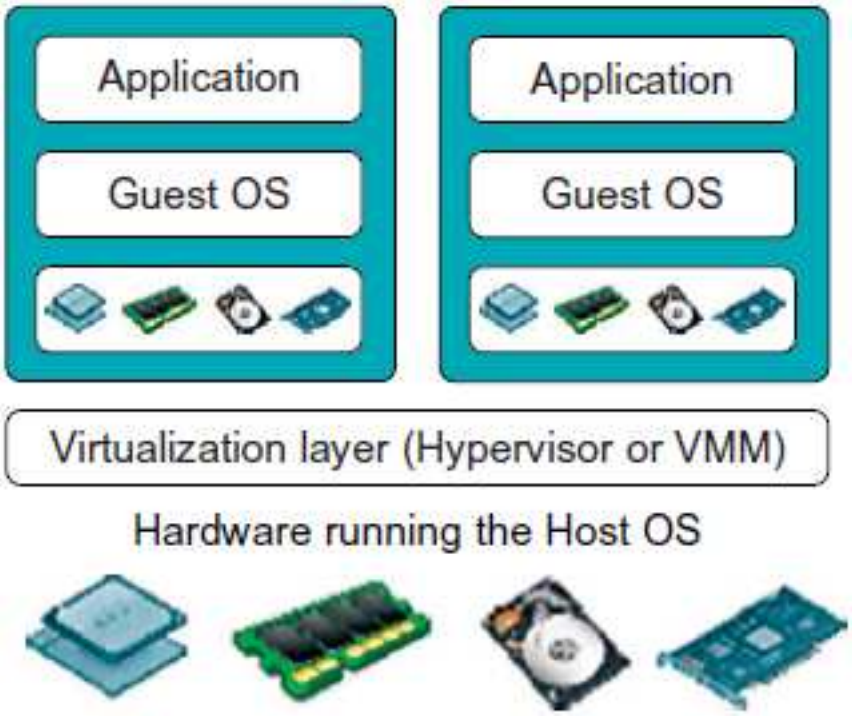
# Maszyny wirtualne

**Maszyna wirtualna** wprowadza warstwę abstrakcji, która znajduje się nad poziomem sprzętu, ale poniżej systemu operacyjnego (lub oprogramowania bezpośrednio obsługującego sprzęt):

- prezentuje warstwę sprzętu wirtualnego wspieranego przez sprzęt fizyczny,
- **monitor maszyny wirtualnej** (VMM — *Virtual Machine Manager*), zwany również **hypervisorem** implementuje interfejs wirtualizacji, tworzy iluzję izolowanych maszyn wirtualnych.




System zainstalowany i działający w ramach maszyny wirtualnej nazywamy systemem **gościa**.

System operacyjny pracujący bezpośrednio na fizycznym sprzęcie nazywamy systemem **gospodarza**.

# Interfejs VMM a interfejs systemu operacyjnego

- System operacyjny zapewnia wysoki poziom abstrakcji:
  - mechanizm szeregowania przydzielający procesory wątkom „na wyłączność”,
  - wirtualna przestrzeń adresowa procesów wspierana przez pamięć wirtualną,
  - dostęp do dysku przez abstrakcję systemu plików,
  - dostęp do urządzeń przez API wysokiego poziomu (np. funkcje `open()`, `read()`, `write()` w systemach uniksowych).

- VMM zapewnia niski poziom abstrakcji:
  - wydaje się, że oprogramowanie działa na „gołym” sprzęcie, z bezpośrednim dostępem do pamięci fizycznej i urządzeń (każda maszyna wirtualna zwykle posiada własny system operacyjny).

- Zarówno system operacyjny, jak i VMM starają się odizolować różnych lokatorów (procesy/maszyny wirtualne) i wymuszają sprawiedliwość (*fairness*) ze względu na wykorzystywany fizyczny sprzęt.

