Basic Concepts

Decentralizované vs. distribuované

Ako sa počítače v prvom rade navzájom prepojili?

- **Integračné zobrazenie (view)** – potreba pripojenia existujúcich počítačov
- **Rozšírené zobrazenie (view)** – už existujúce počítače vyžadujú rozšírenia

Decentralizovaný systém - Sieťový počítačový systém, v ktorom sú procesy a zdroje nevyhnutne rozptýlené na viacerých počítačoch (Blockchain, riadiace systémy, federatívne učenie)

Distribuovaný systém - Sieťový počítačový systém, v ktorom sú procesy a zdroje vhodne a dostatočne rozptýlené na viacerých počítačoch (Škálovateľné (cloudové) služby, CDNs content delivery networks, NAS)

Centralizované riešenia sú vo všeobecnosti oveľa jednoduchšie

Decentralizácia aj distribúcia sú zložité úlohy - Neznáme závislosti, čiastočné zlyhania, vysoko dynamické (zvyšujúca sa zložitosť), zraniteľnosť.

Decentralizácia nikdy nie je cieľom systému, je to vlastnosť systému.

Distribúcia je cieľ ako dostatočne rozšíriť systém.

Perspektívy distribuovaného systému

- **Architektúra** - spoločné organizácie, spoločné znaky
- **Proces** - Pochopenie rôznych foriem procesov
- **Komunikácia** - Zariadenia, ktoré vymieňajú údaje medzi procesmi
- **Koordinácia** (procesov) - napr. globálne hodiny, vzájomný prístup atď.
- **Pomenovanie** - Rozlíšenie názvu prístupového bodu, pomenovanej entity
- **Konzistencia a replikácia** - Kompromisy medzi konzistenciou, replikáciou a výkonom
- **Odolnosť voči chybám** - Maskovanie porúch a ich obnova
- **Bezpečnosť** - zabezpečiť autorizovaný prístup k zdrojom a pochopiť úlohu bezpečnosti v predchádzajúcich perspektívach

Ciele dizajnu

1. **Zdieľanie zdrojov -** Prístup k vzdialeným zdrojom, Zdieľanie lokálnych zdrojov
2. **Transparentnosť -** Skrytie fyzickej distribúcie zdrojov

   Typy priehľadnosti

   - Prístup - Skryť rozdiely v reprezentácii údajov a spôsobe prístupu k objektu
   - Umiestnenie - Skryť, kde sa nachádza objekt
   - Premiestnenie - Skryť, že objekt môže byť počas používania presunutý na iné miesto
   - Migrácia - Skryť, že sa objekt môže presunúť na iné miesto
   - Replikácia - Skryť, že objekt je replikovaný

- Súbežnosť - Skryť, že objekt môže byť zdieľaný niekoľkými nezávislými používateľmi
- Zlyhanie - Skryť zlyhanie a obnovenie objektu

Stupeň prehliadnosti

3. **Otvorenosť -** Ponúka služby podľa štandardných pravidiel, ktoré popisujú syntax a sémantiku týchto služieb
   Interface Definition Language (IDL) – formálna syntax, neformálna sémantika.
   Správne špecifikácie sú úplné a neutrálne
   - Interoperabilita – rôzne komponenty môžu spolupracovať
   - Prenosnosť – znovupoužiteľnosť v rôznych distribuovaných systémoch

   Rozšíriteľnosť musí byť podporovaná a jednoduchá

   Oddelenie politiky od mechanizmov - Dynamická konfigurácia komponentov užívateľmi

4. **Spoľahlivosť**
   Dostupnosť - Meranie, či je systém pripravený na okamžité použitie.
   Spoľahlivosť - Meranie, či systém môže bežať nepretržite bez zlyhanie
   Bezpečnosť - Meranie bezpečnosti porúch
   Udržateľnosť - Meranie toho, aké ľahké je opraviť systém
5. **Bezpečnosť -** Autentifikácia, Autorizácia, Použitie kryptografie na vytvorenie zabezpečeného kanála
6. **Škálovateľnosť**
   Systém sa musí prispôsobiť rastúcemu počtu požiadaviek.
   Dimenzie škálovateľnosti:
   - **Veľkostná škálovateľnosti,**
   - **Geografická škálovateľnosť,**
   - **Administratívna škálovateľnosť**

   Techniky škálovania:
   - **Zväčšenie (vertikálne) vs. zväčšenie (horizontálne),**
   - **Skrytie komunikačných latencií (asynchrónna komunikácia),**
   - **Rozdelenie a distribúcia**
   - **Replikácia a ukladanie do vyrovnávacej pamäte**

   Obmedzenia škálovateľnosti
   - **Centralizované služby**
   - **Centralizované údaje**
   - **Centralizované algoritmy**

High performance distributed computing

Klaster - Jednoduché počítače zoskupené do „superpočítača“ vo vysokej rýchlosti siete

Mriežka - Spolupráca zdrojov z rôznych organizácií (virtuálne organizácia)

Cloud - Množina virtualizovaných zdrojov prístupných prostredníctvom webových služieb (IaaS, PaaS, SaaS)

Distribuované informačné systémy

Distribuované transakčné procesy

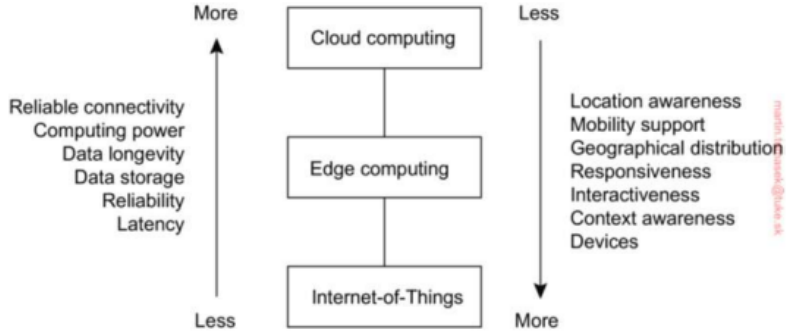
- Operácie sledujú vlastnosti ACID
- Definícia komunikačných primitív pre transakciu
- Vnorené transakcie a monitor spracovania transakcií

Integrácia podnikových aplikácií (EAI)

- **Servisne orientovaná architektúra (SOA)**
- Enterprise Service Bus (ESB)

Pervazívne systémy (spreading widely throughout an area)

- Internet vecí (IoT)
- Všadeprítomné výpočtové systémy
- Mobilné počítačové systémy
- Senzorové siete



Architectures

Štýly architektúr

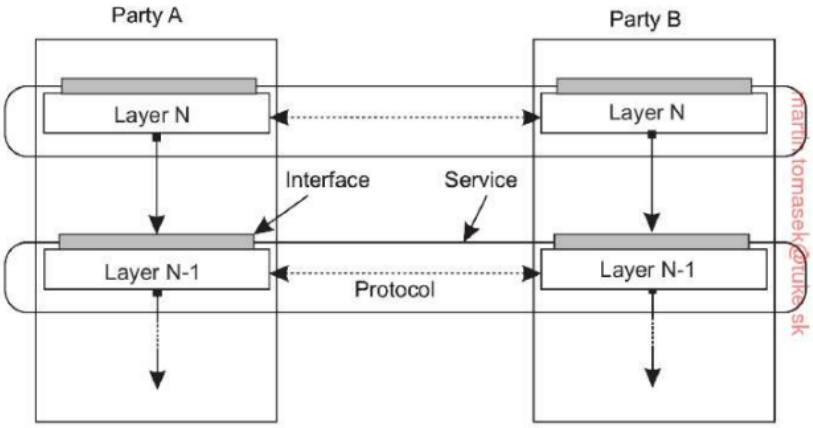
- Vrstvené architektúry
- Architektúry orientované na služby
- Publish-subscribe architectures

Downcall – volanie žiadosť-odpoveď na nižšiu úroveň

Upcalls – signalizácia vyššej úrovni o výskyte udalosti (volajúci operátor)

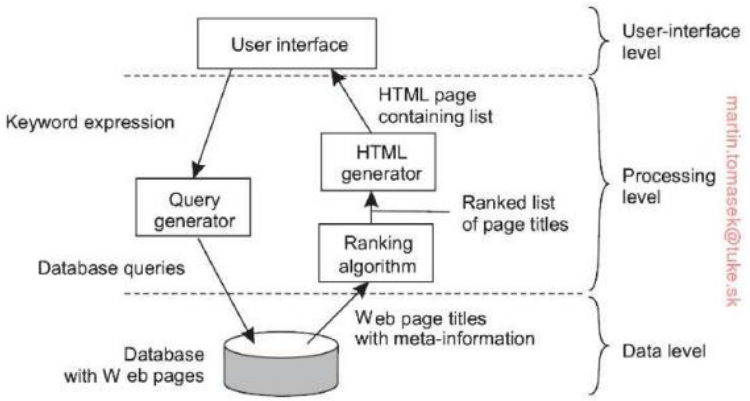
Vrstvené komunikačné protokoly - Komunikácia na jednej vrstve

- Služba
- Rozhranie
- Protokol



Vrstvenie aplikácie - Typické tri logické úrovne aplikácie

- Úroveň aplikačného rozhrania
- Úroveň spracovania
- Úroveň údajov

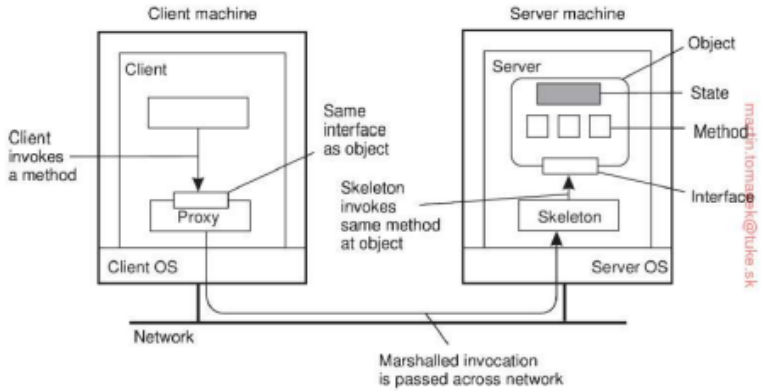


Servisne orientované architektúry (SOA)

- Zloženie existujúcich komponentov
- Princíp zapuzdrenia – Entita (Služba, objekt, mikroslužba, zdroj, ...) ako celok je realizovaná ako samostatná entita, hoci je môže prípadne využiť aj iné služby

Objektový architektonický štýl

- Objektové architektúry zodpovedajú komponentom
- Implementácia vzdialeného objektu pomocou proxy



Mikroslužby

- Zloženie mnohých rôznych služieb - Služba je jedna jednotka obsahujúca rôzne zdroje s jedným rozhraním (zvyčajne čo najmenšia)
- Mikroslužba predstavuje samostatnú, nezávislú službu - Modularizácia je kľúčom k navrhovaniu mikroslužieb, **Oddelenie funkčnosti a dát**
- Organizácia nasadzovania distribuovaných aplikácií naprieč rôzne vrstvy - **Message-oriented communication vyžaduje relaxované konzistenčné požiadavky**

Coarse-grained service composition

- Integrácia podnikových aplikácií, prevádzkujú rôzne organizácie
- Zložené služby musia fungovať v harmónii, aby poskytovali nové integrácie a funkcionality
- Väčšinou sa používa štýl založený na mikroslužbách

Architektúry založené na zdrojoch

- Alternatívny prístup k zloženiu služieb
  - Zbierka zdrojov, ktoré sú individuálne riadené komponentmi
  - Typický prístup pre webové distribuované systémy
- **RESTful architektúry**
  - Zdroje sú identifikované pomocou jedinej schémy pomenovania (napr. URL)
  - Všetky služby ponúkajú rovnaké rozhranie (napr. operácie HTTP)
  - Správy odoslané do služby alebo zo služby sú plne popísané (napr. dokumenty, XML alebo JSON)
  - Po vykonaní operácie služba zabudne všetko o volaní

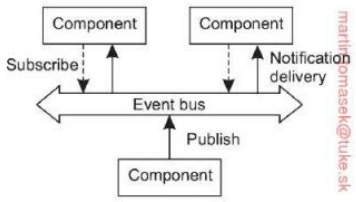
Vo webových aplikáciách široko (ne)používame RESTful prístup implementovať funkčnosť RPC

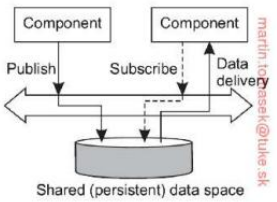
Architektúry publikovania a služieb

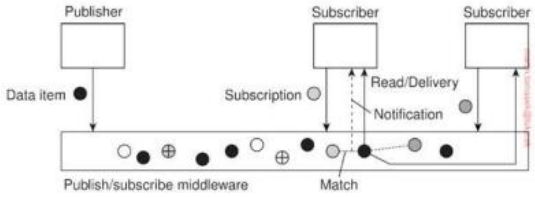
- **<u>Voľné prepojenie procesov</u>** - Procesy sa môžu pripojiť alebo opustiť distribuovaný systém nezávisle
- **<u>Priama koordinácia</u>** - Procesy sú **dočasne prepojené** (oba sú v prevádzke) **a referenčne prepojené** (oba si navzájom poznajú identifikátor/koncový bod), Koordinácia prebieha priamou komunikáciou
- **<u>Koordinácia poštových schránok</u>** - Procesy sú **dočasne oddelené** (nie všetky sú v prevádzke), ale sú **referenčne spojené**, Vkladanie správ do (možno zdieľanej) poštovej schránky
- **<u>Koordinácia založená na udalostiach</u>** - Procesy sú **dočasne prepojené, ale referenčne oddelené** (nepoznajú sa), Proces môže publikovať notifikáciu popisujúcu výskyt udalosti
- **<u>Koordinácia zdieľaného priestoru</u>** - Procesy sú **dočasne oddelené a referenčne oddelené**, Spracujte len umiestnenie udalosti do zdieľaného priestoru

Štýl založený na udalostiach verzus zdieľaný dátový priestor štýl

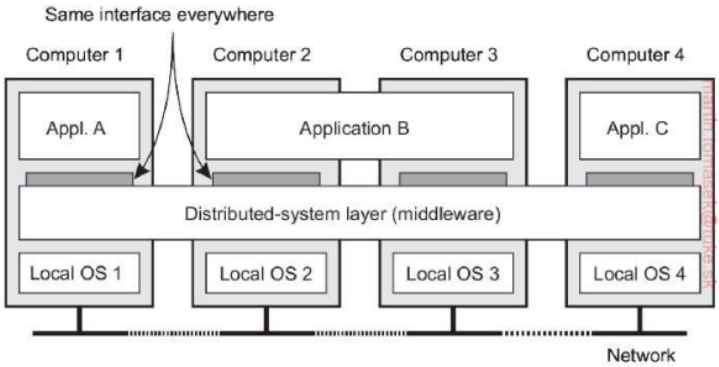
- Topic-based subscription - subscribe presnej hodnoty (témy - topicu)
- Content-based subscription – subscibe založený na rozsahu hodnôt, queries
- Processing of matching - Upozorniť a odoslať údaje subscriberovi, Upozorniť iba subscribera (integrita údajov v rámci middlewaru)







Middleware



- Správa zdrojov (zdieľanie a nasadenie v sieťovom prostredí)
- Zariadenia na komunikáciu medzi aplikáciami
- Bezpečnostné služby
- Účtovacie (accounting) služby
- Maskovanie a zotavenie z porúch

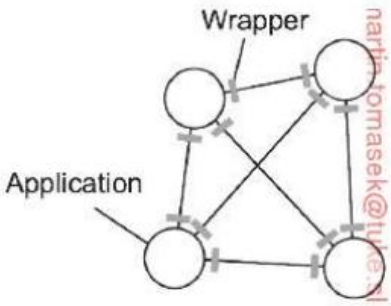
Na dosiahnutie otvorenosti používa middleware rôzne návrhové vzory

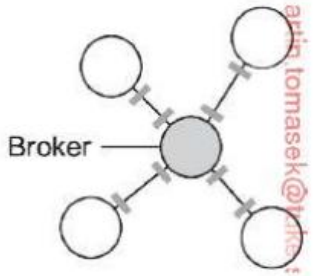
- **Wrappers**
- **Interceptors**

Modifikovateľný middleware je založený na princípoch adaptívneho vývoja softvéru - softvér Dizajn založený na komponentoch, dedenie, podtypovanie, late binding, Generické programovanie

Wrappers

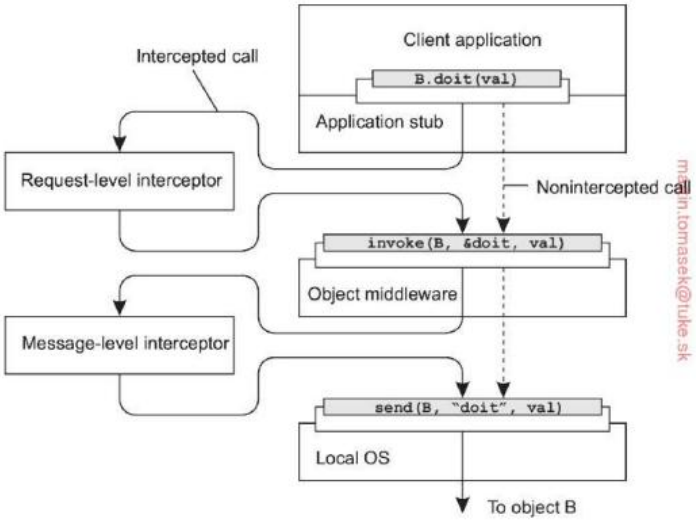
Špeciálny komponent (objektový adaptér alebo broker) s rozhraním





Interceptors

Prispôsobenie middlewaru potrebám aplikácie, Podporované v mnohých objektovo orientovaných distribuovaných systémoch



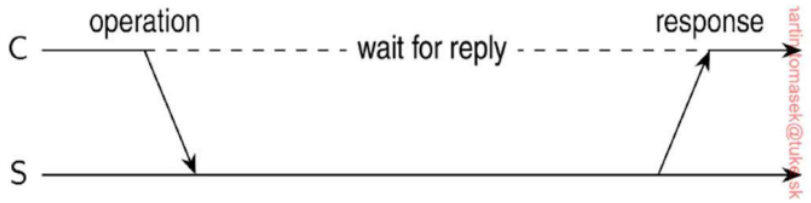
Architektúra systému

Skutočná organizácia a umiestnenie komponentov v distribuovanom systému

- **Vrstvené systémy (model klient-server)**
  - Jednoduchá architektúra klient-server,
  - Viacvrstvové architektúry
- **Symetricky distribuované systémy (model peer-to-peer)**
  - Štruktúrované systémy peer-to-peer
  - Neštruktúrované peer-to-peer systémy
  - Hierarchické peer-to-peer systémy
- **Hybridné systémy**
  - Cloud computing
  - Edge-cloud architektúra
  - Blockchain architektúry

