

Операційні системи

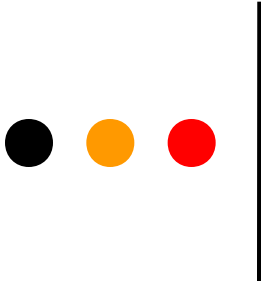
Лекція 3

Шляхи реалізації вимог до
сучасних операційних систем



План лекції

- Функціональні і ринкові вимоги до ОС
- Апаратна незалежність і здатність ОС до перенесення
- Розширюваність
- Програмна сумісність, прикладні програмні середовища
- Віртуалізація



Функціональні і ринкові вимоги до ОС

- *Функціональні* – вимоги до функцій, які підтримує ОС (вимоги користувача)
- *Ринкові* – вимоги до економічної ефективності розроблення і супроводження ОС (вимоги розробника)



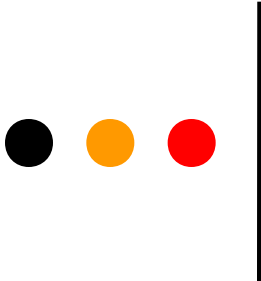
Функціональні вимоги до ОС

- Ефективне керування ресурсами
- Зручний інтерфейс користувача
- Зручний та ефективний інтерфейс прикладних програм
- Багатозадачність, багатопотоковість
- Віртуальна пам'ять
- Багатовіконний графічний інтерфейс
- Підтримка мережної взаємодії
- Надійність, відмовостійкість
- Безпека даних



Ринкові вимоги до ОС

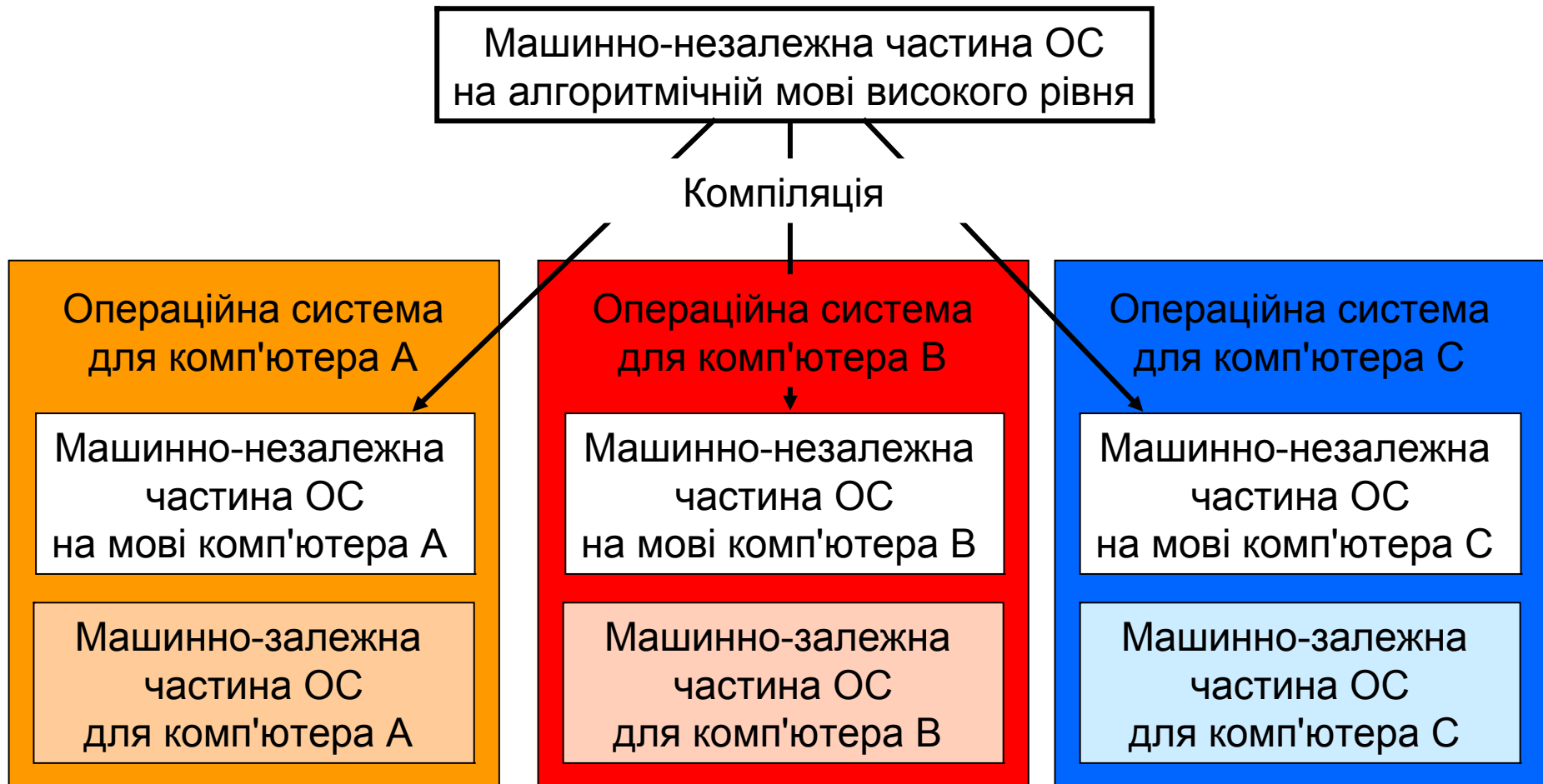
- Здатність до перенесення (*portability*)
- Програмна сумісність (у тому числі – зворотна сумісність)
- Розширюваність