# Kiedy wirtualizacja jest przydatna?

Multipleksowanie (naprzemienne użycie) fizycznego sprzętu w centrach obliczeniowych:

- Klient chce, aby jego aplikacja działała na dedykowanej maszynie ... ale jego aplikacja może mieć niski poziom wykorzystania sprzętu!
- Złe rozwiązanie: przydzielenie oddzielnej fizycznej maszyny każdemu klientowi.
- Dobre rozwiązanie: wiele maszyn wirtualnych na bazie pojedynczej maszyny fizycznej.
- Maszyna fizyczna będzie mocno wykorzystana, nawet jeśli poszczególne maszyny wirtualne nie są.
- Operatorzy centrów obliczeniowych mogą kupować mniej fizycznych maszyn!
- Zarządzanie farmą maszyn wirtualnych jest dużo łatwiejsze niż fizycznych, np. instalację systemu można wykonać jeden raz na dysku wirtualnym, a potem powielić ten dysk dowolną liczbę razy.

# Kiedy wirtualizacja jest przydatna?

Technologia wirtualizacji w połączeniu ze zdalnym dostępem przez Internet umożliwia rozwój pochodnych technologii, takich jak wirtualne serwisy obliczeniowe.

- Możliwe jest tworzenie wirtualnych usług obliczeniowych „w chmurze”, pozwalających uruchamiać aplikacje wielu klientów.

  Przykłady: Amazon EC2 (Elastic Compute Cloud), Google Compute Engine, Microsoft Azure, OVH, Hostinger, Dreamhost, i wiele innych.

# Kiedy wirtualizacja jest przydatna?

Bezpieczeństwo: maszyny wirtualne w naturalny sposób tworzą izolowane środowisko pracy dla każdego z systemów gości. Gwarantuje to pełne bezpieczeństwo systemu gospodarza przed zagrożeniami ze strony gości, i bezpieczeństwo gości przez sobą nawzajem, ponieważ ich działania nie mogą wyjść poza ich własny system operacyjny.

- Jeśli aplikacja działająca na maszynie wirtualnej potrzebuje komunikować się ze światem zewnętrznym to można to zapewnić przez: (i) współdzielone pliki, (ii) połączenia sieciowe przez wirtualne interfejsy sieciowe. Jednak taką komunikację łatwo jest monitorować i zabezpieczyć.
- Wirtualizacja pozwala na zapewnienie bezpieczeństwa systemów i ich danych, ponieważ gdy dane nie są przechowywane na indywidualnych komputerach użytkowników łatwiej jest zapewnić bezpieczeństwo fizycznego dostępu, tworzenie kopii zapasowych, itp.
- Jednak, wirtualizacja i przetwarzanie w chmurze wprowadza pewne nowe zagrożenia, ponieważ często wrażliwe dane użytkowników muszą znaleźć się w środowisku, którego ich właściciel w pełni nie kontroluje.
- Dzięki wirtualizacji można uruchomić podejrzaną aplikację na maszynie wirtualnej (np. załącznik do wiadomości e-mail, oprogramowanie z nieznanego źródła, itp.), zamiast na własnym komputerze.

# Kiedy wirtualizacja jest przydatna?

Zwiększona produktywność dla programistów:

- Scenariusz 1: można uruchomić Linuksa jako gospodarza, i na nim stabilne i bezpieczne oprogramowanie deweloperskie, a na maszynach wirtualnych używać Windowsa i/lub MacOSa do browsowania/komunikacji/rozrywki.
- Scenariusz 2: programista pracujący nad kodem jądra może testować je na maszynie wirtualnej, tak, że gdy jądro zawiesza się lub „pada", maszyna developerska nadal pracuje i jest stabilna!
- Scenariusz 3: uruchomioną aplikację (np. w celu testowania) można **zatrzymać**, tzn. zawiesić w celu np. przeanalizowania stanu, i późniejszego wznowienia. Ten mechanizm jest dostępny w zwykłych systemach operacyjnych dla procesów, jednak maszyny wirtualne pozwalają znacznie rozszerzyć jego wykorzystanie.

  Można wykonać kopię, albo **migawkę** (*snapshot*), całego działającego systemu gościa. Tę kopię można później uruchomić, np. w celu zresetowania stanu systemu. Można ją również powielić, przenieść i uruchomić na innym systemie. Można również zaprogramować automatyczne okresowe wykonywanie migawek.

  Jest nawet możliwe równoważenie obciążenia maszyn fizycznych, przez przenoszenie pracujących migawek (*live migration*) między maszynami, z zachowaniem stanu, np. połączeń sieciowych.