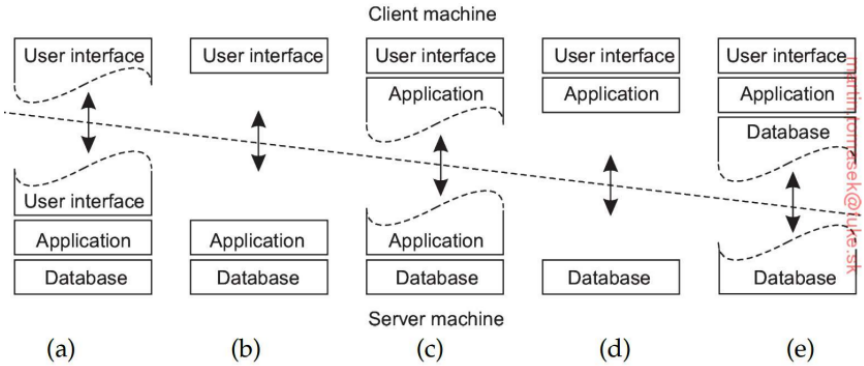
Simple client-server architecture

Všeobecná interakcia medzi klientom a serverom, Abstrakcia na pochopenie a riadenie komplexných distribuovaných systémov



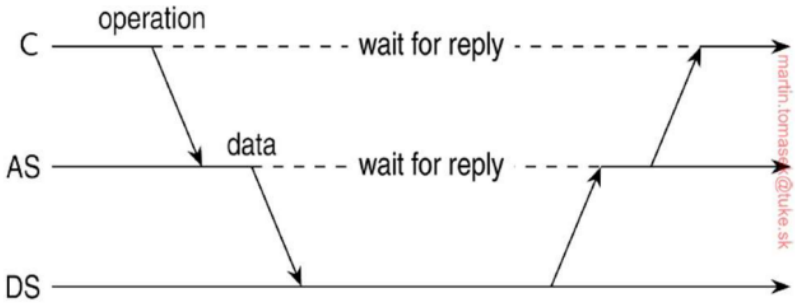
Multitiered architectures

Organizácia klient-server v dvojúrovňovej architektúre - Od tenkého klienta (a) po hrubého klienta (e)



Three-tiered client-server architecture

Server sa správa ako klient, kaskadové volania zvyšujú oneskorenia a možnú chybovosť



Peer-to-peer systems properties

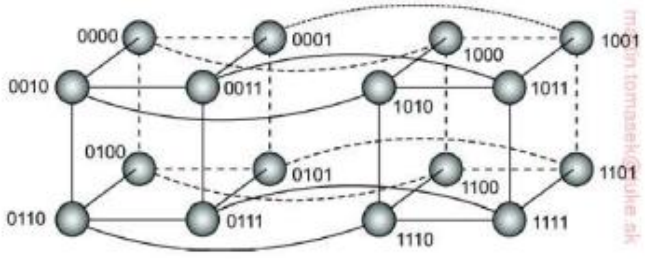
Vertikálna distribúcia (klient-server) vs. horizontálna distribúcia, Interakcia medzi procesmi je symetrická (každý uzol je klient a server)

Vlastnosti

- Žiadna centrálna koordinácia
- Žiadna centrálna databáza
- Uzol nemá globálny prehľad o systéme
- Správanie systému je kombináciou lokálnych interakcií medzi uzlami
- Uzly sú autonómne
- Uzly a pripojenia nemusia byť nevyhnutne spoľahlivé
- Všetky údaje v systéme musia byť dostupné
- **Prekryvná sieť -** Komunikačné kanály tvorené procesmi

Štruktúrovaný peer-to-peer system

- **Špecifická deterministická logická topológia** (prstenec, binárny strom, mriežka atď.) pre efektívnosť vyhľadávania údajov
- kľúč (údajová položka) = hash (hodnota údajovej položky),
- Systém ukladá páry (kľúč, hodnota), každý uzol ukladá podmnožinu kľúčov
- Systém implementuje distribuovanú hašovaciu tabuľku (DHT)



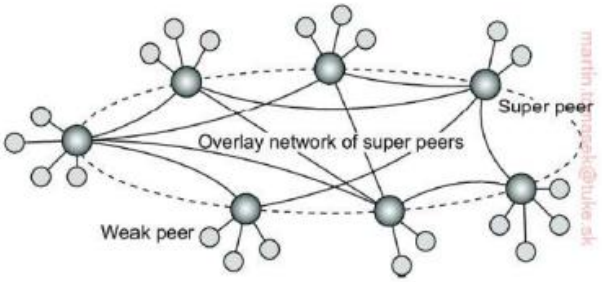
Neštruktúrovaný peer-to-peer system

- **Ad hoc zoznam susedov** – náhodný graf - Hrana medzi uzlami existuje len s určitou pravdepodobnosťou, často meniaci sa zoznam uzlov
- Vyhľadávanie sa realizuje vyhľadávaním údajov
- **Flooding** - Pošli požiadavku všetkým susedom, ak niektorý odpovie, komunikuj s ním, inak prepošli všetkým susedom (ignoruj už videné požiadavky), Veľmi drahé, musíme definovať čas životnosti (TTL) s presným maximálnym počtom skokov

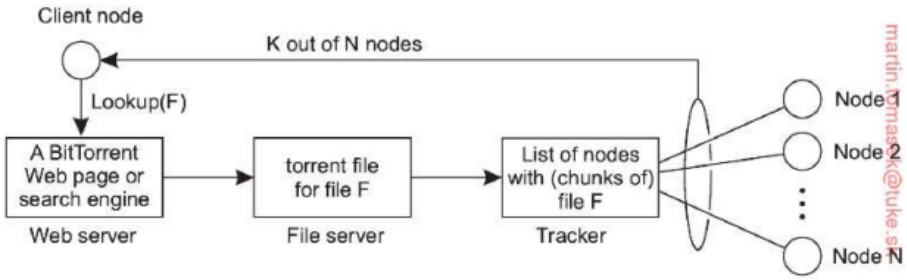
- **Random walks** - Pošli požiadavku náhodnému susedovi (alebo n náhodným susedom súčasne), Opäť je potrebné TTL a možno je možné poslať viac náhodných požiadaviek
- **Policy-based search methods** - Sledujte rovesníkov, ktorí reagujú pozitívne (preferovaní susedia)

Hierarchická sieť peer-to-peer

- **Super peer** - Špeciálny uzol, ktorý udržiava index údajových položiek (Napr. (CDN) pomocou brokerov), Malo by ísť o dlhodobý proces
- **Weak peer** - Pravidelný peer ako klient super peerovi (Napr. Skype, BitTorrent alebo Botnet)

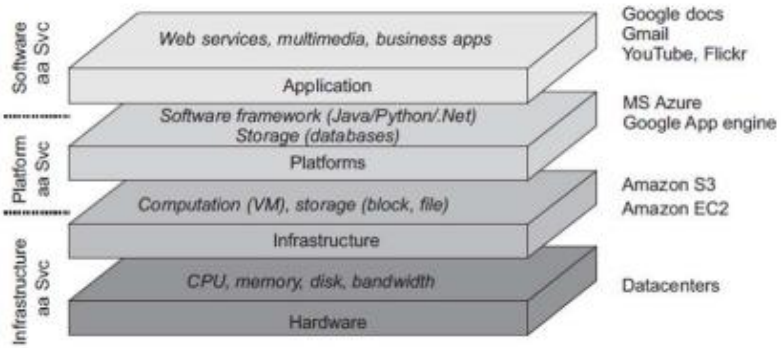


PR: bittorrent



Cloud computing

- **Infrastructure-as-a-Service** (IaaS) pokrývajúca hardvérovú a infraštruktúrnu vrstvu.
- **Platforma ako služba** (PaaS) pokrývajúca vrstvu platformy
- **Softvér ako služba** (SaaS), v ktorej sú zahrnuté ich aplikácie
- **Funkcia ako služba** (FaaS) umožňuje klientovi spúšťať kód bez toho, aby sa obťažoval dokonca aj so spustením servera na spracovanie kódu



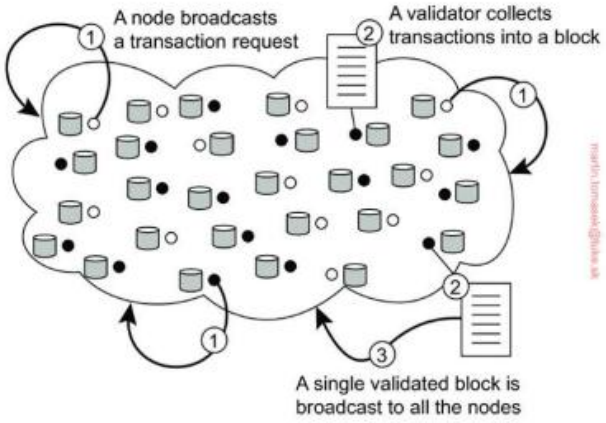
Edge-cloud architecture

Umiestnenie služby „na okraji“ siete

- **Okrajové zariadenia** – napr. zariadenia pripojené k sieti (IoT) môžu potrebovať viac než len cloudové služby
- **Okrajová infraštruktúra** – napr. hranica medzi podnikovou sieťou a poskytovateľom internetových služieb (ISP)
- **Fog computing** – napr. zdieľanie služieb prostredníctvom logicky centralizovanej infraštruktúry (cloud, lokálne dátové centrá)
- Potreba edge computingu kvôli **Latencii/šírke pásma, spoľahlivosť, bezpečnosť/súkromie**

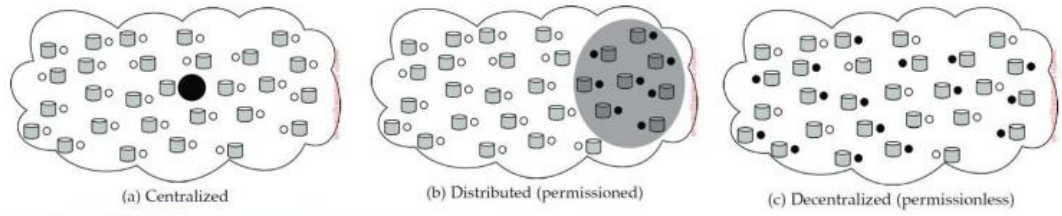
Blockchain architektúry

- Distribuovaná účtovná kniha
- Nemenný reťazec nemenných blokov údajov
- Zvyčajne registrácia transakcií a ich validácia bez dôveryhodnej tretej strany



Rozhodovanie o tom, ktorý uzol funguje ako validátor

- **Centralizované riešenie**, v ktorom overuje dôveryhodná tretia strana - transakcie ako predtým (a)
- **Distribuované riešenie, v ktorom je malá, vopred vybraná skupina** - procesy preberá rolu dôveryhodnej tretej strany (b)
- **Plne decentralizované riešenie,** v ktorom v zásade všetky zúčastnené uzly v blockchaine spoločne dosiahnu konsenzus bez akejkoľvek (distribuovanej) tretej strany (c)



Processes

Procesy

- **Operačný systém vytvára množstvo virtuálnych procesorov** na beh procesov
- **Tabuľka procesov eviduje virtuálne procesory** (CPU hodnoty registra, pamäťovú mapu, otvorené súbory, accounting, privilégiá atď.)
- **Proces je program vykonávaný na jednom virtuálnom procesore**
- Procesy nemôžu škodlivo alebo neúmyselne ovplyvniť správanie iných procesov – súbežne a konkurenčne zdieľajú CPU a iné hardvérové prostriedky
- **Transparentnosť súbežnosti je nákladná** - Vytvorenie nezávislého adresného priestoru, Prepínanie CPU medzi procesmi, Výmena procesov medzi hlavnou pamäťou a diskom

Vlákna

- Proces obsahuje viacero nezávislých vlákien - Jednoduchšie vytváranie distribuovaných aplikácií, Lepší výkon
- Nie je zabezpečený žiaden stupeň transparentnosti súbežnosti
- Kontext vlákna - Iba kontext CPU a ďalšie informácie pre správu vlákien
- Viacvláknové spracovanie vedie k zvýšeniu výkonu
- Vlákna nie sú automaticky chránené proti sebe - Viac úsilia potrebného na vývoj viacvláknových aplikácií

Vlákna na užívateľskej úrovni (knižnice vlákien)

- Lacné vytváranie a ničenie vlákien
- Prepínanie kontextov vlákien vyžaduje len niekoľko pokynov

Many-to-one model vlákien

- Viacero vlákien je mapovaných do jednej entity plánu
- Blokovanie celého procesu (a všetkých jeho vlákien)
- Napr. blokovanie vstupno-výstupnej operácie nie je efektívne

One-to-one model vlákian

- Každé vlákno je plánovateľná entita
- Operáciu vlákna vykonáva jadro (systémové volanie)
- Prepínanie kontextu vlákna je rovnako drahé ako procesy prepínania
- Typická implementácia v súčasných operačných systémoch

Vlákna v nedistribuovaných systémoch

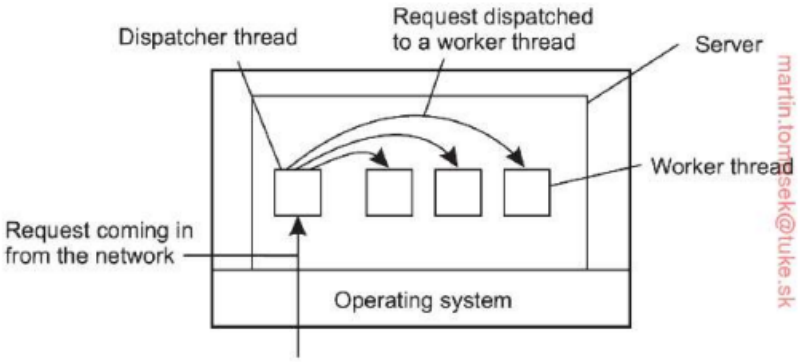
- Blokujúce systémové volanie v jednovláknových procesoch blokuje celú aplikáciu
- Vlákna s inou funkcionalitou môžu bežať nezávisle počas blokovania
- Možnosť využitia paralelizmu na multiprocesore systému - Každé vlákno je priradené inému CPU, zdieľané dáta sú uložené v zdieľanej hlavnej pamäti
- Užitočné vo veľkých aplikáciách - Zhromažďovanie spolupracujúcich programov pomocou IPC na komunikáciu (potrubia, fronty správ, zdieľaná pamäť)

Klienti s viacerými vláknami

- Oneskorenie spiatočnej cesty v rozsiahlej sieti môže byť v sekundách
- Obvyklý spôsob, ako skryť komunikačnú latenciu, je začať komunikáciu a pokračovať v niečom inom - Webový prehliadač začne zobrazovať dáta, kým ešte prichádzajú, Súčasne je možné otvoriť niekoľko spojení
- Súčasné pripojenia k rôznym replikám serverov na efektívny paralelný prenos dát a zobrazenie výsledkov

Viacvláknové **servery**

- Dispatcher číta prichádzajúce požiadavky a potom vyberie pracovné vlákno na spracovanie požiadavky
- Modely na zostavenie servera
  - Multithreading – paralelizmus, blokovanie systémových volaní
  - Jednovláknový proces – žiadny paralelizmus, blokovanie systémových volaní
  - Konečný stroj – paralelizmus, neblokujúce systémové volania



Virtualizácia

Viacvláknové procesy vykonávajú súčasné vykonávanie - V jednoprocesorovom počítači je to ilúzia simultánneho vykonávania

Virtualizácia zdrojov - Aplikačný softvér prežije svoj základný softvér a hardvér systémov

Virtualizácia pomáha spúšťať starší softvér

Virtualizácia pomáha znižovať rôznorodosť zariadení

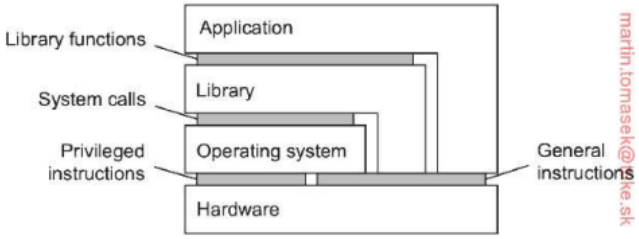
Prenosnosť a flexibilita – jednoduchá replikácia dynamického obsahu

Architektúry virtuálnych strojov

Štyri rôzne úrovne rozhraní:

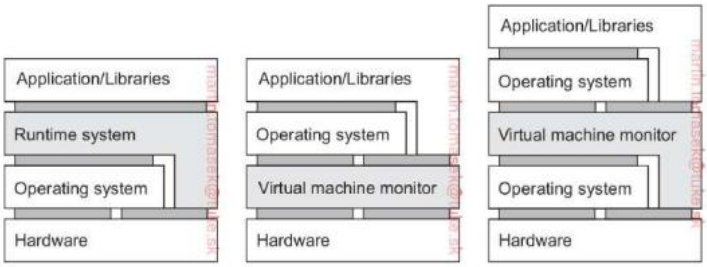
- **Všeobecné inštrukcie**, ktoré môže vyvolať akýkoľvek program

- **Privilegované inštrukcie**, ktoré môžu vyvolať iba privilegované programy (napr. operačný systém)
- **Systémové volania** ponúkané operačným systémom
- **Aplikačné programové rozhranie** (API) s volaniami knižnice (systémové volania sú skryté pomocou rozhrania API)



Prístupy virtualizácie

- **Procesový virtuálny stroj** - **runtime systém** s abstraktnými pokynmi na spúšťanie aplikácií (napr. JVM), **Iba pre jeden proces**
- **Natívny virtuálny stroj -** Virtualizácia priamo na hardvéri – **hypervízor** ponúka hardvérové inštrukcie ako rozhranie, **Beží súčasne s rôznymi programami**
- **Hosťovaný virtuálny stroj** - Hypervisor je aplikácia spustená na hostiteľskom operačnom systéme



Kontajnery

Rôzne aplikácie bežia vedľa seba, každá používa svoje vlastné prostredie bez toho, aby si všimla, že existujú aj ďalšie aplikácie s iným prostredím

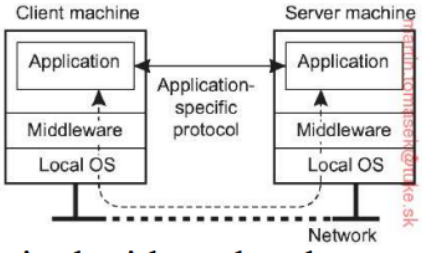
- Menné priestory (Namespaces)
- Súborový systém Union (Union file system)
- Kontrolné skupiny (Control groups)

Virtuálne stroje (ťažké) verzus kontajnery (ľahké)

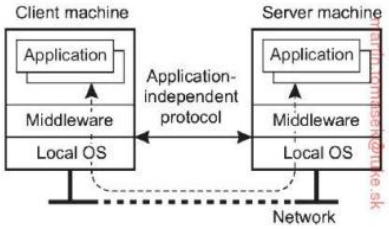
- Výkon (CPU, využitie pamäte)
- Výkon I/O operácií
- Plánovanie zdrojov

Klient

- Klienti ako sieťové používateľské rozhranie - Synchronizácia cez rozhranie na úrovni aplikácie



- Klient iba ako terminál bez lokálneho úložiska (prístup tenkého klienta)



Tenký klient

Oddelenie aplikačnej logiky od príkazov používateľského rozhrania

Typická aplikácia je **X Window System**

- X jadro – ponúka nízkoúrovňové rozhranie na ovládanie obrazovky, klávesnica a myš
- Xlib – rozhranie X jadra dostupné pre aplikácie
- X protokol – aplikačný komunikačný protokol medzi inštanciou Xlib a X jadra

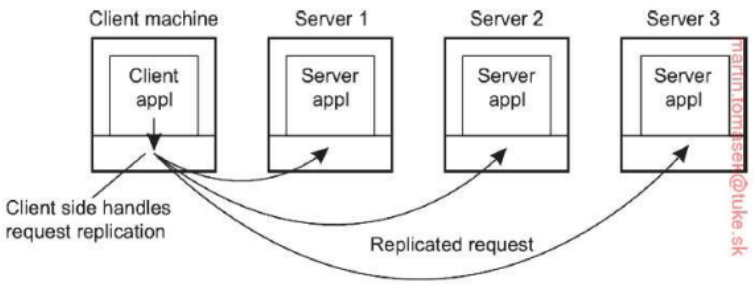
Kompletné ovládanie vzdialeného displeja (VNC, RDP)

- Používajú sa kompresné techniky, ktoré zabezpečujú dostupnosť šírky pásma

Časť úrovne spracovania a údajov je na strane klienta (Bankomaty, POS systémy, čítačky čiarových kódov, TV STB atď.)

