# Elosztott rendszerek: Alapelvek és paradigmák
# Distributed Systems: Principles and Paradigms

Maarten van Steen[1] Kitlei Róbert [2]

[1]VU Amsterdam, Dept. Computer Science
[2]ELTE Informatikai Kar

## 3. rész: Folyamatok

2015. május 24.

# Tartalomjegyzék

# Szálak: bevezetés

## Alapötlet

A legtöbb hardvereszköznek létezik szoftveres megfelelője.

Processzor (CPU): Hardvereszköz, utasításokat képes sorban végrehajtani, amelyek egy megadott utasításkészletből származnak.

Szál (thread): A processzor egyfajta szoftveres megfelelője, minimális kontextussal (környezettel). Ha a szálat megállítjuk, a kontextus elmenthető és továbbfuttatáshoz visszatölthető.

Folyamat (process, task): Egy vagy több szálat összefogó nagyobb egység. Egy folyamat szálai közös memóriaterületen (címtartományon) dolgoznak, azonban különböző folyamatok nem látják egymás memóriaterületét.

## Hasonló elnevezések

Fontos: nem összekeverendő!

$$\text{stream} = \text{folyam} \neq \text{folyamat} = \text{processz} \neq \text{processzor}$$

# Kontextusváltás

## Kontextusváltás

- kontextusváltás: A másik folyamatnak/szálnak történő vezérlésátadás, illetve a megfelelő kontextusok cseréje. Így egy processzor több folyamatot/szálat is végre tud hajtani.
- Processzor kontextusa: Az utasítások végrehajtásában szerepet játszó kisszámú regiszter (elemi értéktároló) tartalma.
- Szál kontextusa: Jellemzően nem sokkal bővebb a processzorkontextusnál. A szálak közötti váltáshoz nem kell igénybe venni az operációs rendszer szolgáltatásait.
- Folyamat kontextusa: Ahhoz, hogy a régi és az új folyamat memóriaterülete elkülönüljön, a memóriavezérlő (memory management unit, MMU) tartalmának jórészét át kell írni, amire csak a kernel szintnek van joga.
  A folyamatok létrehozása, törlése és a kontextusváltás köztük sokkal költségesebb a szálakénál.

# Szálak és operációs rendszerek

## Hol legyenek a szálak?

A szálakat kezelheti az operációs rendszer, vagy tőle független szálkönyvtárak. Mindkét megközelítésnek vannak előnyei és hátrányai.

## Szálak folyamaton belül: szálkönyvtárak

- előny: Minden műveletet egyetlen folyamaton belül kezelünk, ez hatékony.
- hátrány: Az operációs rendszer számára a szál minden művelete a gazdafolyamattól érkezik $\Rightarrow$ ha a kernel blokkolja a szálat (pl. lemezművelet során), a folyamat is blokkolódik.
- hátrány: Ha a kernel nem látja a szálakat közvetlenül, hogyan tud szignálokat közvetíteni nekik?

# Szálak és operációs rendszerek

## Szálak folyamaton kívül: kernelszintű szálak

A szálkönyvtárak helyezhetőek kernelszintre is. Ekkor *minden* szálművelet rendszerhíváson keresztül érhető el.

- előny: A szálak blokkolása nem okoz problémát: a kernel be tudja ütemezni a gazdafolyamat egy másik szálát.
- előny: A szignálokat a kernel a megfelelő szálhoz tudja irányítani.
- hátrány: Mivel minden művelet a kernelt érinti, ez a hatékonyság rovására megy.