# Wady/niedostatki wirtualizacji

Jakie są problemy związane z wirtualizacją?

- Częściowo rozwiązuje problemy istniejące gdzie indziej:
  - niedostateczna izolacja procesów w ramach systemu operacyjnego,
  - niedostateczne zabezpieczenia w ramach systemu operacyjnego,
  - aplikacje napisane w sposób utrudniający migrację, odporność na awarie, itp.
- Strata wydajności obliczeniowej wynikająca z wirtualizacji: 5...10...20%
  - jednak gdy komunikację międzyprocesową trzeba zamienić na zdalne wywoływanie procedur to strata wydajności staje się poważna,
  - pytanie czy dyski gospodarza są na tyle wydajne by obsłużyć wielu gości, podobnie interfejsy sieciowe,
- Systemy operacyjne w dużym stopniu są dojrzałe i stabilne, natomiast VMM są nową, rozwijającą się technologią, i źródłem dziur przez dłuższy czas.

# Wirtualizacja: słowo o historii

Wirtualizacja nie jest nową koncepcją. Wczesna wersja maszyny wirtualnej pojawiła się w roku 1972 w systemie operacyjnym VM/370 firmy IBM. Dla wykonywania aplikacji tworzona była maszyna wirtualna, z własnym wirtualnym dyskiem, w ramach której można było wykonywać zadania, typowo za pośrednictwem jednoużytkownikowego systemu operacyjnego CMS.

System VM ewoluował przez lata i istnieje do dziś (z/VM). Jednak poza nim, wirtualizacja nie zyskała szerszego uznania w technologiach informacyjnych, aż do lat 1990-tych, kiedy pojawiły się coraz silniejsze i lepiej wyposażone procesory rodziny 80x86, pobudzając zainteresowanie wirtualizacją.

Fundamentalna praca:

> G.J. Popek, R.P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures", Communications of the ACM, Volume 17, Number 7 (1974), pages 412-421.

definiuje charakterystyki monitora maszyny wirtualnej (VMM), oraz określa cechy architektury sprzętowej zapewniającej wsparcie dla wirtualizacji.

# Technologie składowe

Wirtualizację można zaimplementować całkowicie w domenie programowej, jednak jest ona wtedy całkowicie nieefektywna, co wyklucza to podejście z jakichkolwiek zastosowań rzeczywistych.

W praktyce niezbędna jest implementacja umożliwiająca wykonywanie sprzętowe systemów gości, z wyjątkiem małej liczby wybranych operacji. Implementacja takich mechanizmów jest trudna, i w dużym stopniu zależna od właściwości danego procesora. Mają one związek z uprzywilejowaniem danych operacji. W oczywisty sposób, system gospodarza (i pracujący w nim VMM) musi kontrolować dostęp do operacji uprzywilejowanych, umożliwiających zarządzanie procesorem i systemem.

Do technik wirtualizacji należą: (i) pułapka i emulacja, oraz (ii) tłumaczenie binarne, omówione tu po kolei.

# Procesory wirtualne

Koncepcją implementowaną w niektórych systemach jest **procesor wirtualny VCPU**. Nie wykonuje on żadnego kodu, a jedynie reprezentuje stan procesora dla potrzeb maszyny gościa. VMM przechowuje wirtualne procesory wszystkich gości. W momencie przełączenia kontekstu na danego gościa, VMM ładuje jego VCPU w podobny sposób jak egzekutor ładuje zawartość bloku kontrolnego procesu (PCB) w chwili przełączania kontekstu dla jego wykonania.