Апаратна незалежність і здатність ОС до перенесення

- Засоби апаратної підтримки ОС
 - Система переривань
 - Засоби підтримки привілейованого режиму
 - Засоби трансляції адрес
 - Засоби перемикання процесів
 - Системний таймер
 - Засоби захисту оперативної пам'яті
 - Захист пристроїв введення-виведення
- Здатність до перенесення (*portability*)
 - Більша частина коду має бути написана мовою високого рівня, для якої існують транслятори на різних апаратних платформах
 - Код, що залежить від апаратного забезпечення, має бути відокремленим від іншої частини системи
 - Обсяг машинно-залежного коду має бути мінімізованим

Апаратна незалежність і здатність ОС до перенесення





Розширюваність ОС

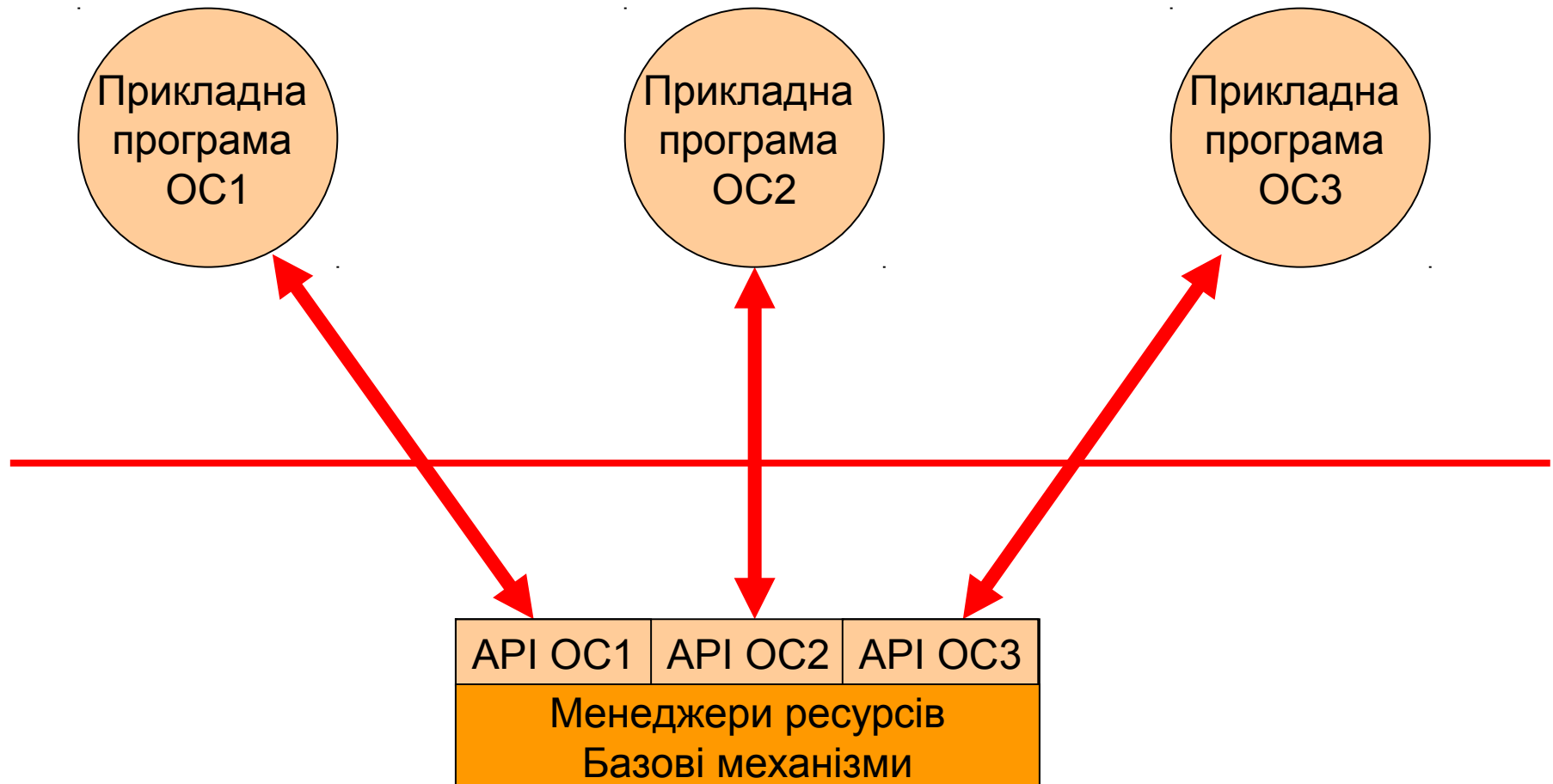
- ОС може жити довше за апаратуру!
- *Розширюваність* – можливість додавання нових функцій при збереженні основної частини коду
 - Підтримка нової апаратури (CD-ROM, flash)
 - Зв'язок з мережами нових типів
 - Нові технології інтерфейсу користувача (GUI)
 - Нова апаратна архітектура (багатопроцесорність)
- Шляхи досягнення розширюваності:
 - Модульна структура ОС
 - Використання об'єктів
 - Технологія клієнт-сервер із застосуванням мікроядрової архітектури
 - Завантажувані модулі драйверів