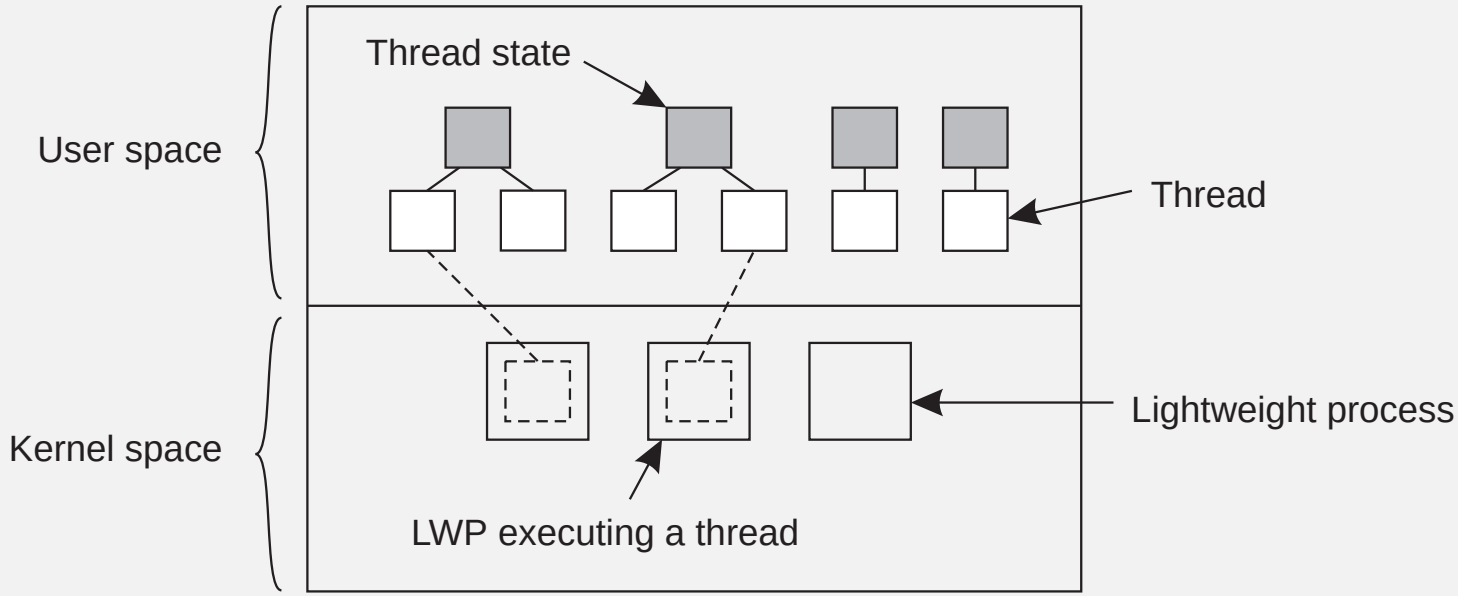
## Köztes megoldás?

Lehet-e olyan megoldást találni, ami ötvözi a fenti két megközelítés előnyeit?

# Solaris szálak

## Könnyűsúlyú folyamatok

könnyűsúlyú folyamat (lightweight process, LWP): Kernelszintű szálak, amelyek felhasználói szintű szálkezelőket futtatnak.




## Támogatottsága

A legtöbb rendszer az előző két megközelítés valamelyikét támogatja.

# Szálak a kliensoldalon

## Példa: többszálú webkliens

A hálózat késleltetésének elfedése:

- A böngésző letöltött egy oldalt, ami több másik tartalomra hivatkozik.
- Mindegyik tartalmat külön szálon tölti le, amíg a HTTP kéréseket kiszolgálják, ezek blokkolódnak.
- Amikor egy-egy fájl megérkezik, a blokkolás megszűnik, és a böngésző megjeleníti a tartalmat.

## Példa: több távoli eljáráshívás (RPC) egyszerre

- Egy kliens több távoli szolgáltatást szeretne igénybe venni. Mindegyik kérést külön szál kezeli.
- Megvárja, amíg mindegyik kérésre megérkezik a válasz.
- Ha különböző gépekre irányulnak a kérések, akár lineáris mértékű gyorsulás is elérhető így.

# Szálak a szerveroldalon

## Cél: a hatékonyság növelése

- Szálakat sokkal olcsóbb elindítani, mint folyamatokat (idő- és tárigény szempontjából egyaránt).
- Mivel egy processzor csak egy szálat tud végrehajtani, a többprocesszoros rendszerek kapacitását csak többszálú szerverek képesek kihasználni.
- A kliensekhez hasonlóan, a hálózat késleltetését lehet elfedni azzal, ha egyszerre több kérést dolgoz fel a szerver.

## Cél: a program szerkezetének javítása

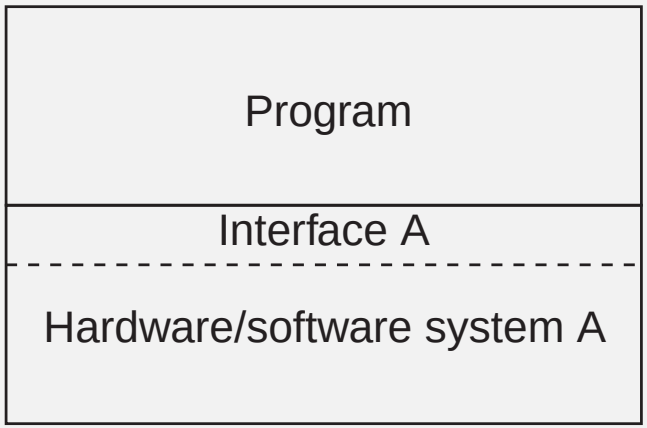
- A program jobban kezelhető lehet, ha sok egyszerű, blokkoló hívást alkalmaz, mint más szerkezet esetén. Ez némi teljesítményveszteséggel járhat.
- A többszálú programok sokszor kisebbek és könnyebben érthetőek, mert jobban átlátható, merre halad a vezérlés.