Transparentnosť distribúcie

- Generovanie klientskeho stubu z definície rozhrania toho, čo má server ponúkať
- Stub poskytuje rovnaké rozhranie ako server, ale skrýva rozdiely v architektúry strojov a skutočnej komunikácie



Server

Čaká na prichádzajúce požiadavky od klientov, stará sa o ne a čaká na ďalšiu prichádzajúcu požiadavku

- **Iteračný server** – čakanie, spracovanie a vrátenie požiadavky prebieha v jednom vlákne
- **Súbežný server** – spracovanie požiadavky prebieha v samostatnom vlákne od čakania

Pripojenie k serveru

- **Koncový bod** – port, na ktorom beží server a klienti ho kontaktujú
- **Žiadny koncový bod** – koncový bod je dynamicky priradený, klient musí najprv vyhľadať koncový bod
- **Superserver** – počúva každý koncový bod, rozdvojuje špecifický proces na spracovanie požiadavky

Prerušenia komunikácie

- Reštartovanie klienta (nie je efektívne)
- Out-of-band Dátová správa sa číta v samostatnom koncovom bode alebo v rámci prenášaných dát

Bezstavový (Stateless) server

- **Neuchováva informácie o stave svojich klientov** - Bežné pre webové aplikácie
- Server udržiava informácie o svojich klientoch (logs) - Keď sa informácie stratia, ponúkaná služba sa nepreruší
- **Mäkký stav - Obmedzený čas na spracovanie stavu na serveri** (napr. autentifikačný token), Po uplynutí platnosti sa stav zahodí a server sa vráti k predvolenému správaniu
- Súbory cookie - Keď server nemôže udržiavať stav, klient (webový prehliadač) si ponechá
- informácie pre server
- **Veľmi jednoduché škálovanie a replikácia serverov** - Typický implementáčný problém pre cloudové aplikácie

Stavový (Stateful) server

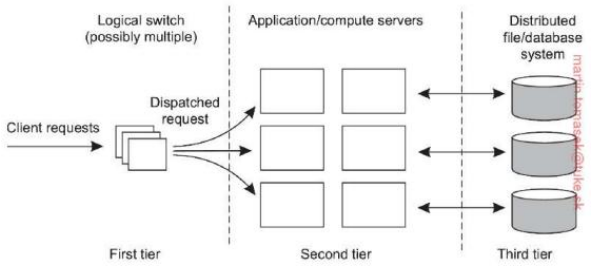
- **Udržiava trvalé informácie o svojich klientoch** - Bežné pre súborové servery
- **Vo všeobecnosti zlyhania servera nemôžu zaručiť konzistenciu** - **Server sa musí vrátiť do stavu tesne pred pádom**
- Stav relácie - Operácie jedného používateľa udržiavané po určitú dobu, ale nie na neurčito
- Trvalé - Informácie uchovávané v databázach, Odolnosť voči chybám je dôležitou otázkou pre dizajn servera

• Bezstavové vs. stavové

- Nemalo by mať vplyv na poskytovanú službu
- Je možné použiť súbory cookie
- Dôležité pre škálovateľnosť servera

Serverové klastre

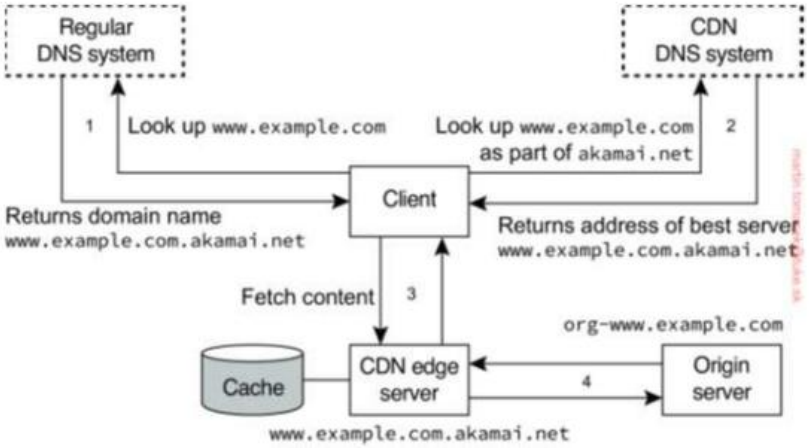
- Zbierka strojov pripojených cez sieť, kde každý stroj beží na jednom alebo viacerých serveroch
- Trojstupňová organizácia klastrov



- Logický spínač, Prepínač transportnej vrstvy, Distribúcia požiadaviek s ohľadom na obsah

Wide-area clusters

- Klastre dátových centier (napr. Google Cloud, AWS, MS Azure) - Ponuka dáta v blízkosti klienta (Použitie v CDN apps)
- Presmerovanie - Presmerovanie DNS, Presmerovanie http

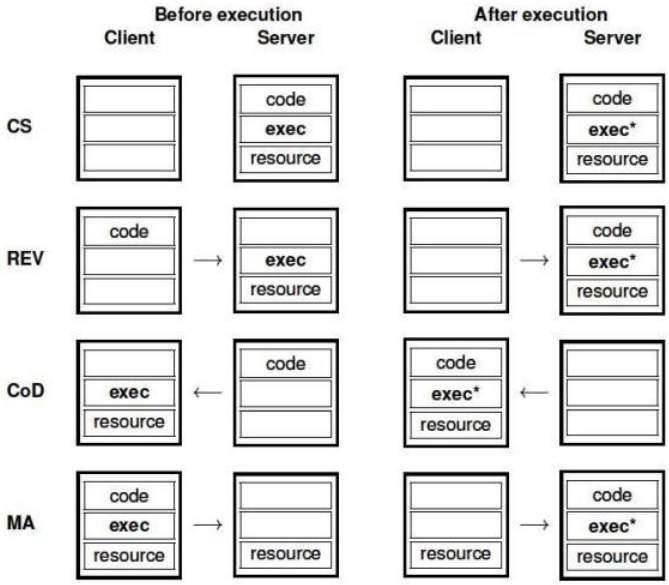


Migrácia kódu

- Komunikácia nie je obmedzená len na odovzdávanie dát – odovzdávanie programov a vzdialené spúšťanie môže zjednodušiť distribuovaný systém
- Dôvody migrácie kódu
- Výkon – rozloženie záťaže, využívanie paralelizmu, Súkromie a bezpečnosť, Flexibilita

Modely pre migráciu kódu

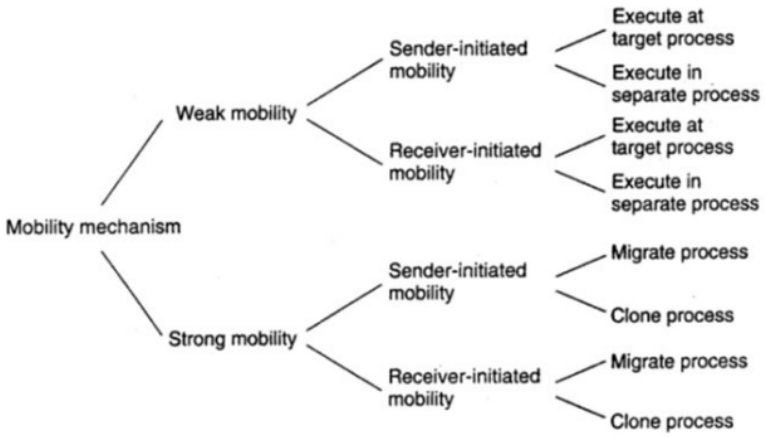
- Rámec procesu
  - Segment kódu
  - Segment zdrojov
  - Segment vykonávania
- Modely
  - Klient-server (CS)
  - Vzdialené vyhodnotenie (RE)
  - Kód na požiadanie (CoD)
  - Mobilný agent (MA)



Mechanizmy migrácie kódu

Weak mobility - Prenáša sa iba segment kódu

Strong mobility – vykonávací segment sa tiež prenáša



Migrácia a zdroje

Viazanie (Bindings)

- **Väzba na identifikátor** – napr. proces používa URL
- **Viazanosť hodnotou** – napr. miestne prístupné knižnice
- **Viazanie podľa typu** – napr. odkazy na lokálne zariadenia

Premiestnenie zdrojov (reallocation)

- **Neviazané zdroje** – napr. dáta alebo súbory priradené, ľahko migrované
- **Upevnené zdroje** – napr. lokálnych databáz, migrácia závisí od jej implementácie
- **Pevné zdroje** – viazané na špecifické prostredie stroja, nemôžu byť presunuté

Migrácia v heterogénnych systémoch

- **Migrácia celého prostredia**
- **Migrácia virtuálnych strojov**
  - Vloženie pamäťových stránok do nového zariadenia a opätovné odoslanie tých neskôr upravených počas procesu migrácie
  - Zastavenie aktuálneho virtuálneho počítača; migrujte pamäť a spustí nový virtuálny stroj
  - Nechať nový virtuálny stroj stiahnuť nové stránky podľa potreby, nechať procesy na novom virtuálnom stroji spustené a kopírovať stránky pamäte na požiadanie

Communication

OSI model

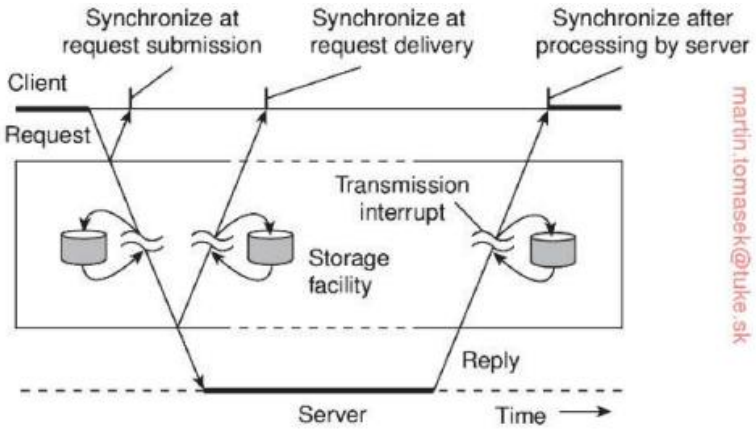
- **Fyzická vrstva** - Štandardizácia elektrických, mechanických a signalizačných rozhraní, Posielanie bitov
- **Vrstva dátového spojenia** - Zoskupuje bitov do rámcov, Kontrolný súčet na kontrolu správnosti prevodu
- **Úroveň siete** - Smeruje správu od odosielateľa k príjemcovi (IP bez pripojenia)
- **Transportné protokoly** - Správa z aplikačnej úrovne je rozdelená na pakety vhodné na prenos s nízkoúrovňovými protokolmi, Vytvorí správu pre aplikačnú úroveň z prijatých paketov, Spoľahlivý prenos - Pakety prichádzajú v správnom poradí (sieť orientovaná na spojenie), Rýchly prenos - Pakety prichádzajú v rôznom poradí (siete bez spojenia), Vrstva je zodpovedná za zostavenie správy; TCP, UDP, (S)RTP, SCTP
- **Vrstva relácie** - Dialógové ovládanie, Sledujeme, ktorá strana práve hovorí, Synchronizačné zariadenia, Dôležité pre vývoj middlewarových riešení
- **Prezentačná vrstva** - Štruktúrovanie správ, Prijímač môže byť upozornený, že správa obsahuje špecifický formát
- **Aplikačná vrstva** - Štandardné sieťové aplikačné protokoly (SMTP, FTP, HTTP, Telnet), Všetky ostatné aplikácie a protokoly (ktoré sa nehodia na nižšie úrovne), Chýbajúci rozdiel medzi aplikáciami a aplikačne-špecifickými protokoly, Všetky distribuované systémy sú len aplikácie

Middlevérové protokoly

- Middleware je aplikácia (väčšinou v aplikačnej vrstve), ktorá obsahuje protokoly na všeobecné použitie (Napr. protokoly vzdialeného volania procedúr, autentifikačné protokoly, autorizačné protokoly, distribuované uzamykacie protokoly)

Typy komunikácie

Middleware ako sprostredkovateľská služba na aplikačnej úrovni komunikácie



Persistent communication

- Prenášaná **správa je uložená v komunikačnom middleware pokiaľ nie je doručená**
- Odosielateľ nemusí po odoslaní pokračovať v komunikácii

- Pri odoslaní správy nie je potrebné spustiť prijímač

Transient communication

- Správa je **uložená v komunikačnom rozhraní iba počas doby keď odosielateľ a prijímateľ sú zapnutý**
- **Keď dôjde k prerušeniu prenosu, správa sa zahodí**

Asynchrónna komunikácia

- **Odosielateľ pokračuje ihneď po odoslaní správy**
- **Správa je (dočasne) uložená middleware**

Synchrónna komunikácia

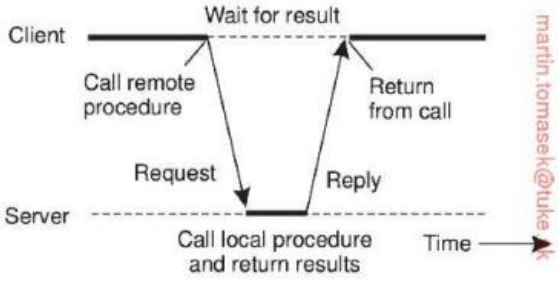
- **Odosielateľ je zablokovaný, kým nie je známe, že jeho žiadosť bola prijatá**
- Odosielateľ A môže byť zablokovaný, pokiaľ mu middleware neoznámi, že môže začať prenos
- Odosielateľ sa môže synchronizovať, kým nebude žiadosť doručená na príjemcu
- Odosielateľ môže počkať, kým sa žiadosť úplne nespracuje a odpoveď je prijatá

Remote procedure call (RPC)

- **Explicitná výmena správ medzi procesmi nezakrýva komunikáciu**
- **Základná myšlienka komunikácie:** Stroj A volá procedúru na stroji B, Proces v A je pozastavený a vykonávanie prebieha na stroji B, Informácie od volajúceho k volanému sa prenesú v parametroch a vrátia sa vo výsledku procedúry, Programátor nevidí prebiehajúcu komunikáciu

Server implementuje procedúru append

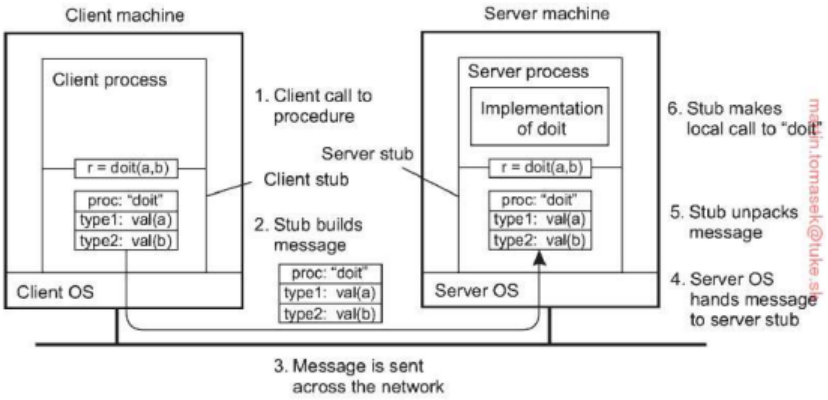
- Klient implementuje stub procedúry append, Zbalí parametre a odošle ich ako správu na server.
- Server implementuje pridanie procedúry stub, Prijme správu, spustí procedúru append servera a vráti sa výsledky pre klienta



Kroky RPC

1. **Procedúra klienta volá stub**
2. **Klientsky stub vytvorí správu a zavolá middleware RPC miestneho klienta**
3. **Middleware klienta RPC odošle správu do midlvéru RPC vzdialeného servera**
4. **Middleware servera RPC odovzdá správu stubu servera**
5. **Server stub rozbalí parametre a zavolá server**
6. **Server vykoná prácu a vráti výsledok do stubu**
7. **Stub servera ho zabalí do správy a zavolá middleware RPC svojho lokálneho servera**
8. **Middleware RPC servera odošle správu do middlewaru RPC klienta**

9. Middleware klienta RPC odovzdá správu klientovi
10. Stub klienta rozbalí výsledky a vráti sa klientovi

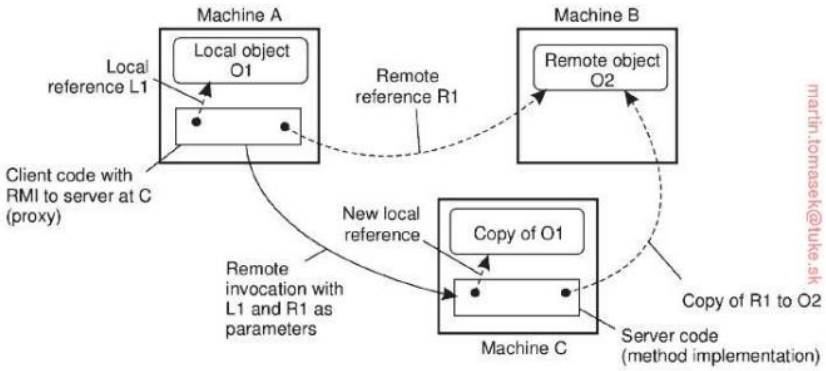


Odovzdávanie parametrov

- **Zoraďovanie parametrov** (serializácia)
- **Vstupná správa** - Názov alebo číslo procedúry, Dátová reprezentácia hodnôt parametrov
- **Výstup správy** - Dátová reprezentácia výslednej hodnoty
- **Odovzdávanie hodnôt** - Reprezentácia dát sa môže líšiť na rôznych HW architektúrach, napr. little endian (Intel) vs. big endian (starší ARM alebo sieťový prenos)
- **Odovzdávanie referencií** - Obsah odkazovanej pamäte je odovzdaný (dĺžka poľa a všetky hodnoty poľa) • Jednoduché polia a štruktúry sú jednoduché... dynamické dátové typy (napr. graf)? - Vzdialená správa referencií

Odovzdávanie parametrov v objektovo orientovaných systémoch

- A volá C s odkazom na miestnu O1 a referenciou na vzdialenú O2 (odkazované z B)
- O1 sa skopíruje z A do C
- O2 sa odkazuje z B na C

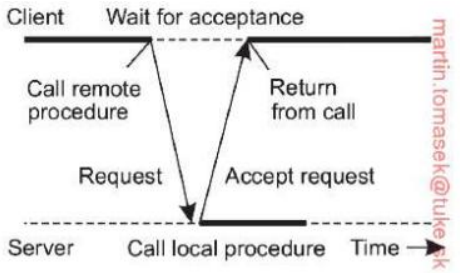


Špecifikácia parametrov a generovanie stubu

Musí sa najsť zhoda na formáte správy, reprezentácií dát, prenosových protokoloch, Interface Definition Language (IDL) a jazykovej podpore (progr. Jaz.)