Програмна сумісність

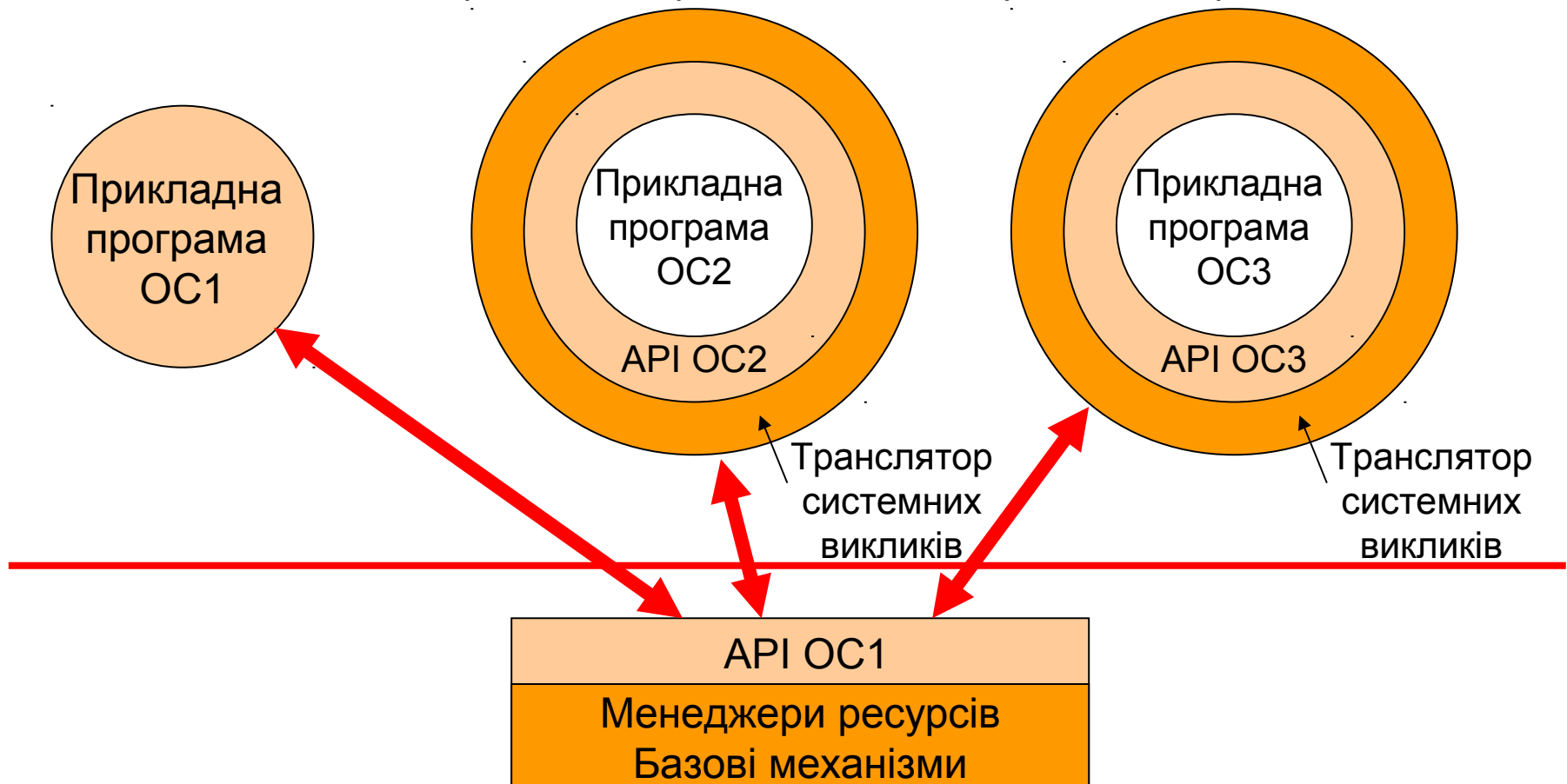
- *Програмна сумісність* – можливість виконувати у середовищі ОС програми, розроблені для іншої ОС
- *Зворотна сумісність* – можливість виконувати у середовищі ОС програми, розроблені для попередньої версії ОС
- *Сумісність вихідних текстів* – можливість перенесення вихідних текстів
 - Необхідна наявність компілятора (стандартизація мов програмування, розробка стандартних компіляторів)
 - Необхідна сумісність API (стандартизація інтерфейсів)
- *Бінарна сумісність* – можливість перенесення виконуваного коду
 - Якщо архітектура процесора (набір команд, система адресації, діапазон адрес) сумісна, тоді необхідні лише
 - сумісність API
 - сумісність внутрішньої структури виконуваного файлу
 - Якщо архітектури процесорів несумісні, то необхідна *емуляція середовища виконання*
 - Для прискорення емуляції – трансляція бібліотек