# Virtualizáció

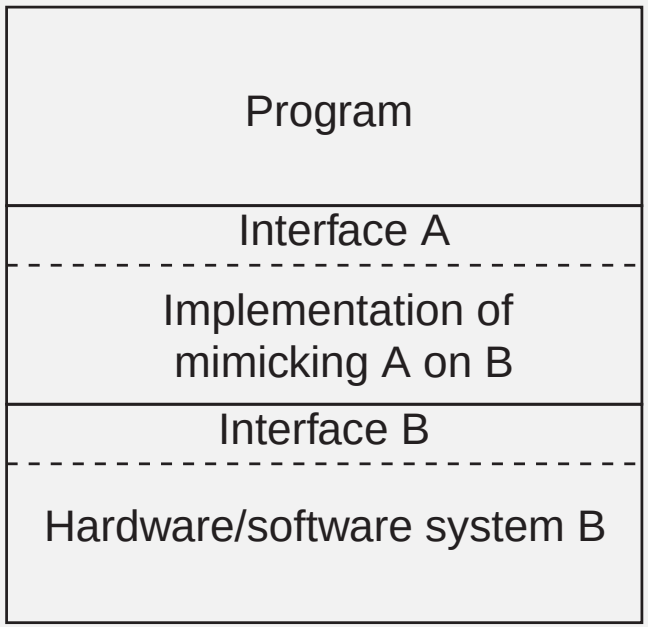
## Fontossága

A virtualizáció szerepe egyre nő több okból.

- A hardver gyorsabban fejlődik a szoftvernél
- Növeli a kód hordozhatóságát és költöztethetőségét
- A hibás vagy megtámadott rendszereket könnyű így elkülöníteni




(a)

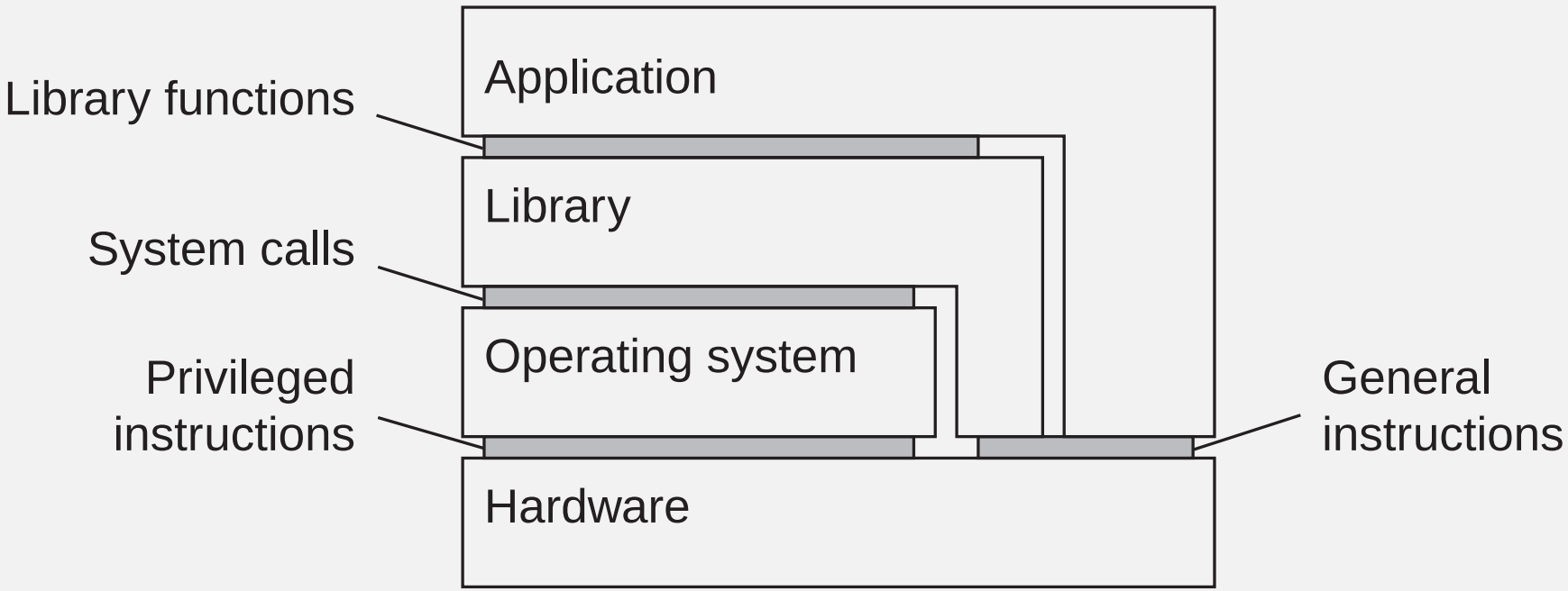



(b)

