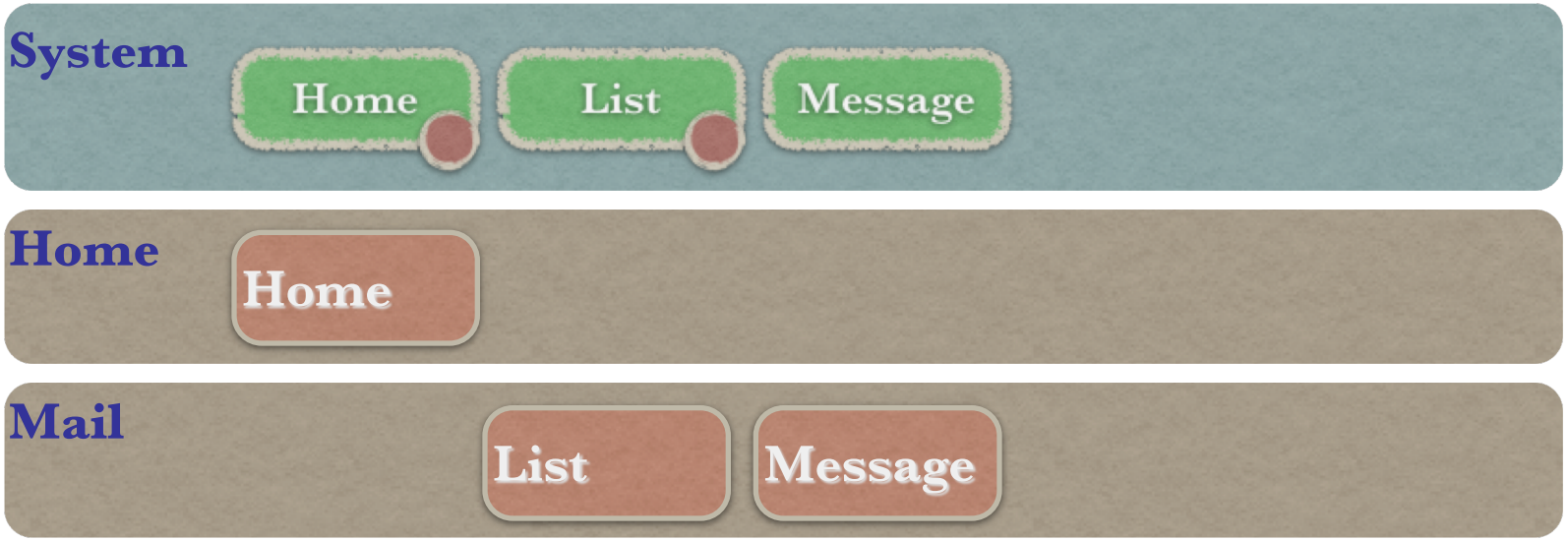
# Example


At the “Home” screen

# Example

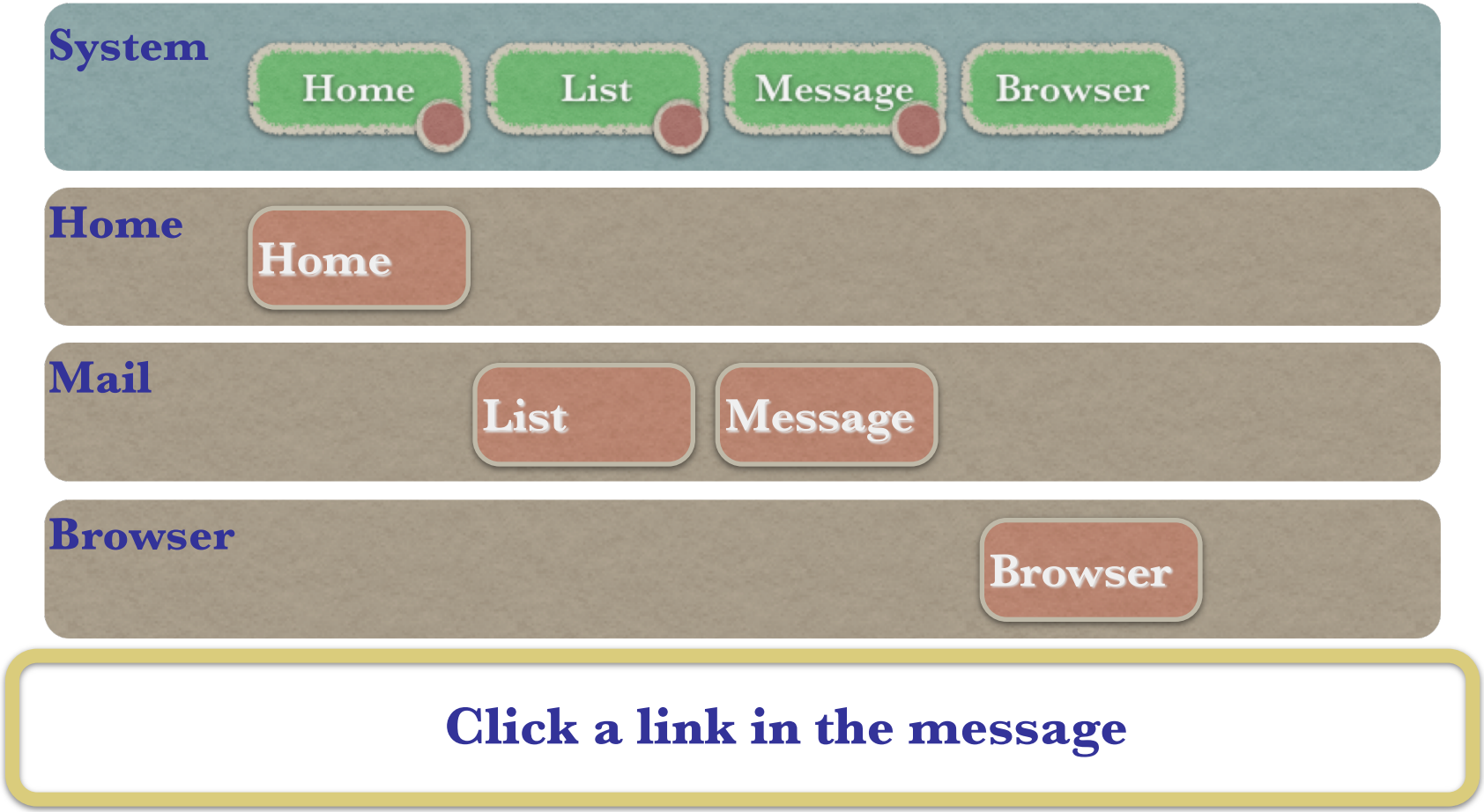