Реалізація рівноправних API



Реалізація прикладних програмних середовищ

Прикладне середовище ОС2 Прикладне середовище ОС3

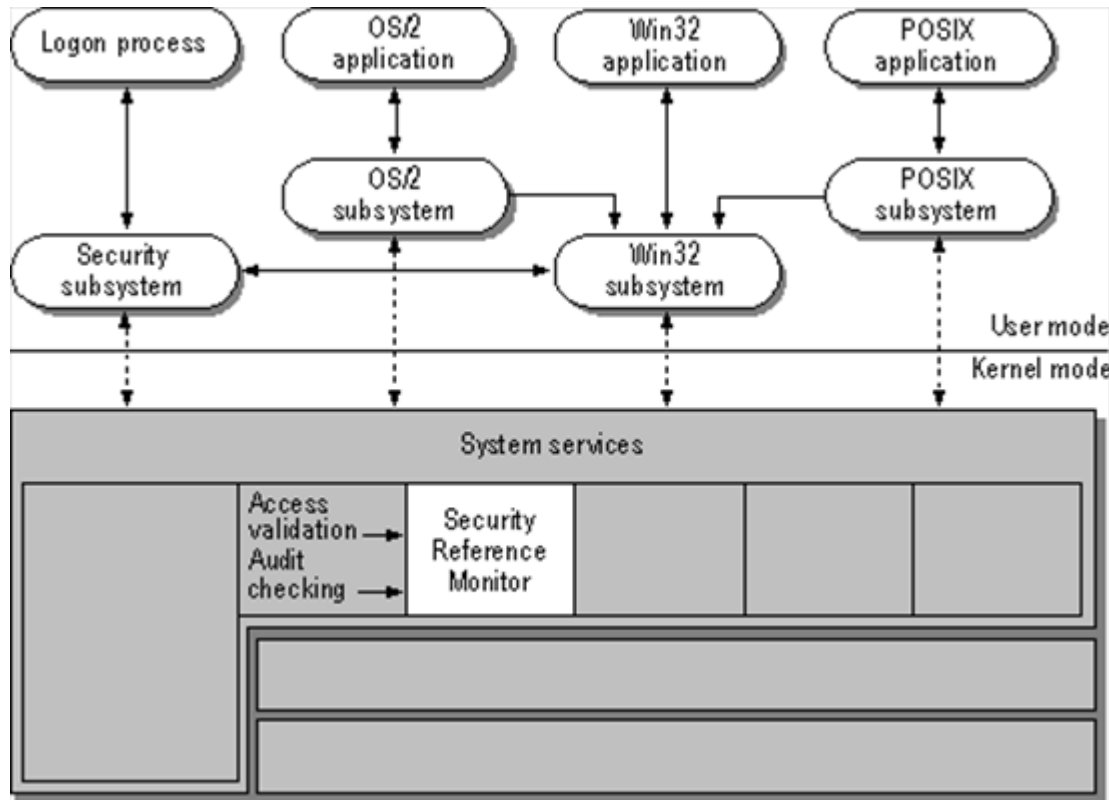


Приклад: Wine для Linux



- Wine — програмне забезпечення, що дозволяє користувачам Linux (і деяких інших UNIX-подібних систем) на архітектурі x86 і amd64 виконувати 16-, 32- і 64-розрядні програми Windows
- Wine є альтернативною реалізацією Windows API — перехоплює виклики програм до системних бібліотек і виконує їх
- Wine дозволяє завантажувати Windows DLL
- Wine портований також на Windows, де використовується для запуску застарілих несумісних програм
- Wine не є емулятором! В результаті програми під Wine виконуються не набагато повільніше, ніж у рідному середовищі, у деяких випадках — швидше

Приклад: Windows NT



- Виконавчі системи POSIX і OS/2 у Windows NT не є реалізацією ядер відповідних ОС
- Натомість, ці системи є реалізацією підсистем режиму користувача, що виконуються безпосередньо над ядром Windows NT
- Вони реалізовані як захищені сервери



Віртуалізація і емуляція

- **Віртуалізація** — створення віртуальних (штучних) об'єктів або середовищ
 - Зокрема, створення набору обчислювальних ресурсів або їх логічного об'єднання, абстрагованого від апаратної реалізації, що забезпечує при цьому логічну ізоляцію обчислювальних процесів, що виконують на одному фізичному ресурсі
 - **Приклад:** запуск кількох операційних систем на одному комп'ютері, у загальному випадку різних. Наданням їм ресурсів керує **хостова операційна система** або **гіпервізор**
- **Емуляція** — копіювання функцій одної обчислювальної системи (**гостя**) на іншій, відмінній від першої, обчислювальній системі (**хості**) таким чином, щоби емульована поведінка максимально відповідала поведінці оригінальної системи (**гостя**)
 - Виконується комплексом програмних засобів, апаратних засобів, або їх сполученням
 - **Приклад:** програмна емуляція апаратних засобів



Проблема віртуалізації

- Віртуальна машина (гість) повинна **не мати можливості** впливати на середовище гіпервізора (хосту) та(або) інших віртуальних машин
 - Приклад **“небезпечної” команди** — заборона переривань
- **Найбезпечніший варіант** — програмна емуляція апаратної платформи (усі команди гостя виконують в інтерпретаторі)
 - Це повільно і неефективно
- **Ефективна реалізація** — виконувати команди гостя безпосередньо на процесорі, але “небезпечні” команди перехоплювати і емулювати їх програмно
 - За певних властивостей апаратної архітектури можна було б виконувати операційну систему гостя у режимі користувача, а ті інструкції, які вимагають режиму ядра (“небезпечні” команди), емулювати програмно
 - Не усі апаратні архітектури відповідають таким вимогам
(Intel x86 — не відповідає)