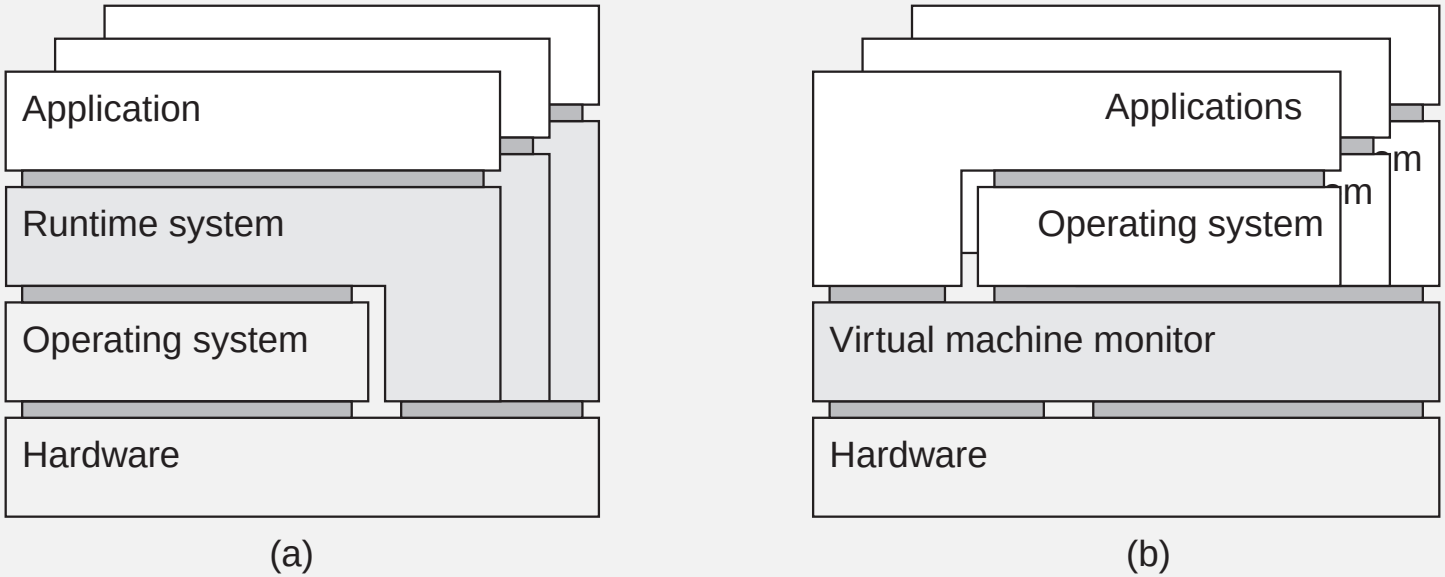
# A virtuális gépek szerkezete

## Virtualizálható komponensek

A rendszereknek sokfajta olyan rétege van, amely mentén virtualizálni lehet a komponenseket. Mindig eldöntendő, milyen interfészeket kell szolgáltatnia a virtuális gépnek (és milyeneket vehet igénybe).



# Process VM, VM monitor




- Process VM: A virtuális gép (virtual machine, VM) közönséges programként fut, bájtkódot (előfordított programkódot) hajt végre. Pl. JVM, CLR, de vannak speciális célúak is, pl. ScummVM.
- VM Monitor (VMM), hypervisor: Hardver teljeskörű virtualizációja, bármilyen program és operációs rendszer futtatására képes. Pl. VMware, VirtualBox.