# Dwa tryby pracy procesora

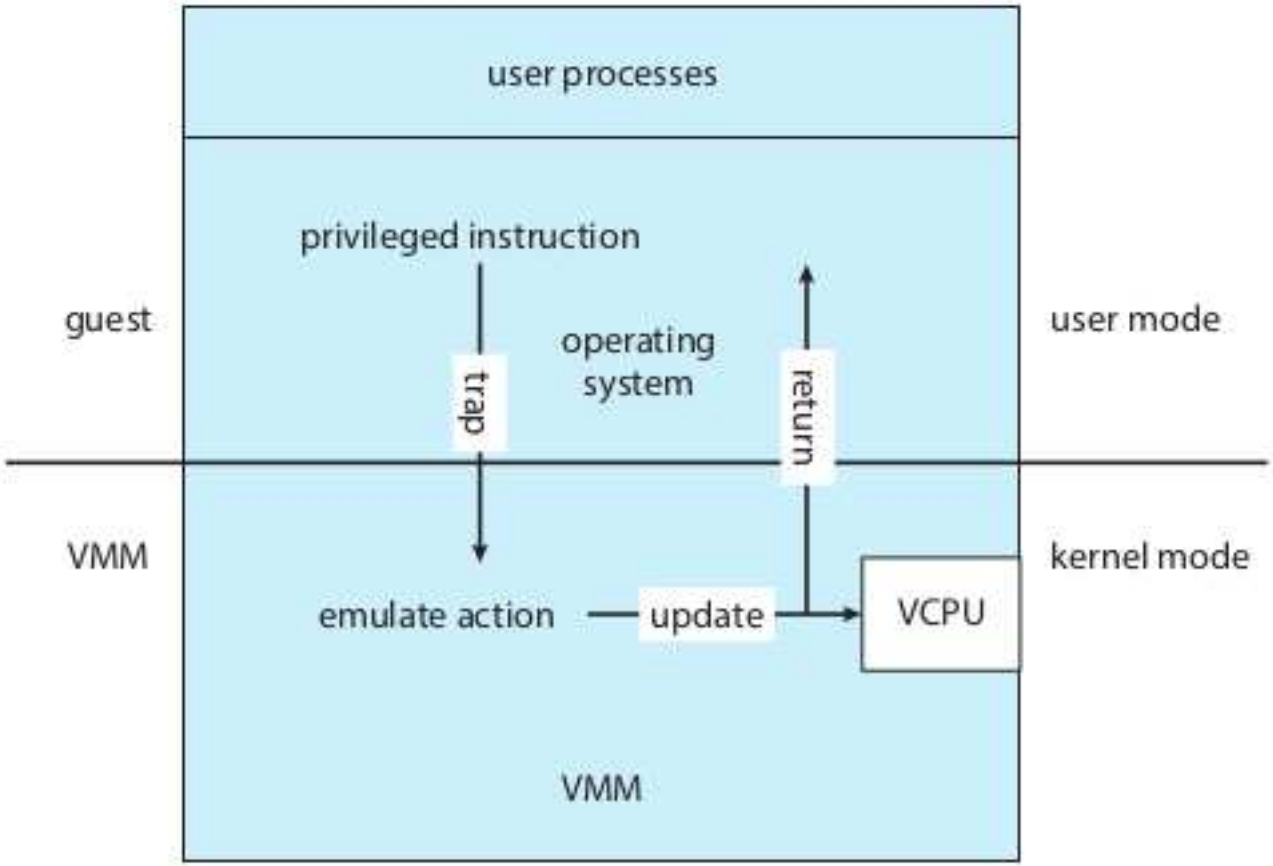
Typowy procesor posiada (co najmniej) dwa tryby pracy: uprzywilejowany **tryb jądra** (*kernel mode*) i zwykły **tryb użytkownika** (*user mode*). Pewne instrukcje określone jako uprzywilejowane mogą być wykonane tylko w trybie jądra. Kod zwykłych aplikacji wykonuje się w trybie użytkownika, a kod jądra systemu operacyjnego w trybie jądra. Gdy program użytkownika wywołuje funkcję systemową, w szczególności aby wykonać instrukcję uprzywilejowaną, lub generuje przerwanie, wykonuje się kod jądra i procesor przechodzi chwilowo do trybu jądra.

Podobnie, maszyna wirtualna musi mieć analogiczne dwa tryby pracy: wirtualny tryb użytkownika, i wirtualny tryb jądra. Podobnie jak w zwykłej maszynie fizycznej, akcje programu użytkownika powodujące przejście do trybu jądra powinny w maszynie wirtualnej spowodować przejście do wirtualnego trybu jądra. Jednak oba wirtualne tryby pracy działają w fizycznym trybie użytkownika.

Zatem w jaki sposób maszyna wirtualna może wykonać operacje uprzywilejowane?