Небезпечні команди

- **Чутливі інструкції** (*sensitive instructions*) — команди процесора, що виконуються по-різному в залежності від того, у якому режимі (користувача або ядра) вони виконуються
 - Введення-виведення, налаштування блока керування пам'яттю, записування у регістр прапорців тощо
- **Привілейовані інструкції** (*privileged instructions*) — команди процесора, які у разі виконання їх у режимі користувача викликають системне переривання
- Віртуалізація можлива лише тоді, коли чутливі інструкції є підмножиною привілейованих інструкцій
 - В Intel 386 команда POPF (заміна регістру прапорців на слово зі стеку) у режимі користувача не замінить деякі прапорці (зокрема, блокування-розблокування переривань), але і переривань не викличе
 - Апаратна архітектура, що підтримує віртуалізацію на архітектурі Intel 386, з'явилася лише у 2005 році
 - Intel — Virtualization Technology (VT)
 - AMD — Secure Virtual Machine (SVM)



Види віртуалізації операційних систем

- *Програмна віртуалізація*
 - *Динамічна (бінарна) трансляція* — гіпервізор фактично переписує код, замінюючи небезпечні команди на інші послідовності команд (так працювала VMware до появи архітектури VT)
 - *Паравіртуалізація* (“неповна” віртуалізація) — гіпервізор надає API, ОС гостя взаємодіє з ним
- *Апаратна віртуалізація*
 - Найкращий варіант, але вимагає підтримку з боку апаратури
vSphere, Xen, Hyper-V
- Віртуалізація на рівні операційної системи
 - Кілька екземплярів простору користувача у рамках однієї операційної системи
Docker, Solaris Containers/Zones, FreeBSD Jail
- Віртуалізація на рівні процесу
WINE

Типи гіпервізорів



- *Автономний гіпервізор* (тип 1, X)
 - Має власні вбудовані драйвери пристроїв, моделі драйверів і планувальник
 - Працює в оточенні усіченого ядра
VMware ESX, XenServer