# VM monitorok működése

## VM monitorok működése

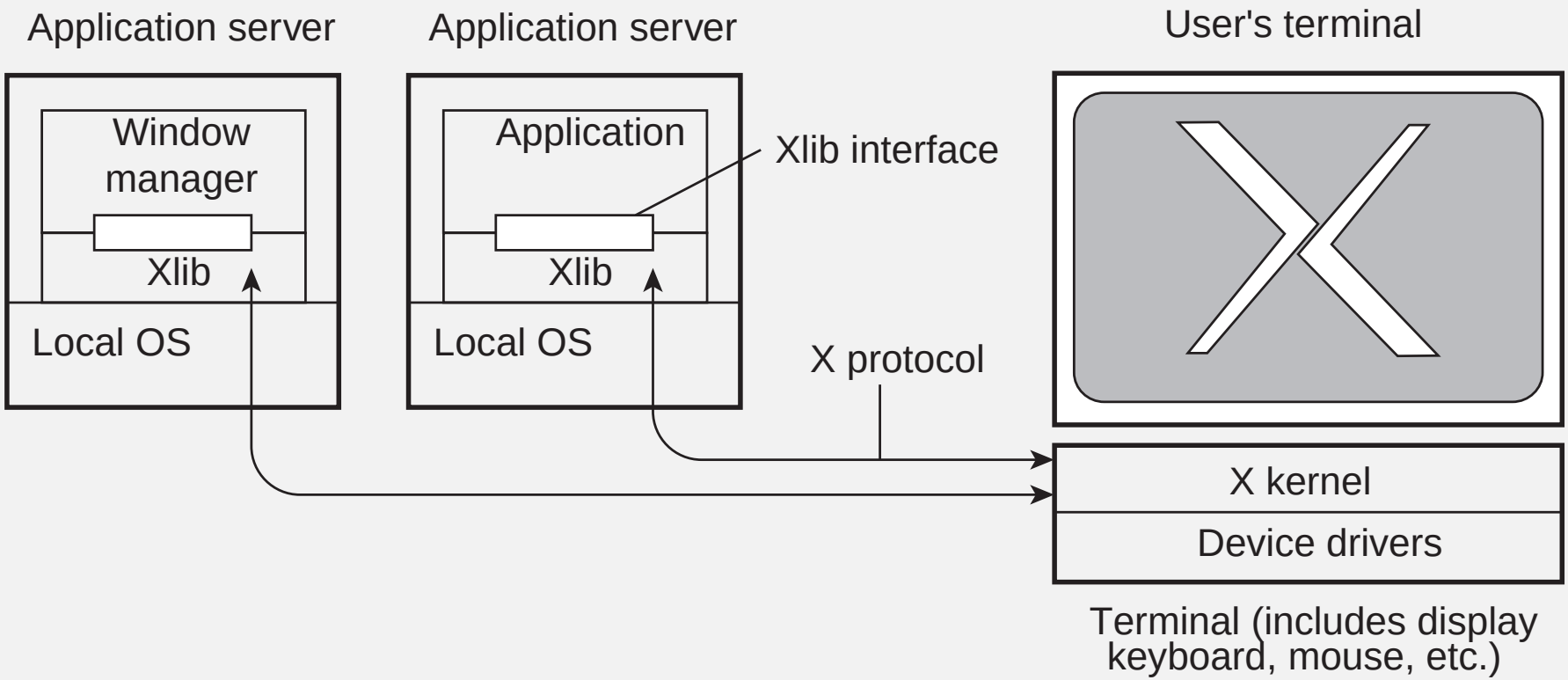
Sok esetben a VMM egy operációs rendszeren belül fut.

- A VMM a futtatott gépi kódú utasításokat átalakítja a gazdagép utasításaivá, és azokat hajtja végre.
- A rendszerhívásokat és egyéb privilegizált utasításokat, amelyek végrehajtásához az operációs rendszer közreműködésére lenne szükség, megkülönböztetett módon kezeli.

# Kliens: felhasználói felület

## Essence

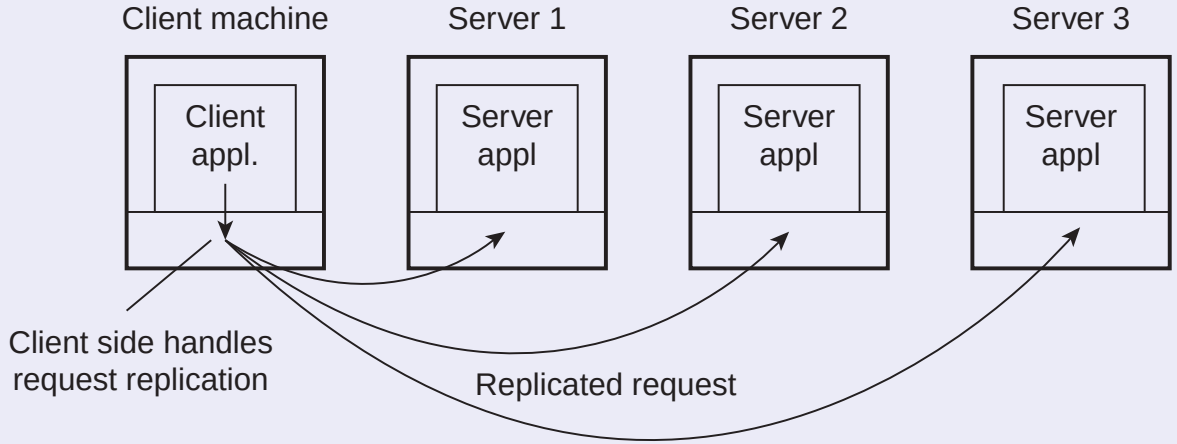
A kliensoldali szoftver egyik legfontosabb feladata a (grafikus) felhasználói interfész biztosítása.