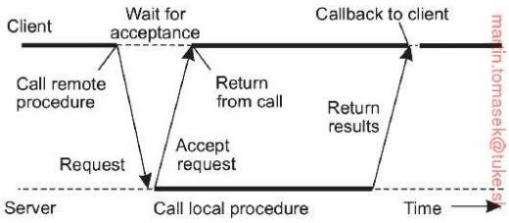
Synchrónne a Asynchrónne RPC

- **Synchrónne** volanie RPC v procedúre sa zablokuje, kým sa nevráti odpoveď, Niekedy však nemusíme čakať na výsledok, je to vhodné pre procedúry bez návratovej hodnoty, napr. prevod peňazí medzi účtami, pridanie objektu do databázy, spustenie služby, dávkové spracovanie
- Pri **asynchrónnom** RPC Server okamžite odošle odpoveď späť po prijatí požiadavky, Odpoveď je len potvrdením prijatej požiadavky klientovi



Odložené synchrónne RPC

- Niekedy môžu klienti vykonať nejakú inú operáciu počas čakania na návrat z RPC
- **Najprv klient zavolá server a pokračuje po tom, čo server potvrdí požiadavku**
- **Druhykrát server zavolá klientovi, aby doručil výsledok** (klient je prerušený, aby dostal výsledok)

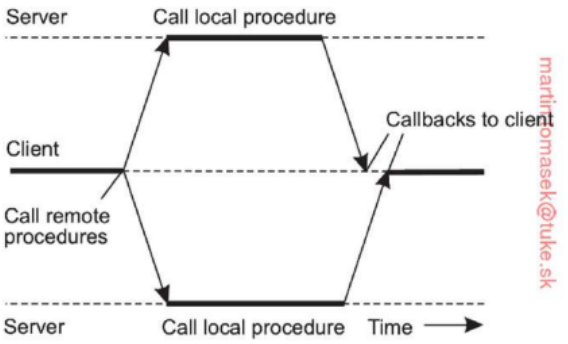


Jednosmerný RPC

- **Klient odošle požiadavku a nečaká na potvrdenie**, **Pokračuje vo vlastnom procese**, Spoľahlivosť požiadavky však nie je zaručená, Klient nevie, či je požiadavka spracovaná alebo nie, Môže však požiadať server, aby zistil, či sú výsledky k dispozícii, Server sám od seba nevolá klienta
- **Takáto asynchrónna komunikácia je veľmi typická v cloud computingu -** Slabé väzby medzi klientom a serverom, Lepšia škálovateľnosť

Multicast RPC

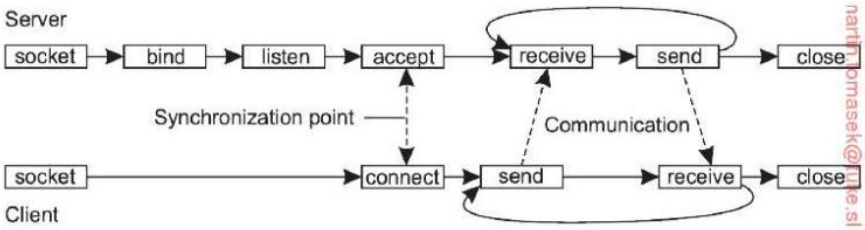
- Jednosmerné RPC možno použiť na **odoslanie požiadavky pre skupinu serverov**, Výsledok sa vráti klientovi, pri spätnom volaní
- Klient nemusí vedieť, s koľkými servermi komunikuje, taktiež sa môže rozhodnúť počkať (alebo pokračovať) na jeden alebo všetky výsledky



Message-oriented communication

- RPC skrýva komunikáciu v distribuovaných systémoch (Chceme zvýšiť transparentnosť prístupu)
- RPC je možné použiť iba vtedy, keď prijímacia strana vykonáva príkaz v rovnakom čase ako je poslaný request
- Namiesto toho môžeme použiť MOC

Berkeley sockets (BSD sockets) -> Socket je komunikačný koncový bod, na ktorý aplikácia dokáže zapisovať dáta a z ktorých sa dajú čítať prichádzajúce dáta.



ZeroMQ

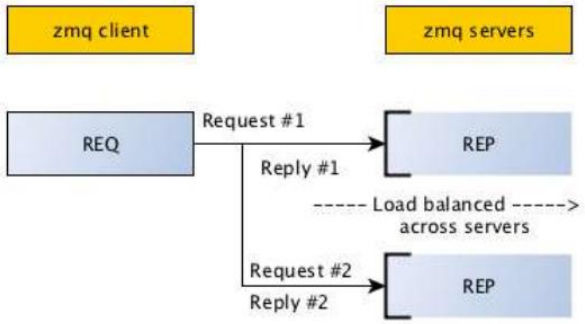
- Prenos správ prebieha cez TCP spojenia na základe skrytého používania Socketov BSD
- Komunikácia je asynchrónna, Odosielateľ bude normálne pokračovať po odoslaní správy

Vyššia úroveň komunikačných vzorcov

Request-reply pattern • Publish-subscribe pattern • Pipeline pattern

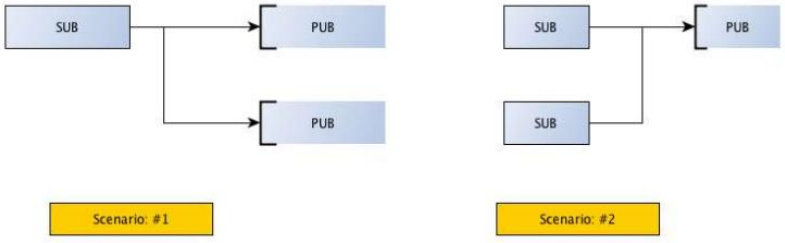
Request-reply pattern

- Klient odošle požiadavku a server odpovie na požiadavku.
- Blokovanie komunikácie na oboch stranách
- Môže sa pripojiť k mnohým serverom a je blokovaný, kým nedostane odpoveď z akéhokoľvek servera



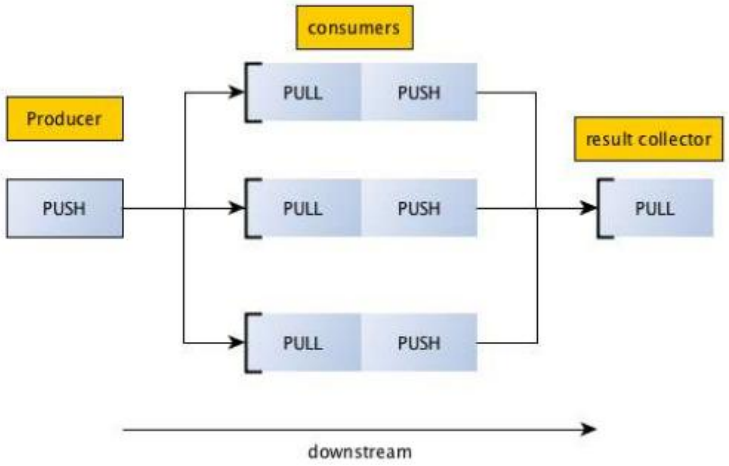
Publish-subscribe pattern

- Správy sa zverejňujú bez vedomia toho, či existuje akýkoľvek odoberateľ týchto správ



Pipeline pattern (Push-pull)

- Distribúcia správ viacerým odoberateľom, usporiadaných v pipeline
- Ekvivalent modelu producer-consumer, výsledky vypočítané konzumerom sa neposielajú proti prúdu (hore), ale po prúde (dole) inému socketu typu pull/consumer



Message-Passing Interface (MPI)

- **Message-oriented primitívy sú pre vysokovýkonné multipočítače** (Jednoduché primitíva odosielania a prijímania nestačia a sockety sú vhodné pre počítače pripojené do klasických počítačových sietí)
- **MPI bol navrhnutý pre paralelné aplikácie a využíva základnú sieť**
- Skupin procesov – groupID, processID a Prechodné komunikačné primitívy

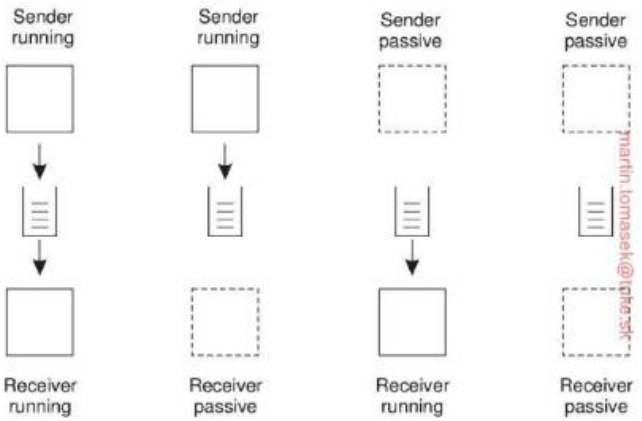
`MPI_bsend` – Append outgoing message to a local send buffer
`MPI_send` – Send a message and wait until copied to local or remote buffer
`MPI_ssend` – Send a message and wait until transmission starts
`MPI_sendrecv` – Send message and wait for reply
`MPI_isend` – Pass reference to outgoing message, and continue
`MPI_issend` – Pass reference to outgoing message, and wait until transmission starts
`MPI_recv` – Receive a message; block if there is none
`MPI_irecv` – Check if there is an incoming message, but do not block

MO perzistentná komunikácia

- Služby middlewaru orientované na správy - **Message-queuing systems or message-oriented middleware (MOM)**
- Podpora trvalej asynchrónnej komunikácie, Strednodobá pamäťová kapacita pre správy, bez toho aby vyžadovala odosielateľa alebo príjemcu počas prenosu
- Prenosy správ, ktoré môžu trvať niekoľko minút namiesto (mili)sekund

Model radenia správ - MQM

Vkladanie správ do špecifických frontov, Každá **aplikácia** má **svoj privátny front** (iba na čítanie aplikáciou), do ktorého ostatné aplikácie môžu odosielať správy, Je možné **zdieľať front pre viacero aplikácií**, Systém iba zaručuje, že správa bude nakoniec vložená do frontu príjemcu



Správa môže v zásade obsahovať akékoľvek údaje

Správy sú správne adresované - Jedinečný názov cieľového frontu pre celý systém

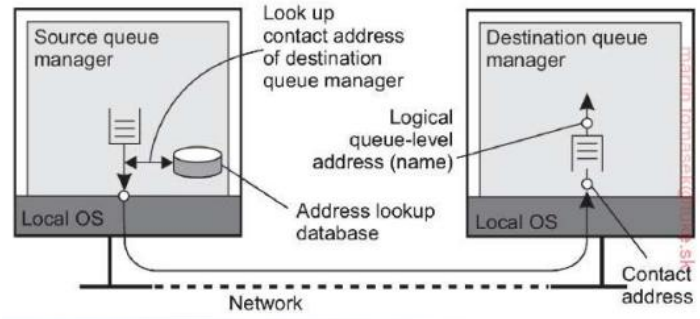
Veľkosť správy môže byť obmedzená - Fragmentácia musí byť pre aplikáciu transparentná

Základné rozhranie obsahuje len niekoľko funkcií

`put` – Append a message to a specified queue
`get` – Block until the specified queue is nonempty, and remove the first message
`poll` – Check a specified queue for messages and remove the first. Never block
`notify` – Install a handler to be called when a message is put into the specified queue

Všeobecná architektúra MQS

- **Zdrojový front** - Správy je možné zaradiť a čítať iba do lokálneho frontu, t. j. do frontu na rovnakom počítači alebo lokálnej sieti
- **Cieľový front** - Správy obsahujú špecifikáciu cieľa, Systém radenia správ je zodpovedný za prenos zo zdrojových do cieľových radov
- **Distribuovaná databáza názvov frontov** - Mapuje názvy frontov na patričné miesta v sieti

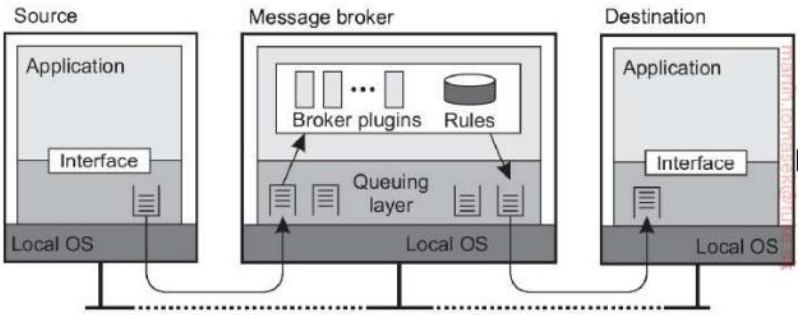


- **Správca frontov** - Spolupracuje s aplikáciou, ktorá odosiela alebo prijíma správy, komunikuje s inými správcami frontov na iných zar. cez sieť

Message brokers

- **Integrácia aplikácií do distribuovaného systému**

- **Rovnaký formát správ pre odosielateľa aj príjemcu**, pričom každá aplikácia má svoj špecifický formát
- **Publish-subscribe model (MQTT)**



Multicastová komunikácia

Odosielanie údajov viacerým prijímačom

- **Multicasting na aplikačnej úrovni -** Nastavenie komunikačných ciest v existujúcej komunikačnej štruktúre (MQTT)
  - Three organization - Jedinečná cesta medzi každým párom uzlov,
  - Mash (mesh) organization - Viacnásobné cesty medzi každým párom uzlov

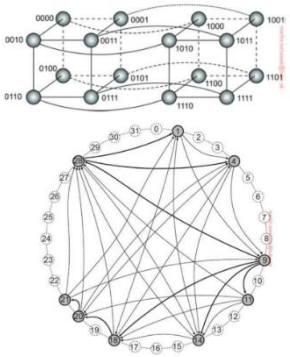
  Príklad Chord: *succ(mid)* je koreňový uzol skupiny multicast *mid*, Nový uzol zavolá *lookup(mid)*, aby sa pripojil k stredu skupiny, Multicasting je potom poskytovaný smerom na *succ(mid)*

  Meranie kvality multicastového stromu

  - **Link stress** – ako často pakety prechádzajú cez linku
  - **Stretch alebo Relative Delay Penalty (RDP)** – pomer oneskorenia medzi dvoma uzlami
  - **Stromové náklady** – minimalizácia nákladov na prepojenie

- **Flooding-based and gossip-based** - Jednoduché spôsoby (menej efektívne) multicastingu bez explicitných komunikačných ciest

  <u>Flooding-based</u> **-** Uzol prepošle správu každému susedovi, okrem toho od ktorého prijal správu, Sleduje prijaté a preposlané správy, Uzol môže ignorovať duplikáty (Erdös-Rényi graph)

  Probabilistic flooding - Zníženie počtu správ riadeným posielaním, Preposielanie sa vykonáva s danou pravdepodobnosťou, Riziko je, že s určitou pravdepodobnosťou sa správa nemusí poslať

  Iné flooding schémy

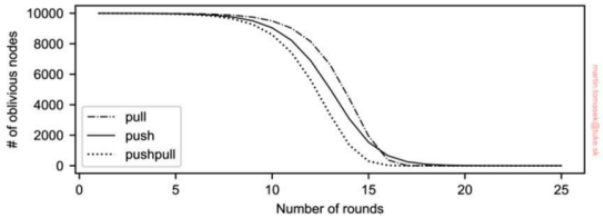
  • Podľa rozmeru - n-rozmerná hyperkocka, Preposlané do vyšších dimenzií

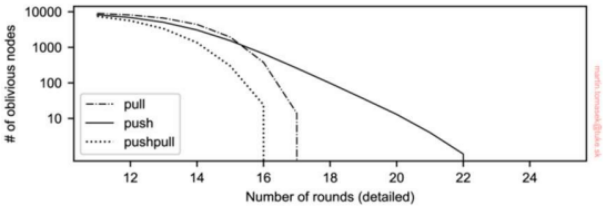
  • Prstencové - Špecifické kruhové prekrytie

<u>Šírenie údajov na základe klebiet (gossip)</u> - Epidemické správanie ako pri chorobách, šírenie informácií vo veľmi veľkých DS, napr. Šírenie aktualizácií (napr. Amazon S3)
Distribučné modely --> **Anti-entropický model** a **Rumor spreading model**

Antientropický model
Uzol P náhodne vyberie iný uzol Q a vymení si s ním aktualizácie

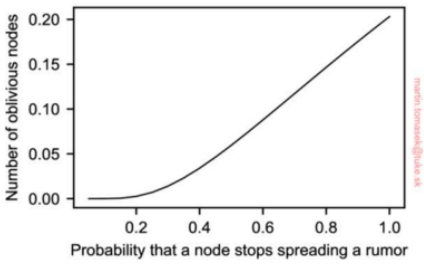
- **P vloží do Q iba svoje vlastné aktualizácie**
- **P sťahuje iba nové aktualizácie z Q**
- **P a Q si navzájom posielajú aktualizácie (prístup push-pull)**





Počet kôl na šírenie jednej aktualizácie do všetkých uzlov v prístupe push-pull je **O(log N),** kde **N je počet uzlov**
Push-based, Pull-based, Pull-pull
Počet uzlov, ktoré neboli aktualizované vyjadr. ako funkcia počtu kôl šírenia

Rumor spreading model
Ak bol P aktualizovaný pre dáta x, kontaktuje ľubovoľný iný uzol Q a pokúsi sa dotlačiť aktualizáciu na Q; P môže stratiť záujem o ďalšie šírenie aktualizácie s pravdepodobnosťou pstop
Nezaručuje aktualizáciu všetkých uzlov



Removing of data

- Šírenie vymazávania dát je náročné
- Node nakoniec dostane staré kópie
- Delete je vnímaný ako aktualizácia

Naming

Name - akákoľvek **entita DS** – resources, processes, users, connections...