# Pułapka i emulacja

Gdy jądro maszyny gościa wywołuje instrukcję uprzywilejowaną, stanowi to błąd (bo system jest w trybie użytkownika) i wyzwala **pułapkę** w VMM (gospodarza). VMM przejmuje kontrolę, i wykonuje (a raczej emuluje) akcję w imieniu jądra gościa, a następnie powoduje powrót do kodu gościa. Ta metoda jest nazywana pułapką i emulacją. W oczywisty sposób, emulacja akcji gościa powoduje spowolnienie jego pracy, zatem gdy aplikacja gościa wykonuje operacje uprzywilejowane, wykonuje się (dużo) wolniej niż gdy wykonuje normalny kod użytkownika.



# Procesory niewirtualizowalne

Niektóre procesory nie mają poprawnego podziału na instrukcje uprzywilejowane i nieuprzywilejowane (w istocie naruszają zasady Goldberga i Popka). Należy do nich rodzina Intel 80x86.

Przykład: instrukcja `popf` ładuje rejestr flag z zawartości stosu. Niektóre z flag powodują akcje uprzywilejowane, ale instrukcja `popf` nie jest instrukcją uprzywilejowaną. Zamiast tego, gdy wykonuje się w trybie użytkownika, ignoruje niektóre z flag, i nadpisuje tylko te niezwiązane z przywilejami. W trybie jądra `popf` nadpisuje wszystkie flagi.

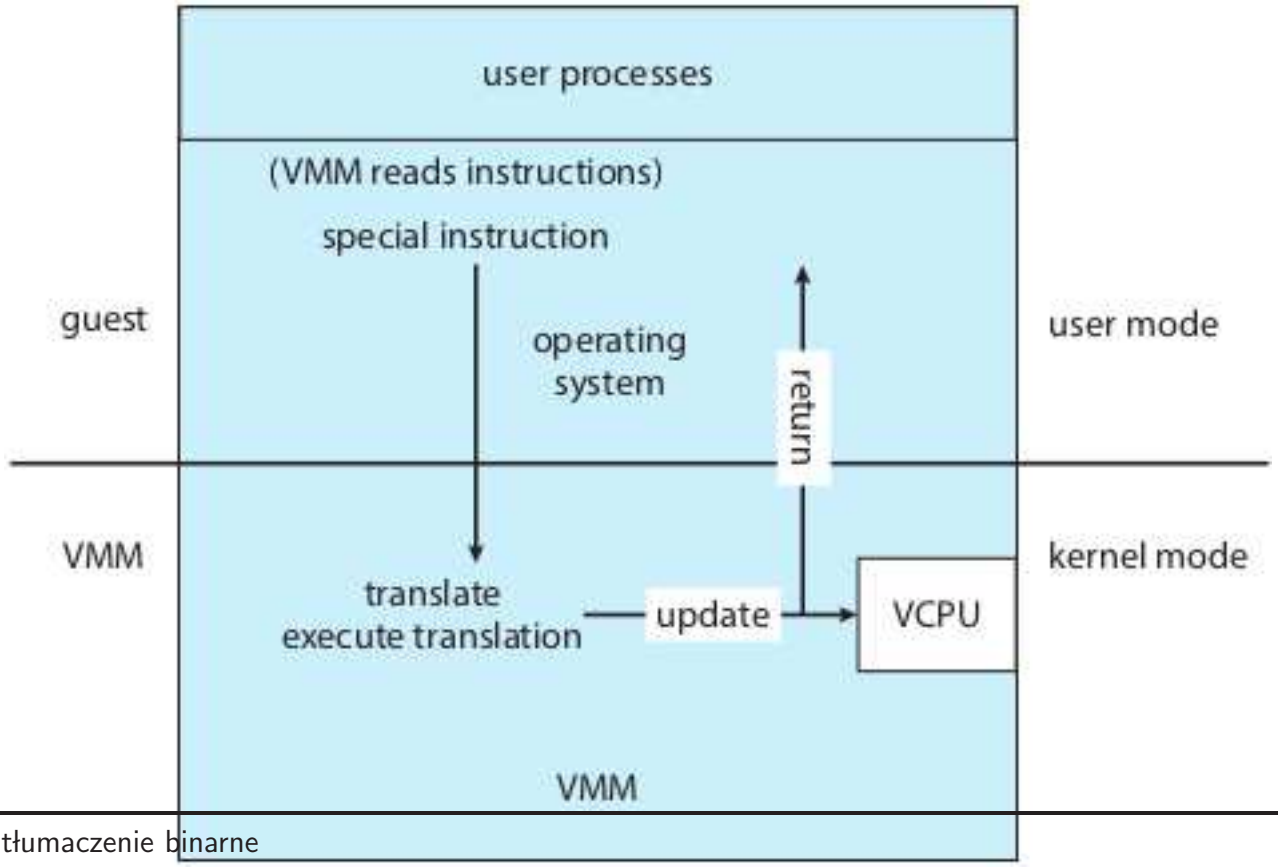
Instrukcje tego typu, które mają dostęp do niskopoziomowego stanu procesora, nazywamy instrukcjami **wrażliwymi** (*sensitive*). Jedna z zasad wirtualizowalności Goldberga i Popka mówi, że wszystkie instrukcje wrażliwe procesora muszą być uprzywilejowane, tzn. wymagać trybu jądra.