Start the “Mail” app and read the list

# Example

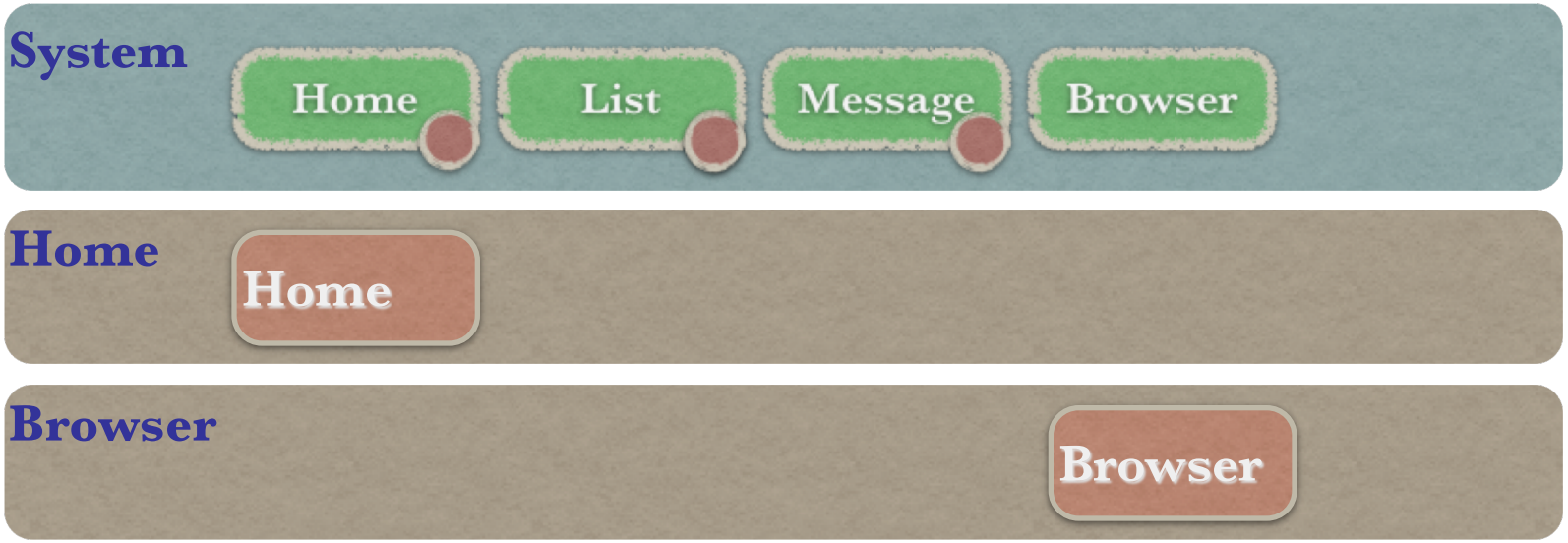



Click on one of the message and see its content