Access point - poskytuje **prístup k entite**, jeho názov je **adresa**, Entita môže zmeniť prístupový bod (napr. služba sa presunie na iného hostiteľa), **Názvy nezávislý od fyzickej polohy**

Identifier - Týka sa maximálne jednej entity, Identifikátor vždy odkazuje na tú istú entitu (nikdy sa nepoužíva opakovane)

Human friendy names - identifikátory zrozumiteľé ľuďom

Name-to-address binding

- Najjednoduchšou formou je centralizovaná tabuľka vo forme párov (meno, adresa) – nestabilné pri veľkých DS
- Názov sa zvyčajne rozkladá na niekoľko častí – Rozlišovanie individuálnych častí adresy sa vykonáva niekoľkými rekurzívnymi vyhľadávaniami

Jednoduché ploché pomenovanie

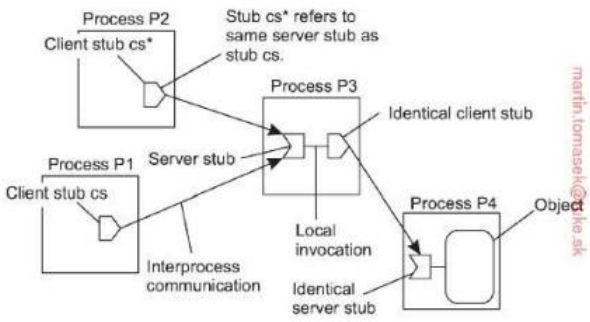
Broadcasting a multicasting

- Správa s identifikátorom sa rozošle každému zariadeniu a každý stroj si skontroluje, či má entitu
- Address Resolution Protocol (ARP)
- Neefektívne pre veľké siete

Forwarding pointers

- Keď sa entita presunie, zanechá odkaz na nové umiestnenie v starom umiestnení (klient si ho vyhľadá podľa referencií)
- Nevýhody - Reťazec ukazovateľov môže byť príliš dlhý a výpočtovo drahý, Ak sa stratí akýkoľvek ukazovateľ v reťazci, entita sa stane nedostupnou

SSP reťaze – skladá sa z client stub a server stub – transparentne s pohľadu klienta

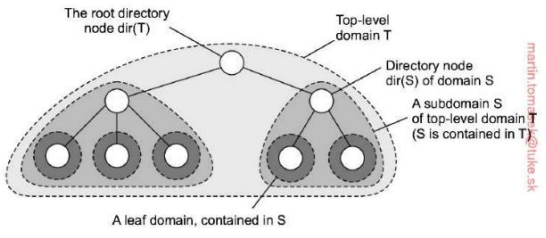


Home-based approaches

Lokácia domova určuje polohu entity, Mobile IP, Nevýhoda - zvýšenie latencie komunikácie

Distributed hash tables

- Vyhľadávanie vyžaduje O(log n) krokov
- Využívanie blízkosti siete (Chord system, ring topology)



Hierarchické prístupy

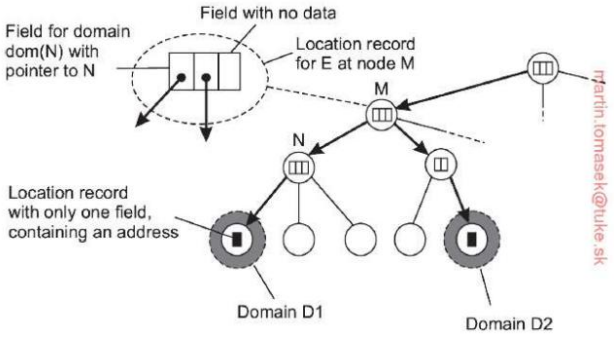
- Domény rozdelené na subdomény

Hierarchical resolution of names

Ukladanie informácií o entitách, ktoré majú adresy v rôznych listových doménach.

Vyhľadanie domény ide zdola, pokiaľ sa nenájde uzol ktorý pozná hľadanú doménu. Každý rodič je väčšia množina ako jeho deti.

Update names – funguje podobným spôsobom ako vyhľadávanie.

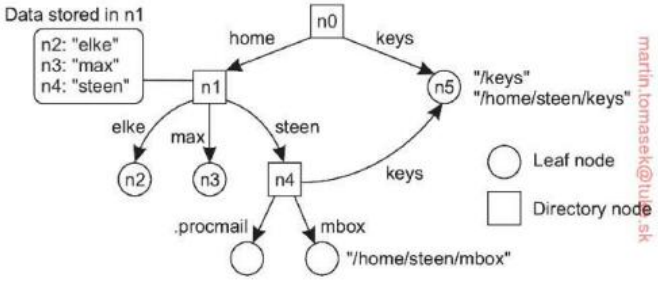


Structured naming

Mená usporiadané do **menných priestorov** (naming systems)

Názvové priestory usporiadané do **orientovaného grafu**

- Listové uzly reprezentujú **pomenované entity**
- Vnútorné uzly reprezentujú **adresáre**



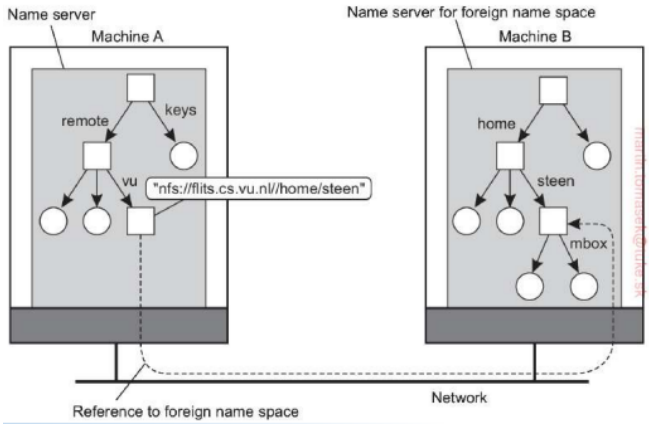
Každý musí poznať koreňový uzol

Mená sú skutočné cesty cez názvový systém (Relatívne vs. absolútne)

Remote name space

V tomto prípade sa pripojí vzdialený súborový systém

Uloží sa mounting point a k súborom zo vzdialeného systému sa pristúpy pomocou lokálnych mien

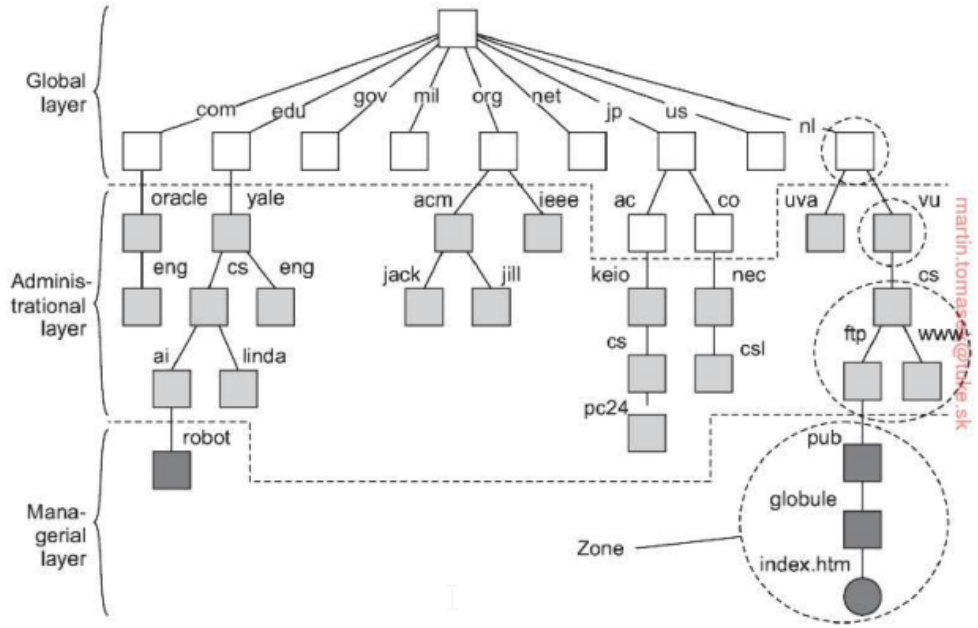


Name spaces pre rozsiahle DS sú usporiadané hierarchicky

Zväčša ide o tri logické vrstvy

- **Globálna vrstva**: uzly najvyššej úrovne (Predstavujú skupiny organizácií)
- **Administratívna vrstva**: uzly spravované jednou organizáciou
- **Manažérska vrstva**: uzly, ktoré sa často menia

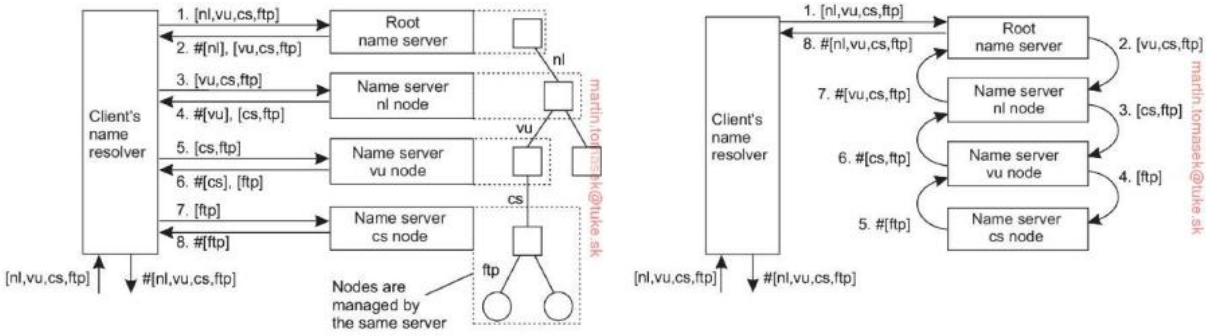
Zóna: časť menného priestoru implementovaná separátnym name serverom



Implementation of name resolution

Iteratívne - Opakovane kontaktujte name space, a tak rieši časti celého názvu

Rekurzívne - Volá svoj lokálny name space, a tak sa pohybuje v hierarchii



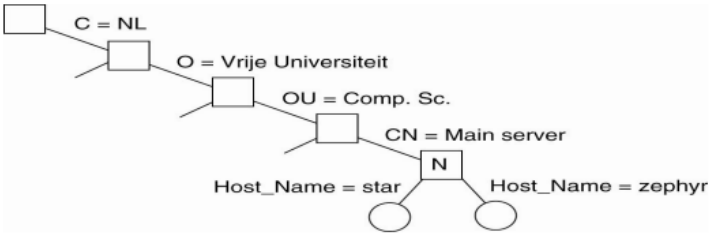
Attribute-based naming Also called **directory services**

- Ukladajú sa páry (atribút, hodnota).
- Viacnásobné hodnoty pre rovnaký atribút
- Možno kombinovať s hierarchickou štruktúrou

Lightweight Directory Access Protocol – LDAP

- Záznam ukladá množstvo párov (atribút, hodnota) entít
- Vyhľadávač dokáže nájsť entity Podľa mena, Podľa atribútu, Podľa hodnoty
- Podobné ako názvové služby (napr. DNS)

LDAP directory information tree



Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Decentralized implementations

Distribuovaný index

Pre každý atribút používame server. Na rozdelenie veľkej množiny hodnôt atribútu môže byť viac podserverov

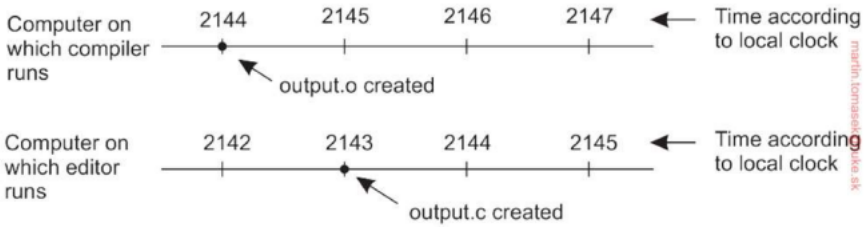
Krivky vyplňujúce priestor • Mapovanie n-rozmerného priestoru n atribútov na jednu dimenziu

Coordination

Synchronizácia v rámci jedného systému už je náročná – Semafory, Správy, Monitory

make recompiles only if output.c is newer than output.o

- Bug is fixed on developer machine (update file output.c)
- make is started on build machine (update file output.o)
- File output.o is not rebuilt! Why?



Physical clock

Nie je možné zaručiť, že všetky fyzické hodiny bežia na rovnakej frekvencií - Každý kryštál kremeňa osciluje na mierne odlišnej frekvencii - Neurčený globálny čas môže spôsobiť problémy

Universal Coordinated Time (UTC)

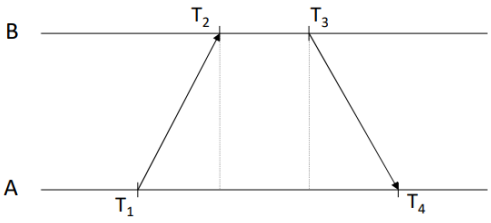
- Základ všetkých moderných časomier
- International atomic time (TAI) v Paríži
- Stále je potrebná určitá synchronizácia s astronomickým časom – napr. priestupný rok a skokové sekundy

Stanice môžu vysielať UTC pre prijímače, aby ho mohli zbierať - Teoretická presnosť je ±1 ms, ale praktická nie je lepšia ako ±10 ms

Network Time Protocol (NTP)

• A requests time of B at its own T1

- B receives request at its T2 , records
- B responds at its T3 , sending values of T2 and T3
- A receives response at its T4
- Question: What is θ = TB – TA at T4 ?



Assume transit time is approximately the same both ways

Assume that B is the time server that A wants to synchronize to

- A knows (T4– T1) from its own clock
- B reports T3 and T2 in response to NTP request
- A computes total transit time of (T4– T1) – (T3– T2)

One-way transit time is approximately half total, i.e., $\delta = \frac{(T_4-T_1)-(T_3-T_2)}{2}$

$$T_B = T_3 + \delta = T_3 + \frac{(T_4-T_1)-(T_3-T_2)}{2} = \frac{(T_4-T_1)+(T_2+T_3)}{2}$$

B's clock at T4 reads approximately

A's clock at T4 is $T_A = T_4$

Thus, difference between B and A clocks at T4 is $\theta = T_B - T_A = \frac{(T_4 - T_1) + (T_2 + T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$

Servery organizované ako **strata**

- Stratum 0 Server sa nastavuje priamo na hodiny
- Stratum 1 sa sám prispôsobí serverom Stratum 0...

Bez stratum sa servery navzájom prispôsobujú

• Ak TA je pomalý, pridajte ε k frekvencii hodín na postupne zrýchlenie

• Ak TA je rýchly, odpočítajte ε od taktu, postupné spomalenie

The Berkeley algorithm

- Časový démon sa pýta všetkých ostatných počítačov na ich hodnoty hodín
- Počítače odpovedajú
- Časový démon vypočíta priemerné oneskorenie a povie každému, ako si má nastaviť hodiny

Synchronizácia hodín v bezdrôtových sieťach

V senzorovej sieti je ťažké sa navzájom prepojiť rovnaký čas

Reference Broadcast Synchronization (RBS)

- Vnútorná synchronizácia času podobná algoritmu Berkeley
- Synchronizované sú iba prijímače
- Vzájomný relatívny posun ako priemer časových rozdielov medzi dvoma uzlami

Problém -> Čas nie je spoľahlivý spôsob synchronizácie - Používatelia pokazia hodiny (a zabudnú si nastaviť časové pásma!), Nepredvídateľné oneskorenia na internete

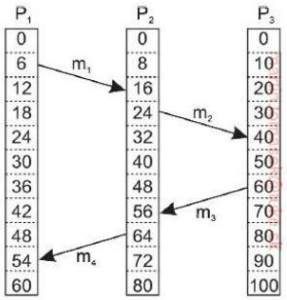
Lamportove logické hodiny

Predstavujú monotónne počítadlá (Lamportova časová logika)

• Definícia: a → b znamená, že a nastane pred b, t. j. všetky procesy súhlasia s tým, že po akcií a nastane akcia b

Tranzitívnosť Ak a -> b a b -> c, potom a -> c

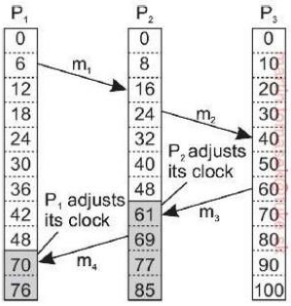
Ako synchronizovať lokálny čas (počítadlá)? -> napr. Tri procesy, každý s vlastnými hodinami



Lamportov algoritmus

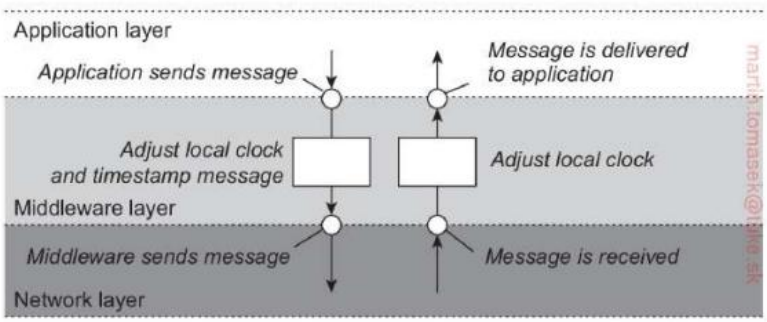
Každý proces Pi si udržiava svoje vlastné lokálne logické „hodiny" Ci