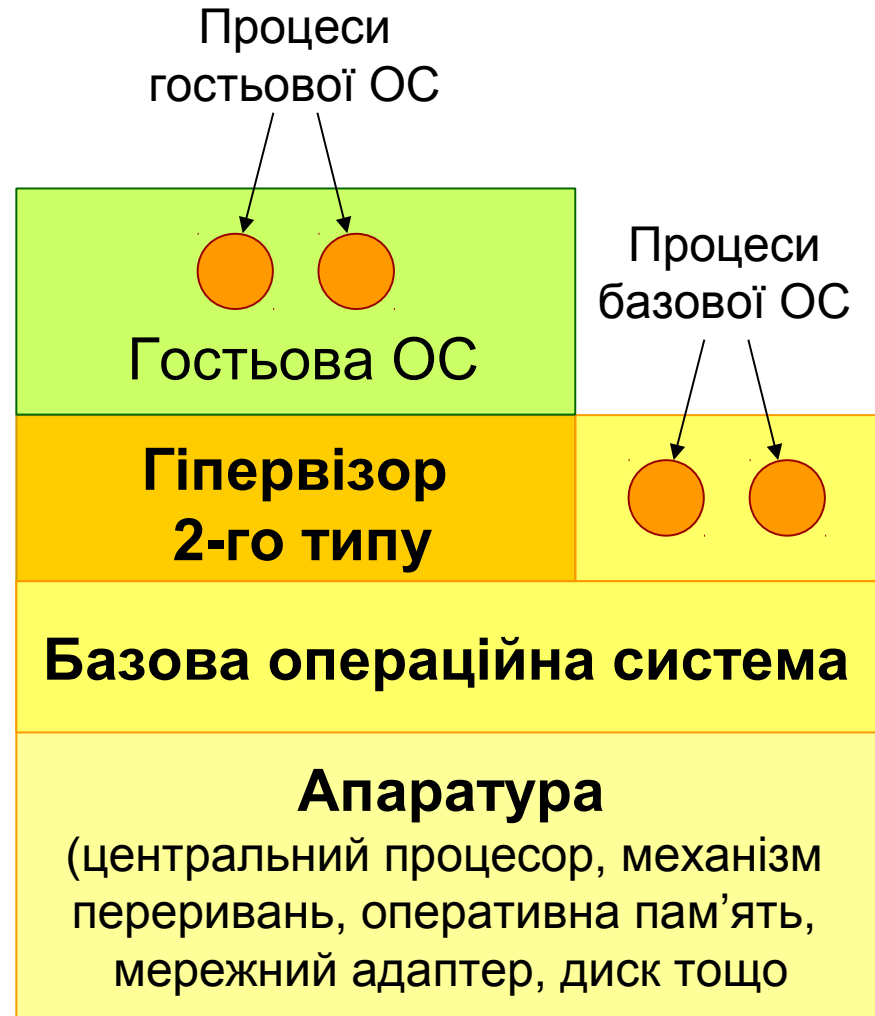