# Example

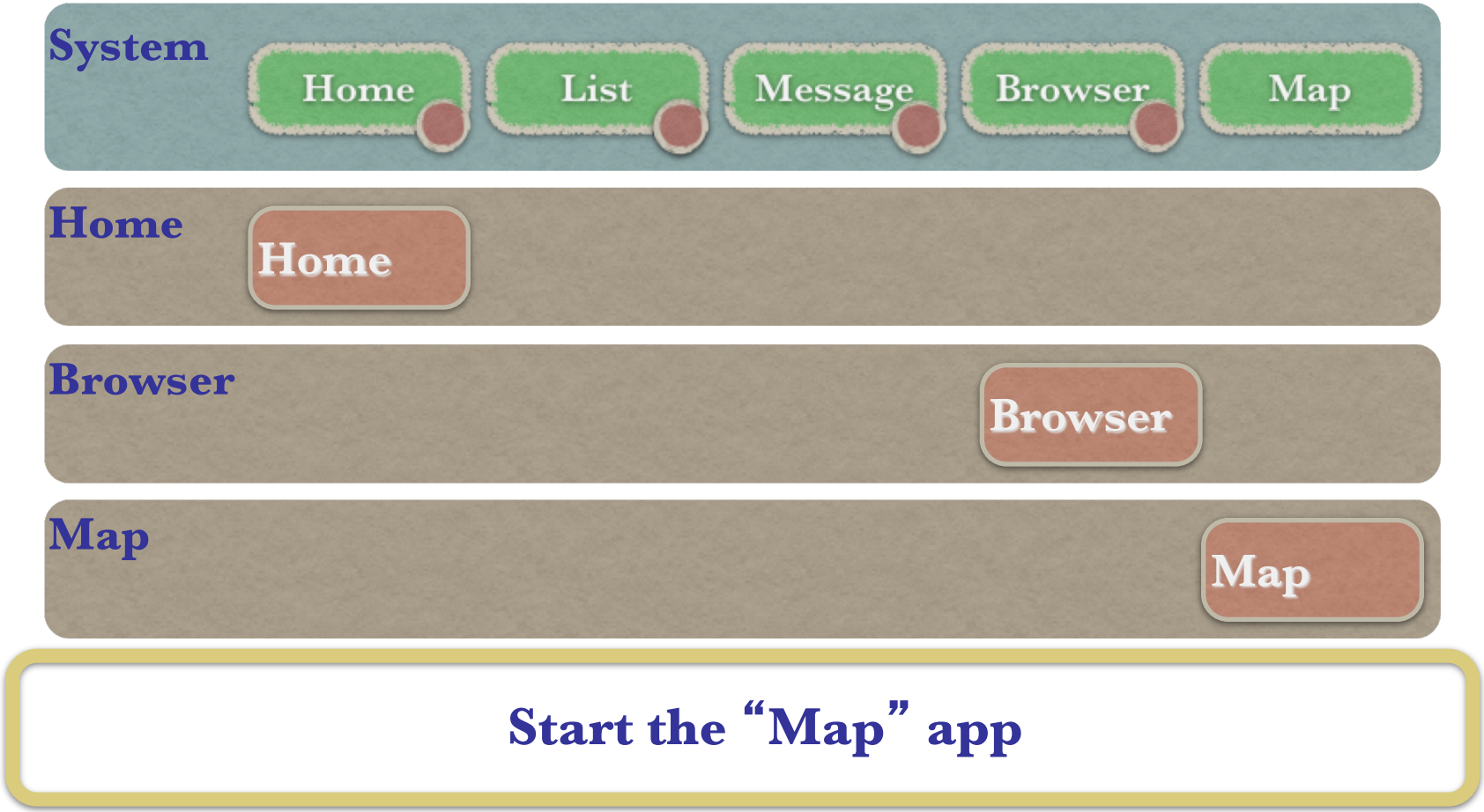

System
Home
List
Message
Browser
Home
Home
Mail
List
Message
Browser
Browser
Click a link in the message