1. Pred vykonaním udalosti Pi inkrementuje lokálne počitadlo Ci (Ci ← Ci + 1)
2. Pi pootm pošle správu Pj, nastaví časovú pečiatku správy na rovnakú ako lokálne počítadlo Ci (tsm ← Ci)
3. Pj prijme správu m, upraví svoje lokálne počítadlo Cj (Cj ← max{Cj, tsm}) a vykona operáciu



Lamportov algoritmus opravuje hodiny

Logické hodiny môžu byť použité ako súčasť middlewaru



Total-ordered multicasting

- Skupina procesov, ktoré si navzájom posielajú multicastové správy, Správa je s časovou pečiatkou s aktuálnym (logickým) časom jeho odosielateľa, Odošle sa teda aj odosielateľovi,
- Proces prijme správu a zaradí ju do lokálneho poradia podľa jeho časovej pečiatky
- Prijímač odošle potvrdenie ostatným procesom (Algoritmus Lamport zabezpečuje, že časová pečiatka prijatej správy je nižšia ako časová pečiatka potvrdenia)
- Všetky procesy budú mať nakoniec rovnakú kópiu lokálneho frontu
- Správa prejde do aplikácie iba vtedy, keď je na čele frontu a bol potvrdený všetkými ostatnými procesmi (Odstráni sa z frontu a odstránia sa súvisiace potvrdenia)
- Každý proces má rovnakú kópiu frontu, všetky správy sú všade dodané v rovnakom poradí

Problém kauzality

- Lamportove logické hodiny vedú k úplne usporiadaným udalostiam
- Na druhej strane, keď sa pozrieme na časové pečiatky Lamporta, nemôže dospieť k záveru, ktoré udalosti spolu kauzálne súvisia
- Ak C(e) < C(e'), nemusí to znamenať e -> e'... Riešením sú vektorové hodiny

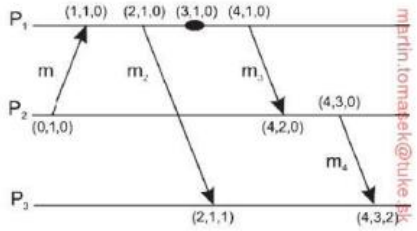
Vektorové hodiny

- VCi[i] je počet udalostí, ktoré sa vyskytli v Pi (sú to logické hodiny pre Pi)
- Ak VCi[j] = n, potom Pi vie, že v Pj sa vyskytlo n udalostí (Ide o znalosť Pi lokalneho času na Pj)

1. Pred vykonaním operácie sa na Pi inkrementuje počítadlo VCi[i]
2. Pi pošle správu m na Pj, pričom nastaví časovú značku vektora m rovnú lokálnemu vektoru VCi
3. Pj prijme správu m, upraví svoj lokálny vektor VCj a vykoná operáciu

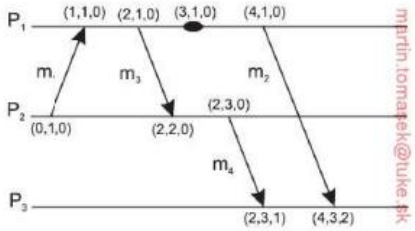
Zachytávanie potenciálnej kauzality

- tsa < tsb if and only if for all k, tsa [k] ≤ tsb [k] and there is at least one k' for which tsa [k'] < tsb [k']



m_2 may casually precede m_4

$ts_{m2} < ts_{m4}$



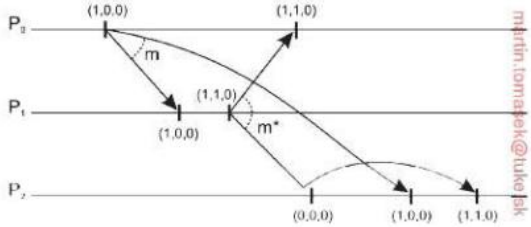
m_2 and m_4 may conflict

$ts_{m2} \not< ts_{m4}$ $ts_{m4} \not< ts_{m2}$

Presadzovanie kauzálnej komunikácie

- Hodiny sa nastavujú iba pri odosielaní správ (nenastavujú sa pri prijímaní správ)
- Keď Pi posiela správu, zvýši VCi[i] o 1
- Keď Pi prijíma správu s časovou pečiatkou tsm, upravuje iba VCi[k] na max{VCi[k], tsm[k]} pre každý k
- Keď Pj prijme m s časovou pečiatkou tsm, oneskorí doručenie do
  - tsm [i] = VCj[i] + 1 and
  - tsm [k] ≤ VCj[k] for all k ≠ i

Casual ordered multicasting je slabší ako total ordered multicasting

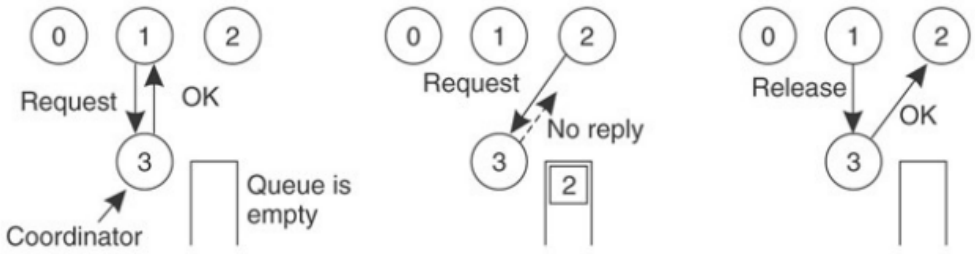


Vzájomné vylúčenie

- Zabráňte nekonzistentnému používaniu alebo nekontrolovaným aktualizáciám zdieľaných zdrojov
- Tokenový prístup
  - Posielanie špeciálnej správy medzi procesmi
  - Jednoducho sa vyhne hladovaniu
  - Problém s deadlockom (token sa stratil)
- Prístup založený na povoleniach

<u>Centralized permission approach</u>

- **Jeden proces je zvolený za koordinátora pre zdroj**
- **Všetci ostatní žiadajú o povolenie**
- Možné reakcie na získanie (**OK, Odmietnuté (opýtajte sa neskôr), Žiadne (volajúci čaká)**)
- Po uvoľnení zdroja je prístup čakajúcemu procesu udelený



Advantages

- Vzájomné vylúčenie garantované koordinátorom
- „Spravodlivé“ zdieľanie možné bez hladovania
- Jednoduchá implementácia

Disadvantages

- Jediný bod zlyhania (koordinátor havaruje)
- Koordinátor musí mať dostatočný výkon

Decentralized permissions

- **N koordinátori – spýtajte sa ich všetkých**
- **Musí mať zhodu m > N / 2**
- Výhoda - neexistuje jediný bod zlyhania
- Nevýhoda - Veľa správ, chaotický

- Probability that a coordinator fails for time interval t during lifetime T
  - $p = \frac{t}{T}$
- Probability that k of m coordinators fail
  - $P(k) = \binom{m}{k} p^k (1-p)^{m-k}$
- Voting violation when f coordinators fail
  - $f \geq 2m - N$
  - $P = \sum_{k=2m-N}^{m} P(k)$

N	m	p	Violation
8	5	3 sec/hour	$< 10^{-5}$
8	6	3 sec/hour	$< 10^{-11}$
16	9	3 sec/hour	$< 10^{-4}$
16	12	3 sec/hour	$< 10^{-21}$
32	17	3 sec/hour	$< 10^{-4}$
32	24	3 sec/hour	$< 10^{-43}$

N	m	p	Violation
8	5	30 sec/hour	$< 10^{-3}$
8	6	30 sec/hour	$< 10^{-7}$
16	9	30 sec/hour	$< 10^{-2}$
16	12	30 sec/hour	$< 10^{-13}$
32	17	30 sec/hour	$< 10^{-2}$
32	24	30 sec/hour	$< 10^{-27}$

Distribuované povolenia

- Používanie Lamportových logických hodín
- **Žiadateľ chce pracovať so súborom a žiada o povolenie všetky ostatné procesy** (vrátane seba samého), Čaká na OK odpovede od všetkých ostatných procesov
- Proces odpovedania
  - **Ak proces nemá záujem o zdroj, odpovie OK (môžeš obsadiť)**
  - **Ak práve používa daný zdroj, zaradí požiadavku, neodpovedajte**
  - **Ak má tiež záujem, odpovedá OK, ak je timestamp žiadateľa skorší, v opačnom prípade ukladá žiadosť do fronty, ak je timestamp žiadateľa neskôr**

- Two processes want to access a shared resource at the same moment
- Process 0 has the lowest timestamp, so it wins
- When process 0 is done, it sends an OK also, so 2 can now go ahead

Advantage

- No central bottleneck
- Fewer messages than decentralized

Disadvantage

- N points of failure
- Failure of one node to respond locks up system

Token system

- Organizujte procesy v logickom kruhu,
- Každý proces pozná nástupcu
- Token sa odovzdáva po kruhu
- Ak má proces záujem o zdroj, čaká na token
- Po dokončení uvoľní token
- Ak je uzol mŕtvy, proces ho preskočí.
- Odovzdá token nástupcovi mŕtveho procesu

Advantages

- Fairness, no starvation
- Recovery from crashes if token is not lost

Disadvantage

- Crash of process holding token
- Difficult to detect the token is lost; difficult to regenerate exactly one token

Election algorithms

- Ak používame jeden proces ako koordinátor pre zdieľaný zdroj, ako vyberieme tento jeden proces?
- Všetky aktuálne zapojené procesy sa spoja, aby vybrali koordinátora
- Ak koordinátor zlyhá, zvoľte nového koordinátor
- Ak sa predtým zrútený alebo izolovaný proces dostane do režimu online, budú sa musieť konať nové voľby

Bully algorithm

Assume

- **All processes know about each other**
- **Processes numbered uniquely**
- They do not know each other's state

Suppose P notices no coordinator

- Sends **election message** to **all higher numbered processes**

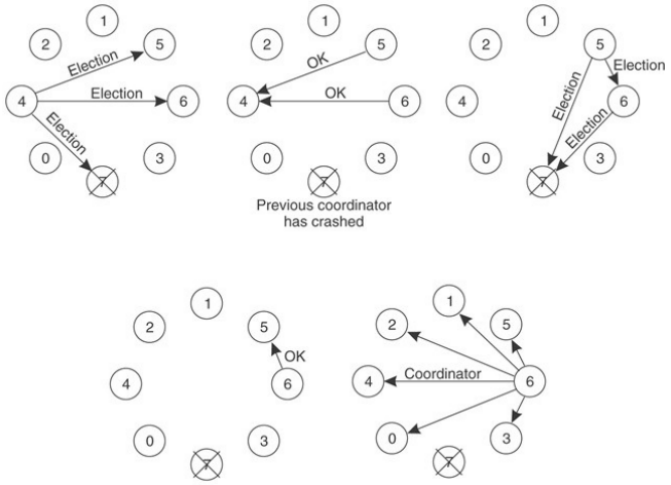
- **If no response, P takes over as coordinator**
- **If any responds, P yields**

Suppose Q receives **election message**

- **Replies OK to sender, saying it will take over**
- Sends a new **election message** to **higher numbered processes**

Repeat until only one process left standing

- **Announces victory by sending message saying that it is the coordinator**



<u>Ring algorithm</u>

All **processes organized in ring**

Suppose **P notices no coordinator**

- **Sends election message to successor** with own process number in body of message (If successor is down, skip to next running process, etc.)

Suppose **Q receives an election message**

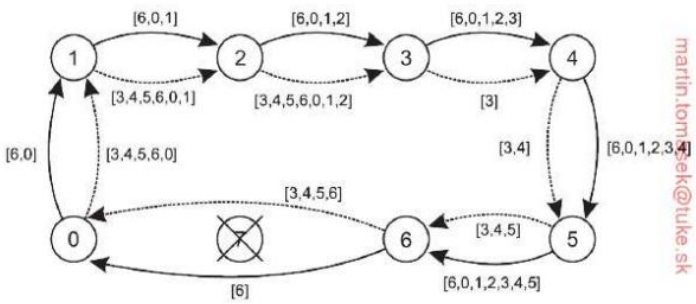
- **Adds own process number to** list in **message** body

Suppose **P receives an election message with its own process number** in body

- **Changes message to coordinator message**, preserving body
- **All processes recognize highest numbered process as new coordinator**

If multiple messages circulate, they will all contain same list of processes (eventually)

If process comes back on-line - Calls new election

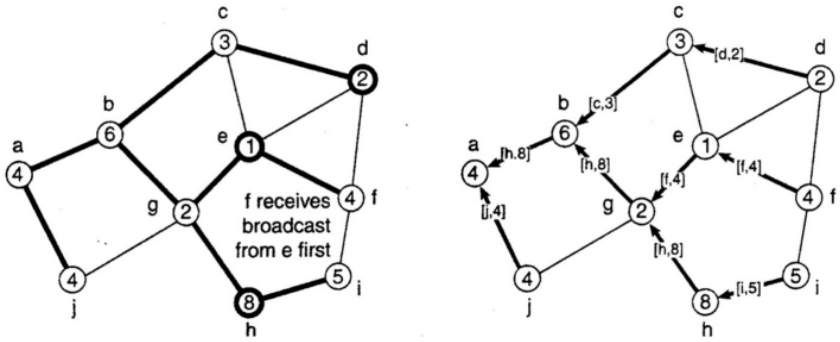


Elections in wireless environment

- Unreliable, and processes may move
- Network topology constantly changing

• Algorithm

1. Any **node starts by sending out an election message** to neighbors
2. When **a node receives** an election **message for the first time, it forwards to neighbors**, and designates the sender as its parent
3. It then **waits for responses** from its neighbors, Nodes compares IDs and puts into response higher one
4. When a node receives an election message for the **second time, it just OKs it**
5. **When message gets to the end of network nodes send back responses with highest number, so starter node will obtain number of new coordinator node**



Consistency and Replication – IN BOOK

Performance and scalability

• We generally need to ensure that all conflicting operations are done in the same order everywhere

• Conflicting operations (from the world of transactions)

• Read-write conflict – a read operation and a write operation act concurrently

• Write-write conflicts – two concurrent write operations

• Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability

• Solution is to weaken consistency requirements so that hopefully global synchronization can be avoided

Fault Tolerance

Distribuovaný systém by mal byť odolný voči chybám - Mal by byť schopný pokračovať v činnosti aj v prípade porúch

Odolnosť voči chybám súvisí so spoľahlivosťou - (Dostupnosť Spoľahlivosť Bezpečnosť Udržateľnosť)

- Dostupnosť je miera, či je systém **pripravený na okamžité použitie** - Systém je v prevádzke v každom momente.
- Spoľahlivosť je miera, či systém môže **fungovať nepretržite bez poruchy**
- Bezpečnosť je **miera bezpečnosti porúch** - Systém zlyhá, nič vážne sa nestane
- Udržateľnosť je mierou toho, **aké ľahké je naspäť opraviť systém**

Poruchy

- Systém **zlyhá (fails)**, keď nemôže pracovať na operáciách
- **Error** je súčasťou stavu systému, ktorý môže viesť k poruche (A fault (porucha) is the cause of the **error**(**chyby**))
- **Dôležité je zistiť, čo chybu spôsobilo**
- **Odolnosť voči poruchám** – systém dokáže **poskytovať služby aj v prítomnosť porúch**
- Poruchy môžu byť:
  - **Prechodný** (Transient - raz sa objaví a zmizne)
  - **Prerušovaný** (správanie sa objaví-zmizne-znova sa objaví)
  - **Trvalý** (zobrazí sa a pretrváva až do opravy)

- **Crash failure** - Server sa zastaví, ale funguje správne, kým pokial sa teda nezastaví
- **Omission failure** - Server neodpovedá na prichádzajúce požiadavky (Vynechanie prijatia, Vynechanie odoslania)
- **Timing failure** - Odpoveď servera leží mimo určeného časového intervalu.
- **Response failure** - Odpoveď servera je nesprávna, Hodnota odpovede je nesprávna, State transiction failure – Server sa odchyľuje od správneho toku (ako by mal bežať)
- **Arbitrary failure (Byzantine failures)** - Server môže produkovať ľubovoľné odpovede v ľubovoľných časoch

Process P no longer perceives any actions from process Q. Can Pconclude that Q has indeed come to halt?