# Kliens: átlátszóság

## A kliensekkel kapcsolatos főbb átlátszóságok

- hozzáférési: az RPC kliensoldali csonkja
- elhelyezési/áthelyezési: a kliensoldali szoftver tartja számon, hol helyezkedik el az erőforrás
- többszörözési: a klienscsonk kezeli a többszörözött hívásokat




- meghibásodási: sokszor csak a klienshez helyezhető – csak ott jelezhető a kommunikációs probléma

# Szerver: általános szerkezet

## Általános modell

szerver: Olyan folyamat, amely egy (vagy akár több) porton várja a kliensek kéréseit. Egy adott porton (ami egy 0 és 65535 közötti szám) a szerver egyfajta szolgáltatást nyújt.
A 0-1023 portok közismert szolgáltatásokat nyújtanak, ezeket Unix alapú rendszereken csak rendszergazdai jogosultsággal lehet foglalni.

| | | |
|---|---|---|
| ftp-data | 20 | File Transfer [adatátvitel] |
| ftp | 21 | File Transfer [vezérlés] |
| ssh | 22 | Secure Shell |
| telnet | 23 | Telnet |
| smtp | 25 | Simple Mail Transfer |
| login | 49 | Login Host Protocol |

# Szerver: általános szerkezet

## Szerverfajták

szuperszerver : Olyan szerver, amelyik több porton figyeli a bejövő kapcsolatokat, és amikor új kérés érkezik, új folyamatot/szálat indít annak kezelésére. Pl. Unix rendszerekben: *inetd*.

iteratív↔konkurens szerver : Az iteratív szerverek egyszerre csak egy kapcsolatot tudnak kezelni, a konkurensek párhuzamosan többet is.

# Szerver: sávon kívüli kommunikáció

## Sürgős üzenetek küldése

Meg lehet-e szakítani egy szerver működését kiszolgálás közben?

## Külön port

A szerver két portot használ, az egyik a sürgős üzeneteknek van fenntartva:

- Ezt külön szál/folyamat kezeli
- Amikor fontos üzenet érkezik, a normál üzenet fogadása szünetel
- A szálnak/folyamatnak nagyobb prioritást kell kapnia, ehhez az oprendszer támogatása szükséges

## Sávon kívüli kommunikáció

Sávon kívüli kommunikáció használata, ha rendelkezésre áll:

- Pl. a TCP protokoll az eredeti kérés kapcsolatán keresztül képes sürgős üzenetek továbbítására
- Szignálok formájában kapható el a szerveren belül

# Szerver: állapot

## Állapot nélküli szerver

Nem tart fenn információkat a kliensről a kapcsolat bontása után.

- Nem tartja számon, melyik kliens milyen fájlból kért adatokat
- Nem ígéri meg, hogy frissen tartja a kliens gyorsítótárát
- Nem tartja számon a bejelentkezett klienseket: nincsen munkamenet (session)

## Következmények

- A kliensek és a szerverek teljesen függetlenek egymástól
- Kevésbé valószínű, hogy inkonzisztencia lép fel azért, mert valamelyik oldal összeomlik
- A hatékonyság rovására mehet, hogy a szerver nem tud semmit a kliensről, pl. nem tudja előre betölteni azokat az adatokat, amelyekre a kliensnek szüksége lehet