# Example

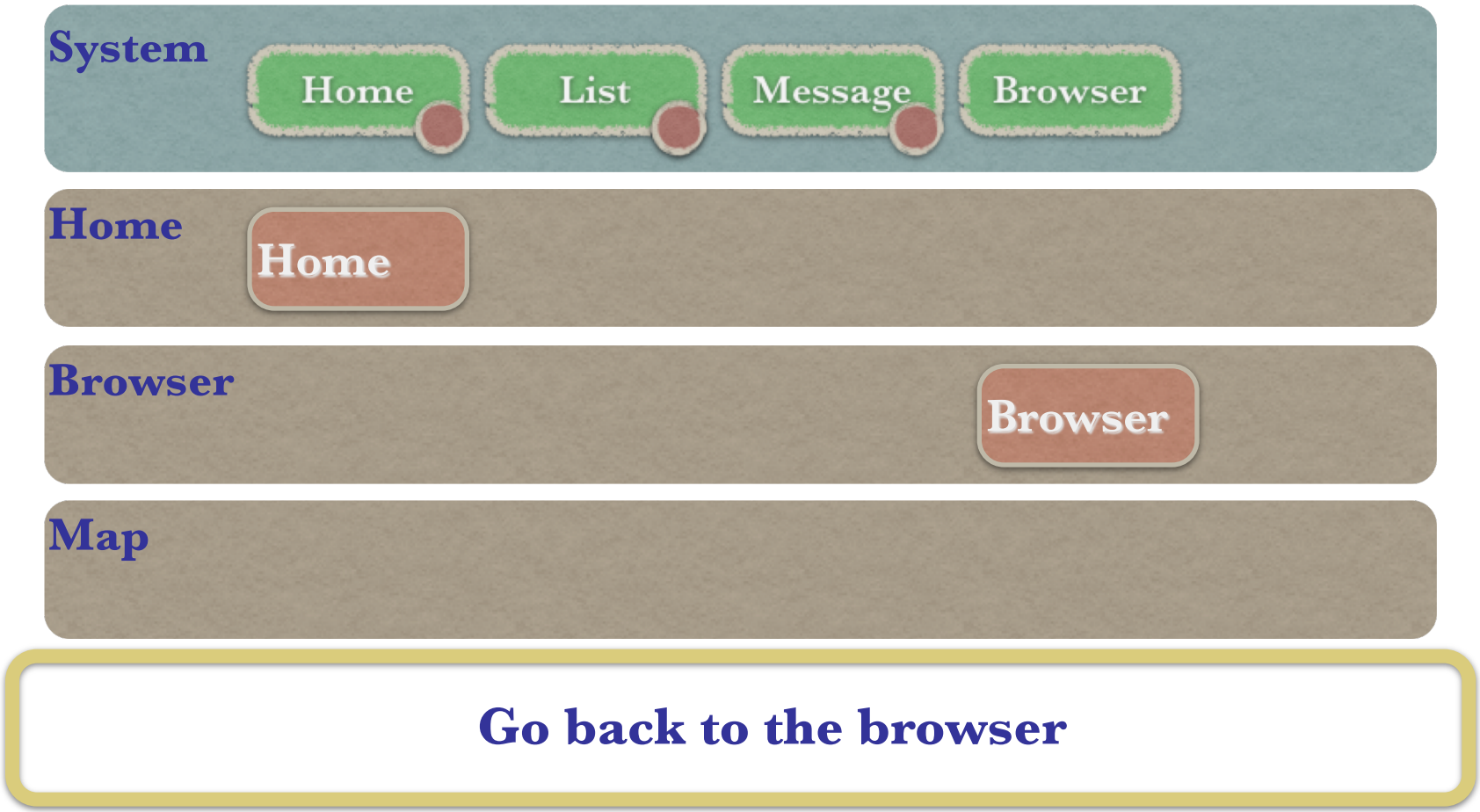



Now we have enough space to start the "Map" app