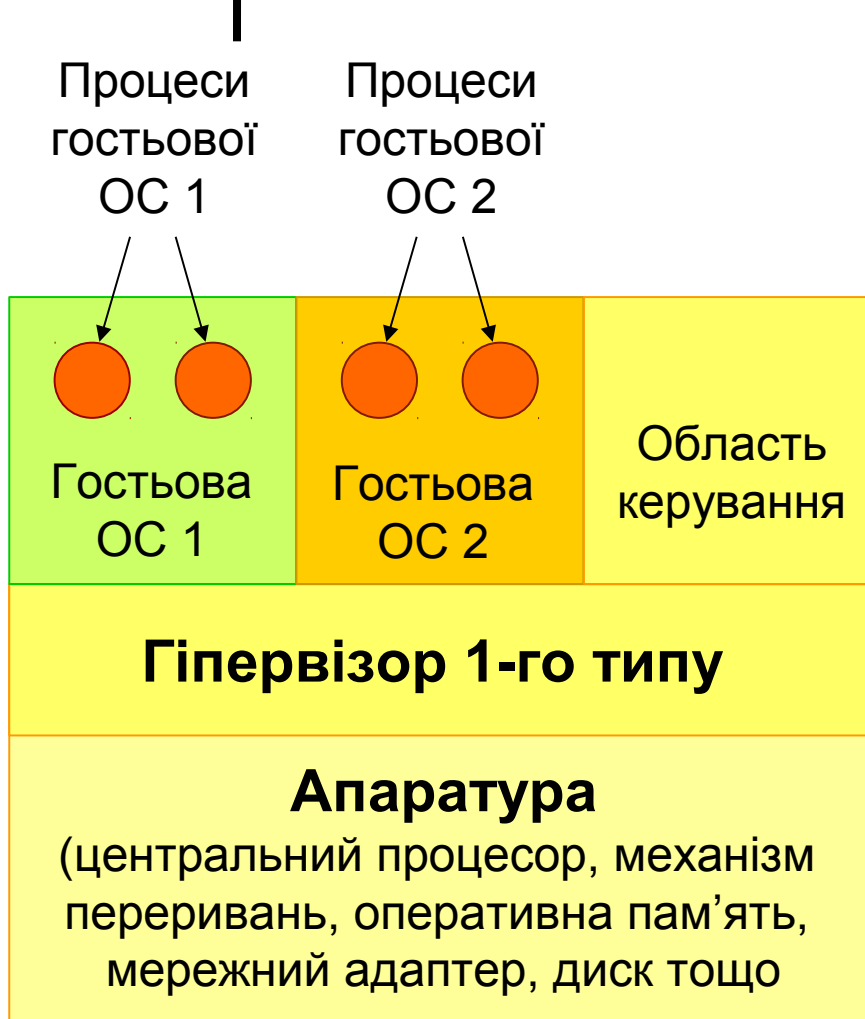