Jednak ponieważ instrukcja `popf` nie jest uprzywilejowana, nie generuje ona pułapki w VMM, i jej akcje nie mogą być poprawnie emulowane przez VMM. Jest to instrukcja wrażliwa ale nieuprzywilejowana, a zatem narusza zasadę Goldberga i Popka. Ogólnie w architekturze 80x86 istnieje 17 podobnych instrukcji, co powoduje, że ta rodzina procesorów jest niewirtualizowalna.

# Tłumaczenie binarne

Problem obsługi uprzywilejowanych instrukcji na niewirtualizowalnych procesorach można rozwiązać za pomocą metody **tłumaczenia binarnego** (*binary translation*):

1. jeśli VCPU gościa jest w trybie użytkownika, to kod gościa może wykonywać się bezpośrednio na fizycznym procesorze,
2. jeśli VCPU gościa jest w trybie jądra, to VMM analizuje każdą instrukcję kodu gościa, i wykonuje je albo normalnie, gdy nie są wrażliwe, albo przez przetłumaczenie je na odpowiednie instrukcje wykonujące równoważne operacje.



# Wydajność

Metody opisane powyżej ogólnie powodują spadek wydajności wykonywania aplikacji, w porównaniu z wykonywaniem na maszynie fizycznej. Wiele szczegółowych technik jest stosowanych, aby tę wydajność poprawić.

Firma VMware przeprowadziła test wydajności tłumaczenia binarnego na przykładzie uruchamiania i natychmiastowego wyłączania systemu Windows XP. Ta procedura wyzwoliła 950,000 tłumaczeń binarnych, każde trwające około 3 mikrosekund, co spowodowało wydłużenie całego procesu o 3 sekundy (5 procent) w porównaniu z jego realizacją na maszynie fizycznej.

# Obsługa tablic stron

System operacyjny gościa obsługuje pamięć wirtualną dla potrzeb swojego systemu, i w jego ramach zestaw tablic stron procesów. W oczywisty sposób, mają się one nijak do tablic stron systemu gospodarza i treści pamiętanych w (oraz usuwanych z) pamięci fizycznej. Zatem system pamięci wirtualnej gościa działa w swoim wirtualnym świecie ☹ i nie jest w stanie zapewnić efektywnego korzystania z pamięci fizycznej swoich aplikacji.

Aby zapewnić poprawną pracę systemu pamięci wirtualnej gości, VMM może stosować metodę **dublowanych tablic stron** SPT (*shadow page tables*). Każda SPT reprezentuje stan pewnej tablicy stron gościa, podobnie jak VCPU reprezentuje stan CPU z punktu widzenia gościa.

Gdy gość modyfikuje tablicę stron, VMM przechwytuje tę operację, i wykonuje odpowiednią zmianę w SPT. Gdy gość wykonuje się na CPU, VMM przekierowuje bieżącą tablicę stron na właściwą SPT.

# Wsparcie sprzętowe wirtualizacji

Bardziej efektywna implementacja wirtualizacji jest możliwa, gdy w procesor zostaną wbudowane specjalne mechanizmy ją wspierające. Na przykład, Intel wprowadził w 2005 roku technologię VT-x, a AMD w 2006 ADM-V, które eliminują potrzebę tłumaczenia binarnego.