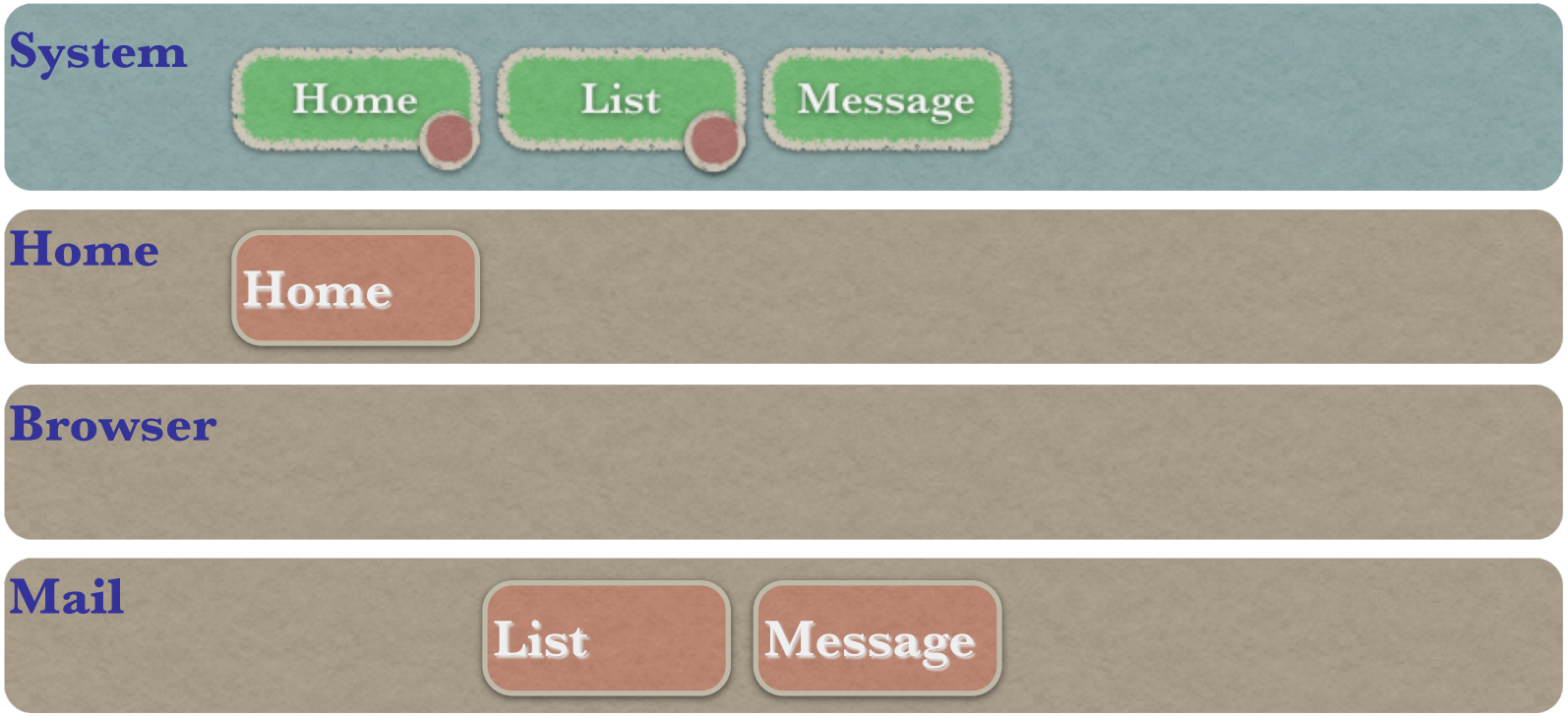
# Example


System
Home
List
Message
Browser
Map
Home
Home
Browser
Browser
Map
Map
Start the “Map” app