# Szerver: állapot

## Állapotteljes szerverek

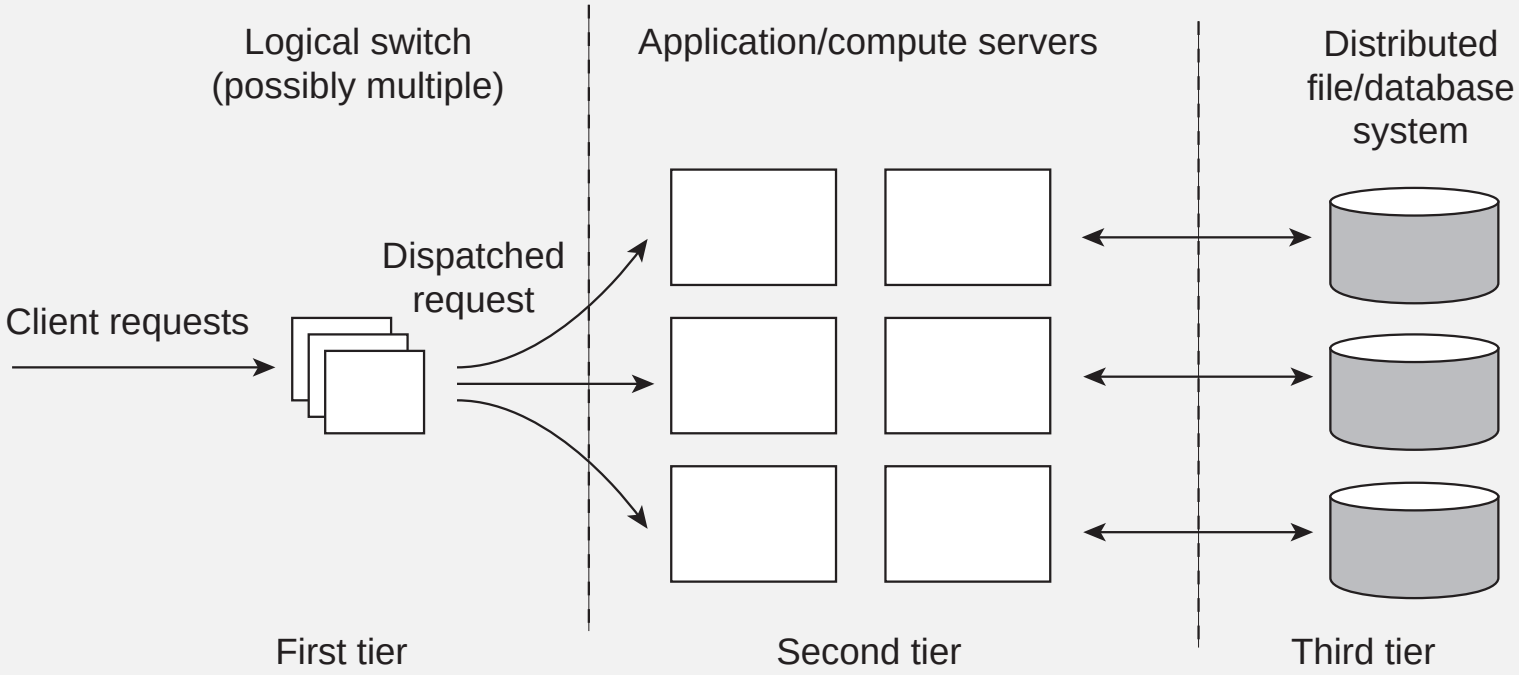
Állapotot tart számon a kliensekről:

- Megjegyzi, melyik fájlokat használta a kliens, és ezeket előre meg tudja nyitni legközelebb
- Megjegyzi, milyen adatokat töltött le a kliens, és frissítéseket küldhet neki

## Előnyök és hátrányok

Az állapotteljes szerverek nagyon hatékonyak tudnak lenni, ha a kliensek lokálisan tárolhatnak adatokat.
Az állapotteljes rendszereket megfelelően megbízhatóvá is lehet tenni a hatékonyság jelentős rontása nélkül.

# Szerver: háromrétegű clusterek




## A diszpécserréteg

Az első réteg feladata nagyon fontos: a beérkező kéréseket hatékonyan kell a megfelelő szerverhez továbbítani.