• Asynchronous system – no

• Synchronous system – yes

• Fail-stop failures • Crash can be reliably detected (e.g. worst-case delay on responses)

• Fail-noisy failures • Crash can be eventually detected

• Fail-silent failures • Cannot distinguish crash failures from omission failures

• Fail-safe failures • Failures cannot do any harm

• Fail-arbitrary failures • Failures are unobservable to being harmful

Failure masking

Redundancia je kľúčovou technikou na skrytie zlyhaní

Typy redundancie

- **Informácie** – pridanie ďalších (kontrolných) informácií – error correction codes
- **Čas** – vytrvalo vykonávať akciu, kým nebude úspešná - Transakcie
- **Fyzické** – pridanie ďalších komponentov (SW + HW) - Replikácia servera, replikácia procesov

Odolnosť procesov

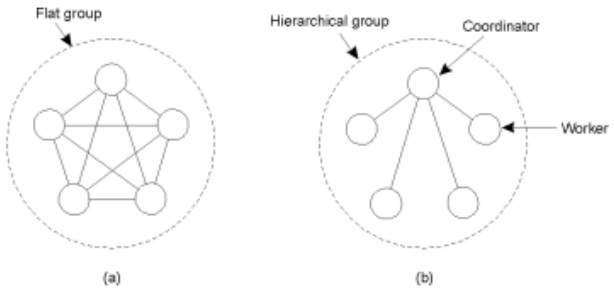
- Maskovanie zlyhaní procesov replikáciou
- Usporiadajte procesy do skupín, správa odoslaná skupine je doručené všetkým členom
- Ak niektorý člen neuspeje, mal by ho doplniť iný

Group organization and management

• Group organization

• Communication in a flat group (a)

• Communication in a simple hierarchical group (b)



Group management

• Centralized (group server) vs. distributed (multicasting)

• Synchronization of leaving and joining

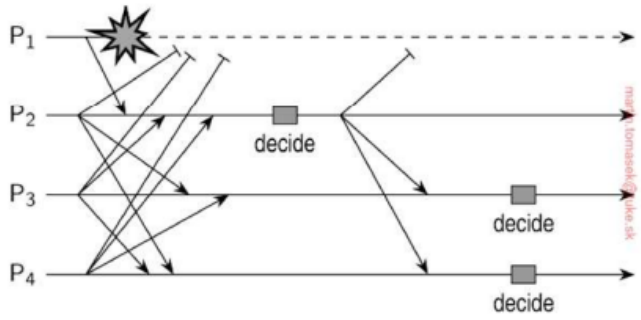
Replikácia procesu

- **Ako Replikovať proces a skupinové repliky v skupine**
- (Koľko replík vytvoríme?) **Systém je odolný voči k chybám, ak dokáže prežiť a fungovať aj keď má k chybných procesov**
  - Pri zlyhaniach - **crash** (chybný proces sa zastaví, ale funguje správne, kým sa nezastaví) - **Celkom k + 1 kópií stačí**
  - Pre **byzantské zlyhania** (chybný proces môže spôsobiť svojvoľné odpovede v ľubovoľnom čase) • Celkom ak mám 3 000 + 1 replika (potrebné sú 2 000 + 1 správných replika)

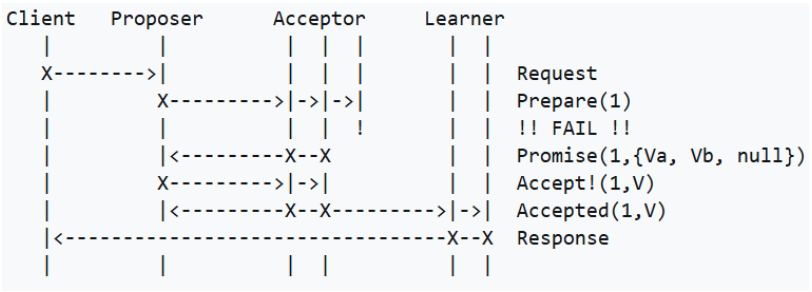
V skupine procesov odolnej voči chybám každý neporuchový proces vykonáva rovnaké príkazy v rovnakom poradí ako každý iný bezchybný proces

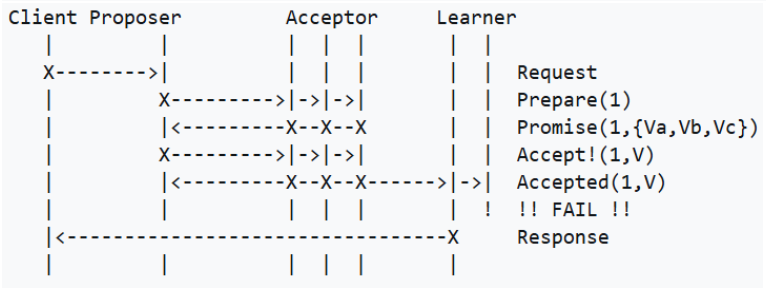
Členovia skupiny musia dosiahnuť **konsenzus** o tom, ktoré príkaz vykonať

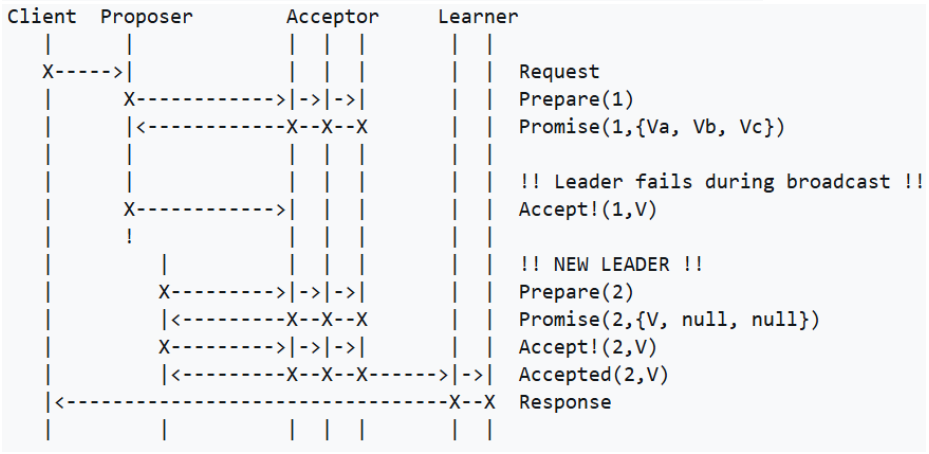
- **<u>Flood-based consensus</u>**
  - Proces Pi odošle svoj zoznam navrhovaných príkazov, ktoré videl každému procesu v skupine
  - Každý proces spája všetky prijaté navrhované príkazy a postupne vybertá príkazy, ktorý sa májú vykonať



- **<u>Paxos</u>**
  - Paxos je rodina protokolov na riešenie konsenzu procesov v sieti. Konsenzus je proces odsúhlasenia jedného výsledku medzi skupinou účastníkov. Tento problém sa stáva zložitým, keď účastníci alebo ich komunikácia môže zaznamenať zlyhania.
  - State machine prístup replikácie k distribuovaným výpočtom
  - **Proposer** vytvorí správu, ktorú nazývame **Propose**. Správa je identifikovaná **jedinečným číslom n**, ktoré musí byť **väčšie ako akékoľvek číslo, ktoré tento navrhovateľ použil v správe Propose**.
  - Keď **Acceptor** prijme správu **Propose**, Acceptor musí preskúmať identifikačné číslo n tejto správy a potom sa rozhodne:
  - **Ak je n vyššie ako každé predchádzajúce číslo** návrhu **prijaté prijímateľom** (od ktoréhokoľvek navrhovateľa), prijímateľ musí vrátiť správu (nazývanú **Promise**), ktorá naznačuje, že **prijímateľ bude ignorovať všetky budúce návrhy s číslom menším alebo rovným n**. <u>Promise musí obsahovať najvyššie číslo spomedzi návrhov, ktoré prijímateľ predtým akceptoval</u>, spolu s príslušnou prijatou hodnotou.
  - Ak je **n menšie alebo rovnaké ako akékoľvek predchádzajúce číslo ponuky prijaté Acceptorom**, acceptor nemusí odpovedať a môže návrh ignorovať.
  - Ak navrhovateľ **dostane prísľuby** od **Kvóra** prijímateľov, **musí svojmu návrhu nastaviť hodnotu v**. Ak akceptujúci predtým prijali akýkoľvek návrh, pošlú svoje hodnoty proposerovi, ktorý teraz musí **nastaviť hodnotu svojho návrhu, v, na hodnotu spojenú s najvyšším číslom návrhu nahlásenýmacceptormi**, nazvime to z. Ak žiadny z acceptorov neprijal návrh do tohto bodu, **proposer si môže vybrať hodnotu, ktorú pôvodne chcel navrhnúť**, povedzme x. **Navrhovateľ pošle Kvóru acceptorov správu Accept(n, v)** s vybranou hodnotou pre svoj návrh v a číslom návrhu n.
  - Ak **prijímateľ** dostane od navrhovateľa správu Accept(n, v), **musí ju prijať vtedy a len vtedy, ak už neprisľúbil, že bude posudzovať iba návrhy s identifikátorom väčším ako n**. Ak prijímateľ nesľúbil, že bude posudzovať iba návrhy s identifikátorom väčším ako n, **mal by zaregistrovať hodnotu práve prijatej správy** ako prijatú hodnotu a **poslať správu o prijatí proposerovi a každému learnerovi**. **Learneri sa naučia určenú hodnotu až po prijatí správ o prijatí od väčšiny prijímateľov**, nie po prijatí prvej správy o prijatí.
  - **V opačnom prípade môže ignorovať správu alebo požiadavku na prijatie**.
  - **Vyžaduje 2F + 1 non-faulty procesy**

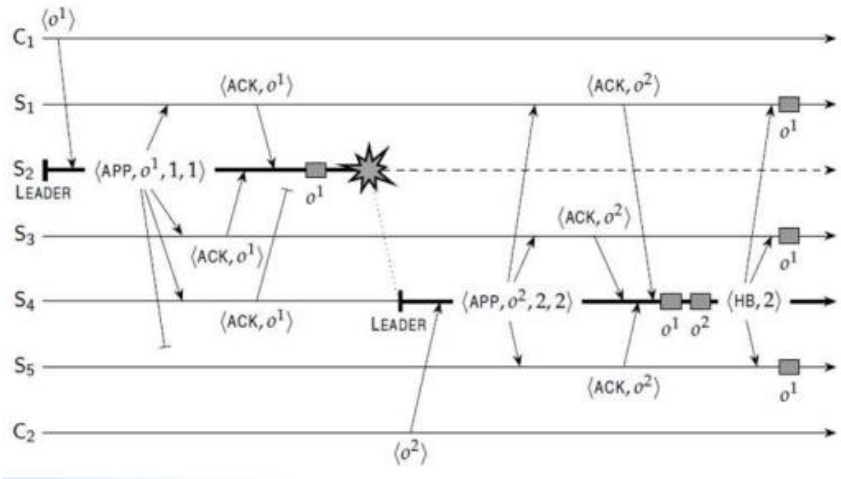




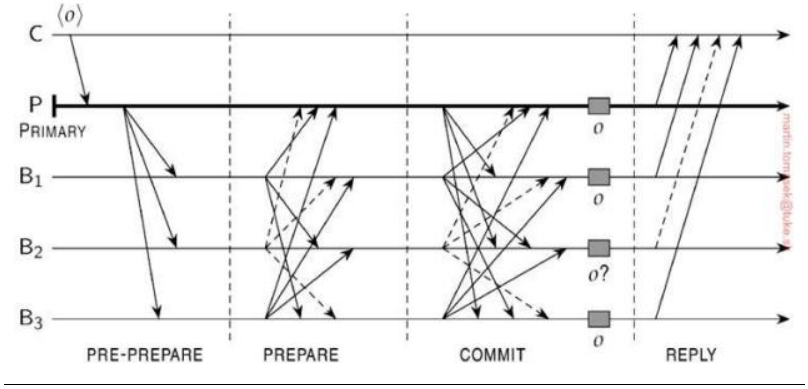


- **Raft**
  - Proces nakoniec správne dospeje k záveru, že iný proces havaroval - Vyvinutý ako reakcia na prirodzené zložitosti známeho konsenzuálneho protokolu Paxos
  - **Primárny záložný protokol, pričom jeden proces funguje ako primárny vedúci server a zálohy ako sledovacie servery**
  - **Klient vždy pošle požiadavku na operáciu vedúcemu a vedúci pošle celý svoj denník všetkým nasledovníkom** (aby sa zachoval kolektívny stav servera)
  - Leader **dostane väčšinu potvrdení**, **vykoná operáciu** a odpoveď klientovi a odošle operáciu odovzdania všetkým sledovateľom (Neodpovedajúci sledovatelia sú odstránení zo skupiny)
  - Keď **hlavný proces zlyhá**, **zvolí sa nový vodca** (leader's log je kolektívny stav servera)

  The situation when a leader crashes after executing an operation but before being able to tell other servers that the operation has been committed



- **<u>Byzantine agreement problem</u> (PBFT – Practic. Byz. Fault Tolerance)**
  - **Byzantská chyba** je stav systému, kde dôjde k poruche tak, že **rôznym pozorovateľom sú prezentované rôzne symptómy**, vrátane nedokonalých informácií o tom, či systémový komponent zlyhal.
  - **Vyžaduje 3f+1 non-faulty procesy kvoli tomu aby sa dalo rozhodnuť, dvaja sa nezhodnu ak jeden posiela T a druhy F.**
  - Výmena správ - Každý účastník pošle svoju hodnotu všetkým ostatným.
  - Hlasovanie rekurzívnou väčšinou - Každý účastník zhromažďuje všetky prijaté hodnoty a určuje väčšinu. Na dosiahnutie dohody sa používajú rekurzívne kolá hlasovania.
  - Rozhodnutie - O dohodnutej hodnote rozhoduje každý účastník na základe väčšinového konsenzu vo finálovom kole.



<u>Consensus in blockchain systems</u>

Dohoda: Všetky neporuchové uzly súhlasia s prijatím bloku transakcií a jeho pozície v blockchaine, príp súhlasia s tým, že by sa to mal zahodit

• Integrita: Každý nechybový uzol vidí rovnaké bloky prijaté transakcie a rovnaké pozície týchto blokov v blockchaine

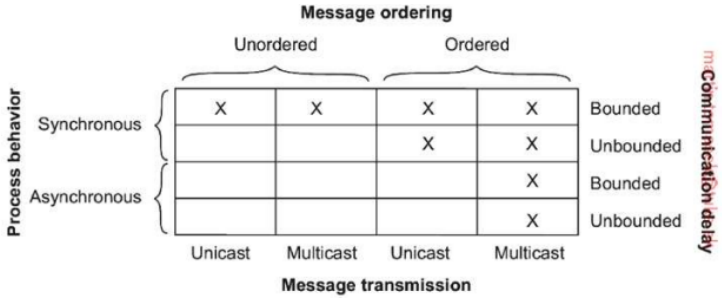
• Ukončenie: Každý nechybový uzol buď zahodí alebo akceptuje transakciu obsiahnutá v bloku, ktorá má byť súčasťou blockchainu

• Platnosť: Ak každý uzol prijme rovnaký overený blok, potom je prijatý do blockchainu

Limitations of agreement in faulty systems

Possible cases

- Synchronous versus asynchronous systems
- Communication delay is bounded or not
- Message delivery is ordered or not
- Message transmission is done through unicasting or multicasting



Failure detection

- Aktívny ping • „Si nažive?“ správu navzájom
- Pravidelná správa tlkot srdca • „Som nažive!“ správu navzájom
- Pasívne, kým neprídu správy • Len keď je dostatočná komunikácia

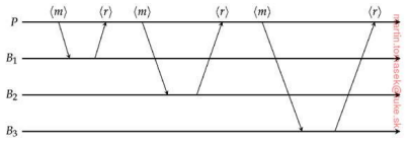
Päť rôznych tried porúch, ktoré sa môžu vyskytnúť v systémoch RPC

- Klient nemôže nájsť server
- Správa požiadavky od klienta na server sa stratí
- Po prijatí požiadavky server spadne
- Odpoveď zo servera klientovi sa stratí
- Klient po odoslaní požiadavky spadne

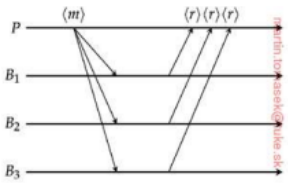
Reliable group communication

Rozdiel medzi prijímaním a doručovaním správ

- Odosielateľ posiela požiadavky, ale stále čaká na odpoveď pred odoslaním ďalšej správy,
- Žiadosti sa odosielajú paralelne, po ktorých odosielateľ čaká na odpoveď



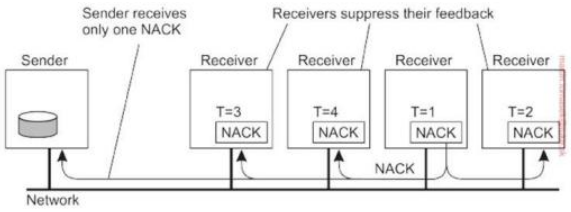
Jednoduché riešenie spoľahlivého multicastingu, keď sú známe všetky prijímače a predpokladá sa, že nezlyhajú



Scalability in reliable multicasting

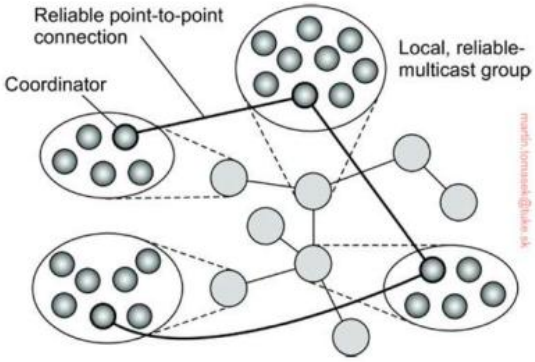
- Implózia spätnej väzby
- Potlačenie spätnej väzby

Niekoľko prijímačov naplánovalo požiadavku na retransmisiu, ale prvá požiadavka na retransmisiu vedie k potlačeniu ďalších (ostatne su ignorované)



Hierarchical reliable multicasting

- Každý miestny koordinátor prepošle správu svojim susedným koordinátorom v strome a neskôr spracuje požiadavky na retransmisiu



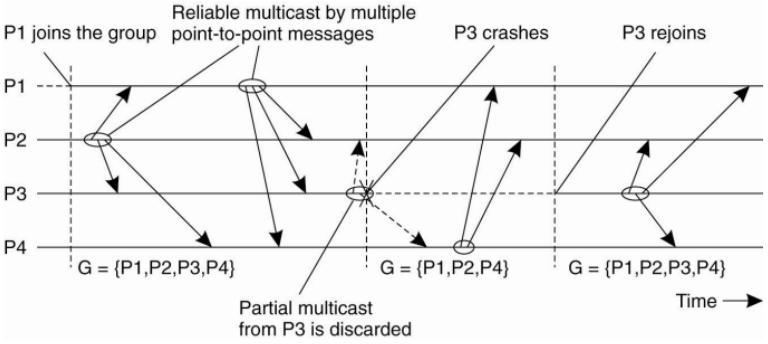
Atomic multicast

Všetky správy sa doručujú do všetkých procesov v rovnakom poradí alebo sa nedoručujú do žiadneho

Skupinové zobrazenie

- Pohľad na množinu procesov obsiahnutých v skupine, ktoré odosielať multicast správu
- Prijatú správu doručia buď všetci, alebo nikto z nich

Virtuálne synchrónne multicast - Správa je multicastovaná do skupiny a je doručená všetkým nechybným procesom v skupine alebo ignorovaná všetkými ak proces vypadne



Message ordering

- Unordered multicasts

Process P1	Process P2	Process P3
sends m1	receives m1	receives m2
sends m2	receives m2	receives m1

- FIFO-ordered multicasts

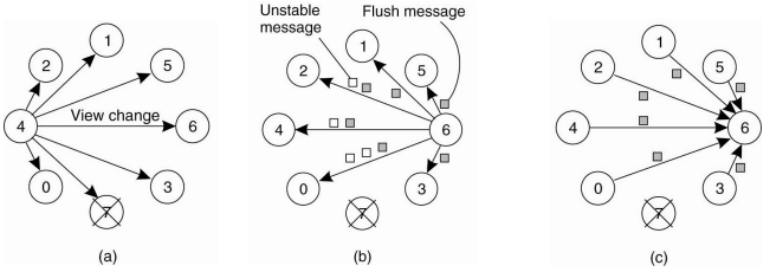
Process P1	Process P2	Process P3	Process P4
sends m1	receives m1	receives m3	sends m3
sends m2	receives m3	receives m1	sends m4
	receives m2	receives m2	
	receives m4	receives m4	

- Causally-ordered multicasts
  - Causality between different messages is preserved (vector timestamps)
- Special case in all types of orderings: Totally-ordered multicast (**atomic multicast**)
  - Messages are delivered in the same order to all in the group

Implementing virtual synchrony

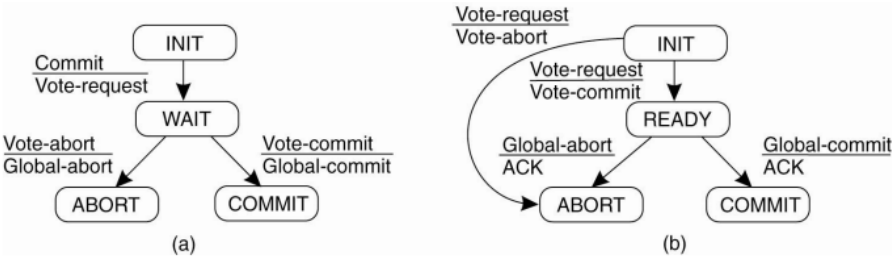
Ak nejaký proces vypadne počas toho ako vysiela a správa sa nedostane všetkým, potom prvý proces, ktorý si všimne výpadok kontaktuje všetkých o výpadku procesu. Ostatné procesy (v tomto príp. č. 6) si skontrolujú či majú nejaké unstable správy (také čo niesu doručené všetkým) a následne tieto správy rozpošlú ostatným. Flush message potvrzduje synchronicitu v sieti – už nie su žiadné unstable správy.