# Example

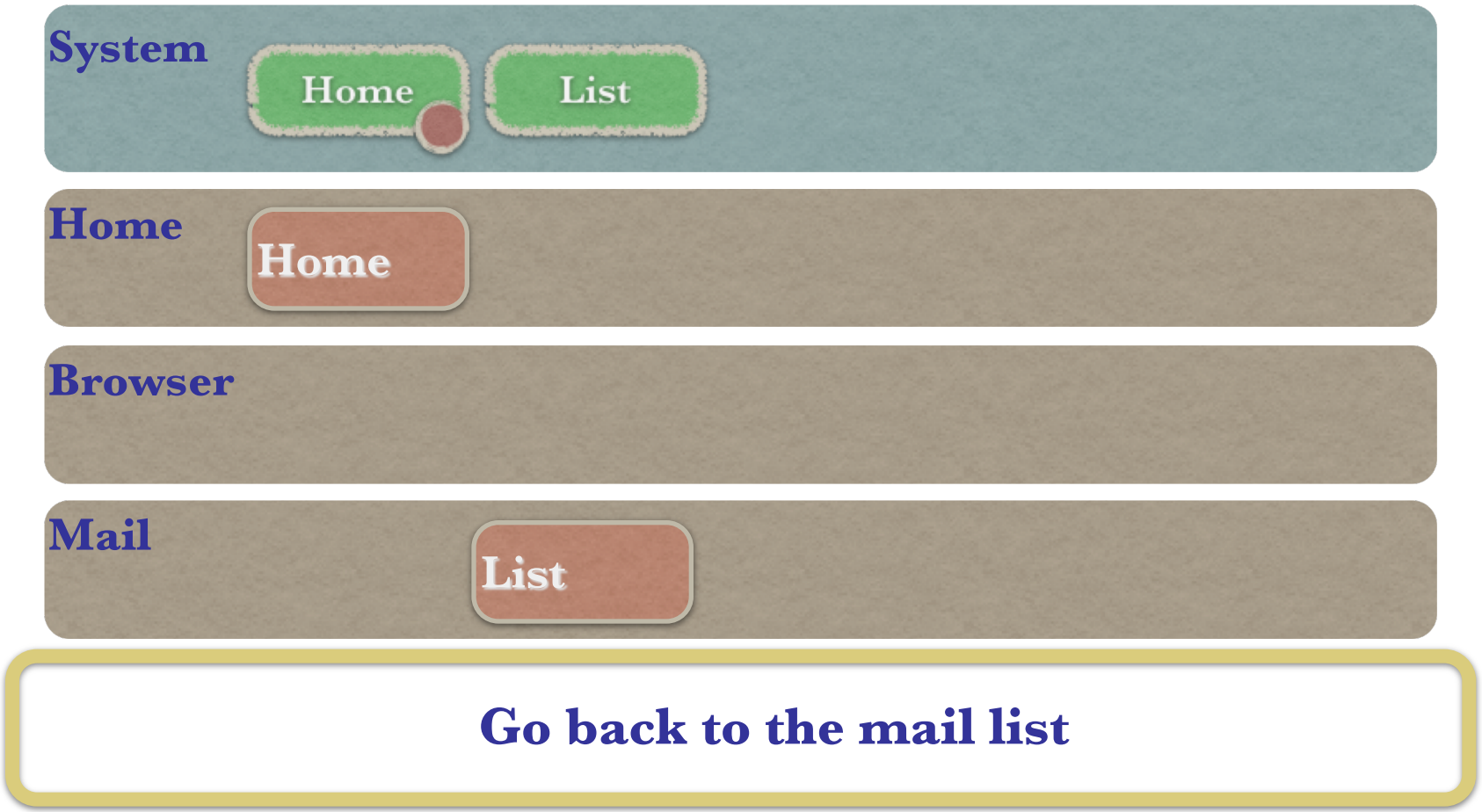

System
Home
List
Message
Browser
Home
Home
Browser
Browser
Map
Go back to the browser