Wprowadzają one dodatkowe tryby procesora odpowiednie dla gospodarza i gościa (host/guest w terminologii AMD, root/nonroot w terminologii Intel). W trybie gospodarza VMM definiuje charakterystyki poszczególnych gości, i gdy gość wykonuje się w trybie gościa, widzi konfigurację sprzętową, która została dla niego zdefiiowana.

Wymienione architektury wprowadzają również mechanizmy wsparcia pamięci wirtualnej dla systemów gości, które powodują, że nie jest już konieczne tworzenie dublowanych tablic stron przez VMM.

Ponadto wprowadzono wsparcie sprzętowe dla operacji I/O, pozwalając na skuteczne programowanie transferów DMA przez gości.

Te modyfikacje spowodowały, że wirtualizacja stała się prosta. W systemie MacOS istnieje biblioteka HyperVisor Framework pozwalająca tworzyć maszyny wirtualne w kilku wierszach kodu.

# Parawirtualizacja

Parawirtualizacja działa inaczej niż pozostałe podejścia do wirtualizacji. Zamiast utrzymywać złudzenie gościa, że pracuje na maszynie fizycznej i zarządza nią, system gościa jest modyfikowany, aby współpracował z VMM i ... zachowywał się jak gość 

Jest to możliwe w przypadku systemów z dostępnym kodem źródłowym (*open source*) i licencją pozwalającą na jego modyfikację. Typowymi przykładami jest Linux i rodzina BSD systemu Uniksa.

Przykładem i liderem w technologii parawirtualizacji jest system Xen. Wprowadził on prostą i czystą warstwę abstrakcji dla urządeń I/O i odpowiednią komunikację pomiędzy gościem a VMM dotyczącą tych urządzeń. W odniesieniu do pamięci wirtualnej, zamiast dublowanych tablic stron, Xen wprowadził specjalny mechanizm `hypercall` od gościa do VMM kiedy gość chce zmodyfikować swoją tablicę stron. Tablice stron gości pracują w trybie tylko do odczytu, co powoduje, że z wyjątkiem błędu braku strony, system pamięci wirtualnej gościa pracuje poprawnie. Obsługa błedu braku strony pracuje ze wsparciem ze strony VMM.

Należy nadmienić, że w przypadku nowych procesorów ze wsparciem dla wirtualizacji, modyfikacja systemów gości, i w ogóle cała parawirtualizacja nie jest potrzebna, i na takich procesorach Xen pracuje jak zwykły VMM, obsługując również systemy nieparawirtualizowalne, jak Windows.

# Wirtualizacja na poziomie systemu operacyjnego

Technologie **kontenerowe** jak Docker:

- kontener jest odizolowaną grupą procesów
- kontener widzi pewien zdefiniowany podzbiór procesów, systemu plików, łaczy sieciowych, itp.
- dobra wydajność, bo wszystko działa w ramach jednego SO, nie potrzeba tworzyć, bootować systemu, obsługiwać trybu uprzywilejowanego, martwić się o pracę pamięci wirtualnej, itp.
- migawki rejestrują tylko stan grupy procesów, nie całego systemu
- ale aplikacje muszą pracować w ramach SO gospodarza, nie ma wyboru.

# Wirtualizacja na poziomie środowiska oprogramowania

Maszyna wirtualna Javy JVM:

- program składa się z pewnej liczby klas Javy; kompilator kompiluje klasy do pliku `.class` **kodów bajtowych** (*bytecode*) niezależnych od sprzętu, zatem można go wykonać na dowolnej implementacji JVM
- JVM najpierw sprawdza poprawność kodu, a potem go wykonuje
- programy Javy nie operują na adresach pamięci, tylko na obiektach, pamięć po skasowanych obiektach jest odzyskiwana przez specjalny proces **odzyskiwania nieużytków** (*garbage collector*)
- kod Javy może być wykonywany przez interpreter języka kodów bajtowych
- inną możliwością jest kompilacja kodów bajtowych przy pierwszym użyciu danej metody do instrukcji maszynowych przez specjalny kompilator, tzw. *just-in-time compiler* (JIT)
- istnieją również sprzętowe implementacje JVM pozwalające wykonywać programy Javy bezpośrednio na procesorze