- Process 4 notices that process 7 has crashed and sends a view change (a)
- Process 6 sends out all its **unstable messages**, followed by a **flush message** (b)
- Process 6 installs the new view when it has received a flush message from everyone else (c)



Two-phase commit

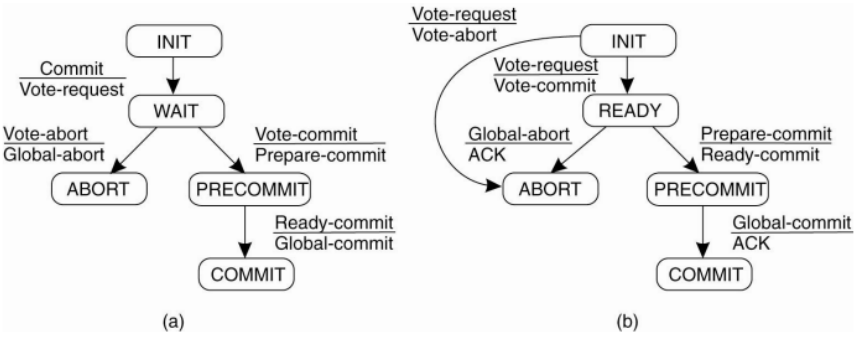
- **Koordinátor pošle VOTE_REQUEST všetkým účastníkom**
- **Účastník vráti VOTE_COMMIT, keď sa chce commitnúť lokálne alebo inak VOTE_ABORT**
- **Koordinátor zhromažďuje odpovede od všetkých účastníkov a odošle GLOBAL_COMMIT, len ak všetci chcú commitnúť. Ak jeden účastník hlasoval za prerušenie, koordinátor pošle GLOBAL_ABORT**
- **Každý účastník čaká na konečné riešenie, potom buď lokálne potvrdiť alebo zrušiť transakciu**
  - Konečnostavový automat pre koordinátora v 2PC (a)
  - Konečný automat pre účastníka (b)



Three-phase commit

Stavy koordinátora a každého účastníka spadajú do týchto dvoch podmienok

- Neexistuje stav, z ktorého je možné urobiť priamo COMMIT alebo ABORT
- Neexistuje stav, v ktorom nie je možné prijať konečné rozhodnutie, a z ktorého možno vykonať prechod do stavu COMMIT
  - The finite state machine for the coordinator in 3PC (a)
  - The finite state machine for a participant (b)



- Coordinator sends *VOTE_REQUEST* (as before)
- If all participants respond affirmatively,
  - Put PRECOMMIT state into log on stable storage
  - Send out *PREPARE_COMMIT* message to all
- After all participants acknowledge,
  - Put COMMIT state in log
  - Send out *GLOBAL_COMMIT*
- Coordinator blocked in WAIT state
  - Safe to abort transaction
- Coordinator blocked in PRECOMMIT state
  - Safe to issue *GLOBAL_COMMIT*
  - Any crashed or partitioned participants will commit when recovered
- Participant blocked in PRECOMMIT state
  - Contact others
  - Collectively decide to commit
- Participant blocked in READY state
  - Contact others
  - If any in ABORT, then abort transaction
  - If any in PRECOMMIT, the move to PRECOMMIT state
  - If all in READY state, then abort transaction

Recovery

Proces, ktorý zlyha, musí byť schopný obnoviť svoj správny stav

Spätné zotavenie

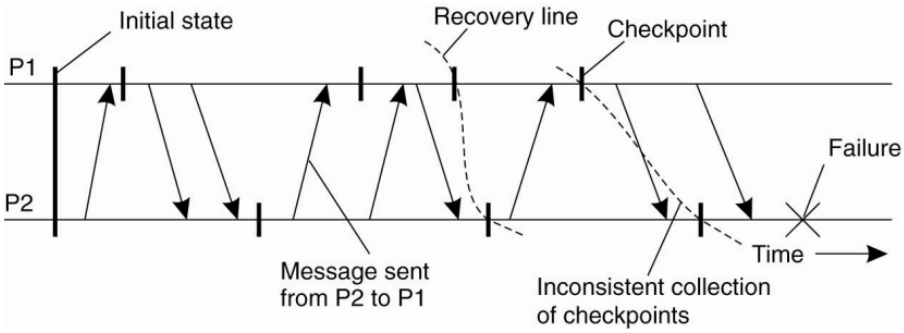
- Vráťte systém z chybného stavu späť do predchádzajúceho správneho stavu
- Stav systému je potrebné priebežne zaznamenávať – kontrolovať a potom sa obnoví, keď sa niečo pokazí (napr. Spoľahlivá komunikácia prostredníctvom opakovaného prenosu paketov)

Dopredné zotavenie

- Preniesť systém z chybného stavu do správneho nového stavu (nie predchádzajúci stav)

Na podporu obnovy je potrebné stabilné úložisko

Checkpointing - Linka obnovy z distribuovaných snímok (Najnovšia konzistentná zbierka kontrolných bodov)



Coordinated checkpointing - Procesy sú pri vytvárani checkpointu synchronizované

Dvojfázový blokovací protokol

- Koordinátor multicast CHECKPOINT_REQUEST
- Koordinátor multicastuje CHECKPOINT_DONE

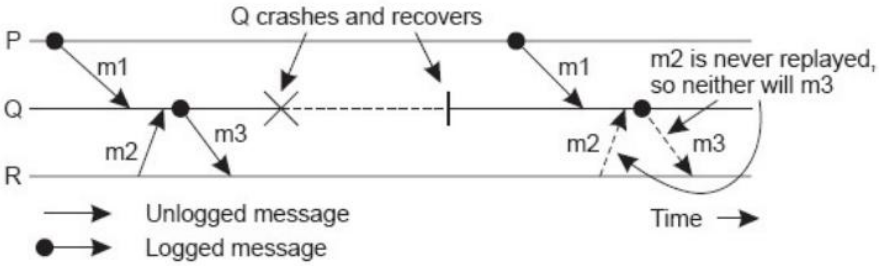
Independent checkpointing - Závislosti sa zaznamenávajú tak, že procesy môžu spoločne vrátiť do konzistentného globálneho stavu

Message logging

Checkpointing je drahý, potrebujeme iný prístup

- **Klasický prenos správ je možné opakovať a stále tak môžeme dosiahnuť globálne konzistentný stav, ale bez toho, aby sme ho museli obnoviť stav z lokálneho úložiska**
- **Logovanie správ umožňuje vytvárať scénare, ktoré sa môžu opakovane prehrávať, takže nemusíme vytvárať checkpointy**
- **Schémy zaznamenávania správ** možno charakterizovať ako **pesimistické alebo optimistické** tým, ako sa správajú k **sirotskému procesu**
- **Osirotený proces je taký, ktorý prežije pád iného procesu ale má nekonzistentný stav po zotavení sa toho druhého procesu**

Incorrect replay of messages after recovery, leading to an orphan process R:



Message-logging schemes

Predpokladáme, že každá správa m má hlavičku obsahujúcu všetky informácie potrebné na opätovné odoslanie m (odosielateľ, príjemca, poradové číslo atď.)

- **Správa sa nazýva stabilná, ak ju už nemožno stratiť**

- **Stabilnú správu možno použiť na obnovenie prehratím jej prenosu**
- Každá **správa m vedie k množine závislých procesov DEP(m), ktorým bola doručená** buď m, alebo správa kauzálne závislá od m
- **Sada COPY(m) pozostáva z procesov, ktoré majú kópiu m, ale nie v ich lokálnom stabilnom úložisku**
- Akýkoľvek proces v COPY(m) môže na požiadanie doručiť kópiu m.
- Proces Q je osamotený proces, ak existuje nestabilná správa m, takže Q je obsiahnutá v DEP(m) a každý proces v COPY(m) má havaroval
- Aby sme sa vyhli osamoteným procesom, musíme zabezpečiť, aby v prípade zlyhania všetkých procesov v COPY(m) nezostali žiadne procesy v DEP(m). To znamená, že všetky procesy v DEP(m) by mali zlyhať. Kedykoľvek sa proces stane závislým od m, zachová si kópiu m

Pessimistic logging

Pre každú nestabilnú správu m zabezpečte, aby bol maximálne jeden proces P závislý od m

- Najhoršie, čo sa môže stať, je, že P havaruje bez toho aby správu vôbec stihol logovať
- Žiadny iný proces sa nemôže stať závislým od m, pretože m bol nestabilný, takže nezostanú žiadne siroty

Optimistic logging

Hlavná práca algoritmu sa vykonáva až po havárii, nie predtým

- Ak pri niektorom m každý proces v COPY(m) zlyhal, potom každý osamotený proces v DEP(m) sa vráti späť do stavu v ktorom už nepatrí do DEP(m)
- Závislosti je potrebné explicitne sledovať, čo vytvára ťažkú implementáciu (v praxi sa viac uplatňujú pesimistic. prístupy)

Security

Model CIA

- **Confidentiality** • Informácie sa sprístupňujú iba oprávneným stranám
- **Integrity** • Zmeny v aktívach systému je možné vykonať iba v autorizovanom systéme spôsobom
- **Availability** • Všetky bezpečnostné mechanizmy musia pre informácie správne fungovať chránia a zabezpečujú, že je k dispozícii, keď je to potrebné

Security threats

- **Interception** • Unauthorized party has gained access to a service or data
- **Interruption** • Services or data become unavailable, unusable, destroyed, etc.
- **Modification** • Unauthorized changing of data or tampering with a service so that it no longer address to its original specification
- **Fabrication** • Additional data or activity are generated that would normally not exist

Security policy and security mechanism

• **Security policy** • Describe precisely which actions the entities in a system are allowed to take and which ones are prohibited

• **Security mechanism** • Mechanisms by which policy can be enforced

• **Encryption** • Transform data to something an attacker cannot understand

• **Authentication** • Verify the claimed identity of a user, client, server, host, or other entity

• **Authorization** • Check whether the entity is authorized to perform the action requested

• **Auditing** • Trace which entity accessed what, and in which way

Design issues

Bezpečnostné princípy

- Predvolené nastavenia zabezpečenia proti poruche
- Otvorený dizajn – nevyhnutné, aby každý aspekt distribuovaného systému bol otvorený na preskúmanie
- Oddelenie privilégií – Zabezpečiť, aby kritické aspekty systému nikdy nemohol plne kontrolovať len jeden subjekt
- Least privilege – Proces by mal fungovať s čo najmenším počtom privilégií (len s nevyhnutnými)
- Least common mechanism – Navrhovanie systémov tak, že tie, ktoré pracujú podobne mali by byť rovnako bezpečnostne implementované.

Layering (vrstvenie) of security mechanisms

- Závisí od dôvery klienta v to, do akej miery je služba zabezpečená na konkrétnu úroveň
- Bezpečnostné mechanizmy sú zvyčajne umiestnené na úrovni middlewaru

Trusted computing base (TCB)

- Súbor všetkých mechanizmov v systéme, ktoré sú potrebné na presadzovanie bezpečnostnej politiky, ktorej treba dôverovať

Privacy

- Controlling who gets to see what, when, and how (appropriate flow of personal info)

Cryptography

Odosielateľ zašifruje správu m na nezrozumiteľnú správu m' a prijímač následne dešifruje prijaté m' na jeho pôvodné m

Symmetric vs. asymmetric cryptosystems

- Symmetric cryptosystem P = DK (EK (P))
- Asymmetric cryptosystem P = DSK(EPK(P))

K – key for encryption and decryption

PK – public key for encryption, SK – private key for decryption

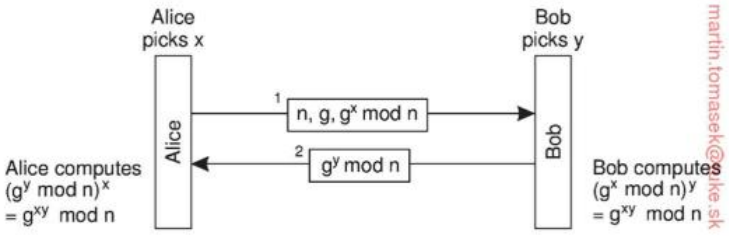
Cryptographic hash function -> h = H(m)

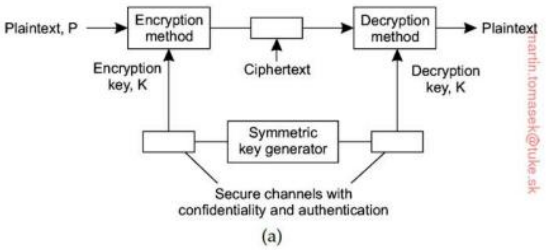
- **One-way function** - Je výpočtovo nemožné nájsť zodpovedajúci vstup m na výstup h
- **Slabá kolízna odolnosť** - Keď je zadaný vstup m, je výpočtovo nemožné nájsť iný rozdielny vstup m' ≠ m, taký že H(m') = H(m)
- **Silná odolnosť voči kolízii** - Keď je zadaná funkcia H, je výpočtovo nemožné nájsť nejakú dva rôzne vstupy m a m', takže H(m) = H(m')

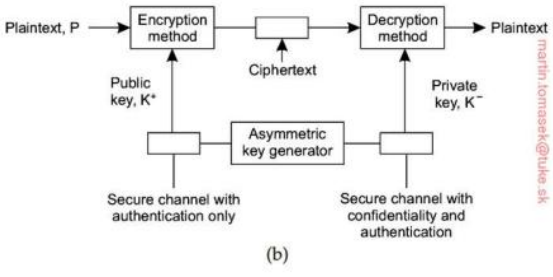
Diffie-Hellman key exchange

• n, g – large random numbers that can be public

• x, y – large random numbers that are private







Authenticated distribution of public key

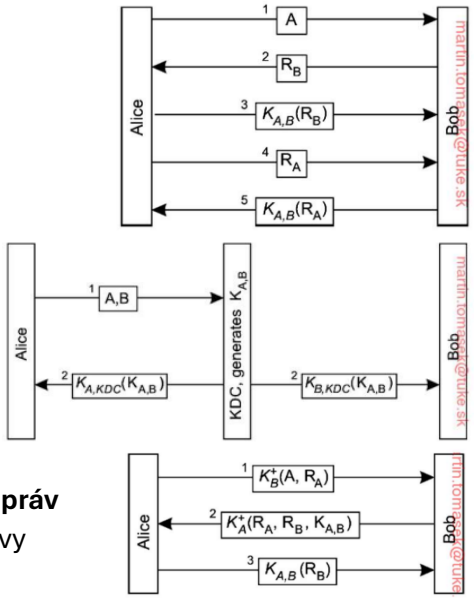
Identifikátor entity, ktorá vlastní kľúč (ten kto vydal public key nie je hocikto, je overený)

Certification authority (CA) - Podpisuje verejný kľúč, identifikátor entity a celý certifikát

Validácia certifikátu - Skontroluje platnosť verejného kľúča a identifikátora

Secure channels

Bezpečná komunikácia medzi klientmi a servermi



- **Autentifikácia komunikačných strán**
  - *KA,B – Secret key shared by A and B*
  - *KA+ – Public key of A*
  - *KA- – Private key of A*
  - *K(d) – Some data d encrypted by key K*
  - **Autentifikácia na základe zdieľaného tajného dátového kľúča**
  - **Autentifikácia pomocou centra distribúcie kľúčov**
  - **Autentifikácia pomocou kryptografie s verejným kľúčom**
- **Zabezpečenie dôvernosti a integrity posielania správ**
  - **Digitálne podpisy** - Podpísanie súhrnu správy
  - Potreba **Session keys**
  - Komunikačné strany vo všeobecnosti používajú jedinečný kľúč zdieľanej relácie na zachovanie dôvernosti
  - Keď sa kanál už nepoužíva, kľúč sa zahodí
  - Autentifikačné kľúče sa používajú čo najmenej (ťažšie ich odhaliť) • Ochrana pred útokom odpovede
  - Keď je ohrozený kľúč relácie, ovplyvní to iba jednu reláciu a nie všetky staré konverzácie
- Ochrana komunikácie v rámci skupiny serverov
  - Členovia skupiny zdieľajú rovnaký tajný kľúč na šifrovanie a dešifrovanie (ale všetci členovia musia byť dôveryhodní)

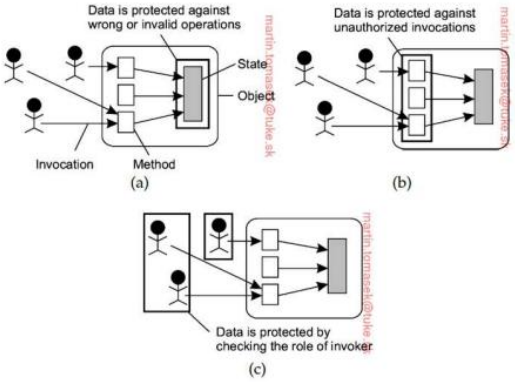
Access control policies

Všeobecný model riadiaci prístup k objektom

- Mandatory access control (MAC)
- Discretionary access control (DAC)
- Role-based access control (RBAC)
- Attribute-based access control (ABAC)

Approaches to protection

- Ochrana pred neplatnými operáciami (a)
- Ochrana pred neoprávneným vyvolaním (b)
- Ochrana pred neoprávnenými používateľmi (c)



Access control matrix - Referenčný monitor kontroluje, či je metóda m objektu o vyvolaná subjektom s uvedená v M (Access Control List (ACL))

Dôvera je uistenie sa, že entita bude vykonávať konkrétne činnosti podľa špecifického očakávania