# Example

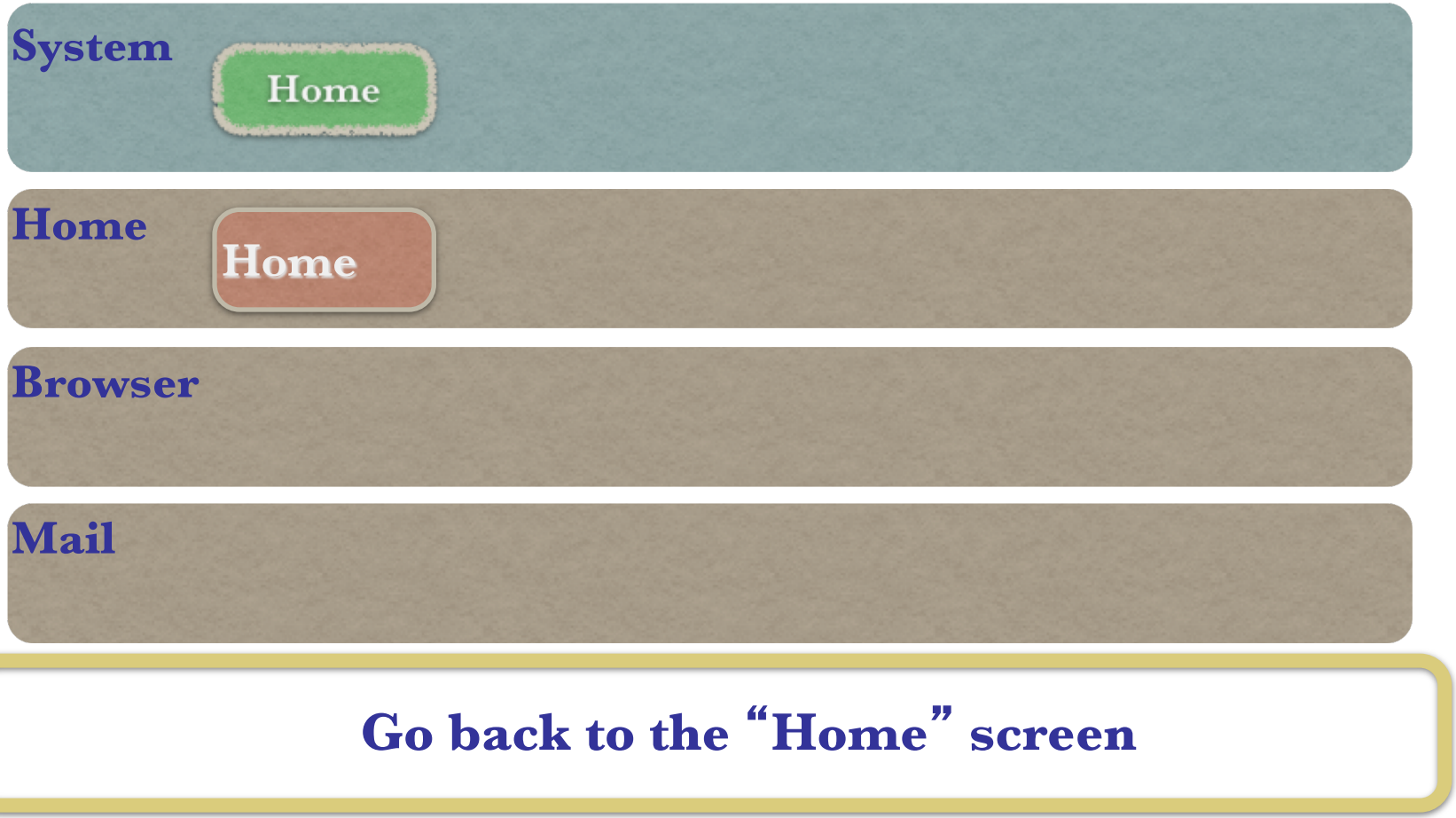



"Mail" app is resumed and shows previous message

# Example


System
Home
List
Home
Home
Browser
Mail
List
Go back to the mail list

# Example


System
Home
Home
Home
Browser
Mail
Go back to the “Home” screen

# Memory management

- Linux kernel does most of the job
- Page-based memory management
  - Virtual address to physical address mapping
- No virtual memory swapping
  - Pros and cons?

# Reminders

- Email me topics you'd like me to cover as part of the review next week