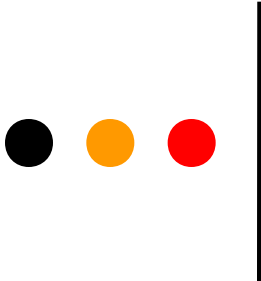
- *На основі базової ОС* (тип 2, V)
 - Працює в одному кільці з ядром базової ОС (кільце 0 — режим ядра)
 - Гостьовий код может виконуватись на фізичному процесорі, але виконання чутливих інструкцій здійснюється через звичайний процес базової ОС — монітор рівню користувача



VMware Workstation, VirtualBox, QEMU

Типи гіпервізорів





ОС мобільних пристроїв: специфічні вимоги

- Мобільні пристрої мають **обмежені ресурси**: обчислювальна потужність, пам'ять, обсяг сховища даних, ємність батареї
- Мобільні пристрої мають **різні форм-фактори і можливості**: розміри і роздільну здатність екранів, клавіатури (або їх відсутність взагалі), різні бездротові інтерфейси, інтегровані і зовнішні периферійні пристрої, різні архітектури процесорів (ARM, x86) тощо
- ОС має бути орієнтована на **непрофесійного користувача**: має бути легкою в користуванні, інтуїтивно зрозумілою, не вимагати значних зусиль в обслуговуванні, але забезпечувати достатню безпеку, зокрема протидію шкідливим програмам
- Ринок вимагає позитивних перших вражень, швидкого розвитку, і не пробачає помилок



Принципи, виведені з вимог

- Ядро ОС має бути компактним і вимагати мінімальний обсяг ресурсів.
- ОС не буде застосовувати витіснення коду і даних з оперативної пам'яті на зовнішній пристрій і повинна працювати і керувати прикладними програмами у фіксованому обсязі ОП.
- Повинні бути надані API і механізми, що дозволять пристрою “спати” будь-коли це можливо для економії батареї.
- Прикладні програми мають бути абстраговані від ОС і апаратури.
- Прикладні програми мають виконуватись у пісочниці для мінімізації впливу шкідливих програм.
- Стандартні компоненти, що існують, повинні застосовуватись максимально.
- Необхідно надати можливість стороннім розробникам створювати безліч прикладних програм.
- Треба створити власний магазин прикладних програм (App store) і зробити його привабливим для розробників .



Реалізація принципів в ОС Android

1. Модифіковане ядро Linux

- Linux є надійною масштабованою технологією, працює на мільйонах вбудованих пристроїв, має потужну підтримку різноманітної апаратури і розвинені засоби безпеки.
- Плюс, вона з відкритим кодом.

2. Java для прикладних програм

- Java є доведеною технологією популярною у розробників: Enterprise (JSP), Mobile (J2ME) та інші. Програми Java виконуються у віртуальній машині, тобто можуть бути незалежними від ОС і обладнання.
- Для мінімізації вимог до процесору і пам'яті бібліотеки Java і середовище виконання Java (VM/runtime) були суттєво перероблені, тому “Android Java” не є 100% еквівалентною “Sun/Oracle Java”.

3. Оптимізована Java VM в якості середовища виконання

- Розробники Android написали власну “легку” Java VM і середовище виконання, оптимізоване для мобільних пристроїв.
- Перша версія, що застосовувалась до Android 4.4, називалась Dalvik.
- Нова версія середовища виконання називається ART (Android RunTime).
- Оптимізована архітектура VM включає специфічні для мобільних пристроїв риси, такі як розділювана пам'ять, швидкий старт, а також втрачає найбільш “важкі” вимоги Java VM (нажаль, це включає і суттєву частину безпеки)

| Архітектура Android

APPLICATIONS

Home

Contacts

Phone

Browser

...

APPLICATIONS FRAMEWORK

Activity
Manager

Window
Manager

Content
Providers

View
System

Package
Manager

Telephony
Manager

Resource
Manager

Location
Manager

Notification
Manager

LIBRARIES

Surface
Manager

Media
Framework

SQLite

OpenGL | ES

FreeType

WebKit

SGX

SSL

libc

ANDROID RUNTIME

Core Libraries

Dalvik/ART
Virtual Machine

LINUX KERNEL

Display
Driver

Camera Driver

Flash Memory
Driver

Binder (IPC)
Driver

Keypad
Driver

WiFi Driver

Audio
Drivers

Power
Management