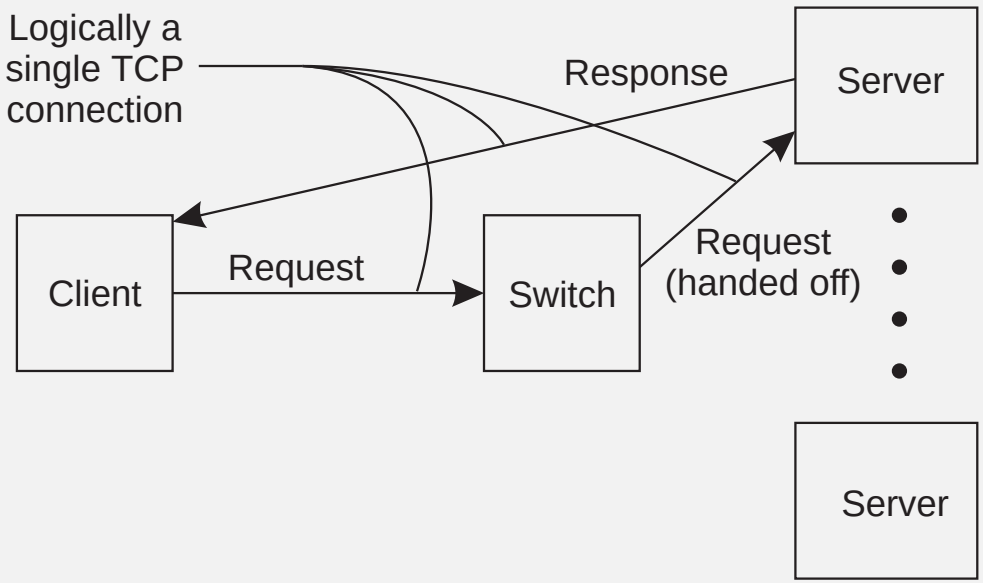
# A kérések kezelése

## Szűk keresztmetszet

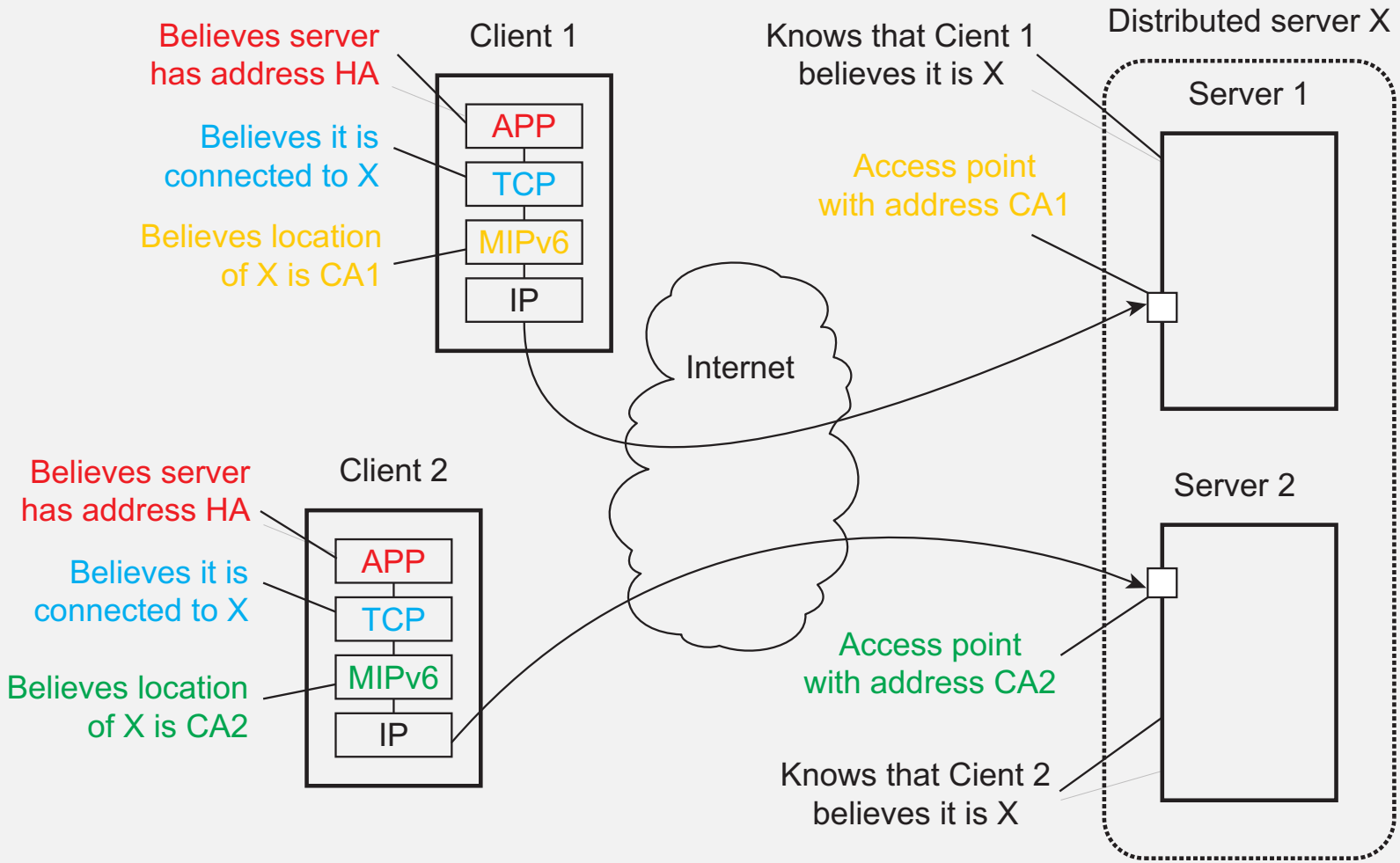
Ha minden kapcsolatot végig az első réteg kezel, könnyen szűk keresztmetszetté válhat.

## Egy lehetséges megoldás

A terhelés csökkenthető, ha a kapcsolatot átadjuk (TCP handoff).



# Elosztott rendszerek: mobil IPv6



# Elosztott rendszerek: mobil IPv6

## Essence

A mobil IPv6-ot támogató kliensek az elosztott szolgáltatás bármelyik peer-jéhez kapcsolódhatnak.

- A *C* kliens kapcsolódik a szerver otthonának (home address, *HA*) IPv6 címéhez
- A *HA* címen a szerver hazai ügynöke (home agent) fogadja a kérést, és a megfelelő felügyeleti címre (care-of address, *CA*) továbbítja
- Ezután *C* és *CA* már közvetlenül tudnak kommunikálni (*HA* érintése nélkül)

## Példa: kollaboratív CDN-ek

Az origin server tölti be *HA* szerepét, és átadja a beérkező kapcsolatot a megfelelő peer szervernek. A kliensek számára az origin és a peer egy szervernek látszik.

# Kódmigráció: jellemző feladatok

## Kódmigráció

kódmigráció: olyan kommunikáció, amely során nem csak adatokat küldünk át

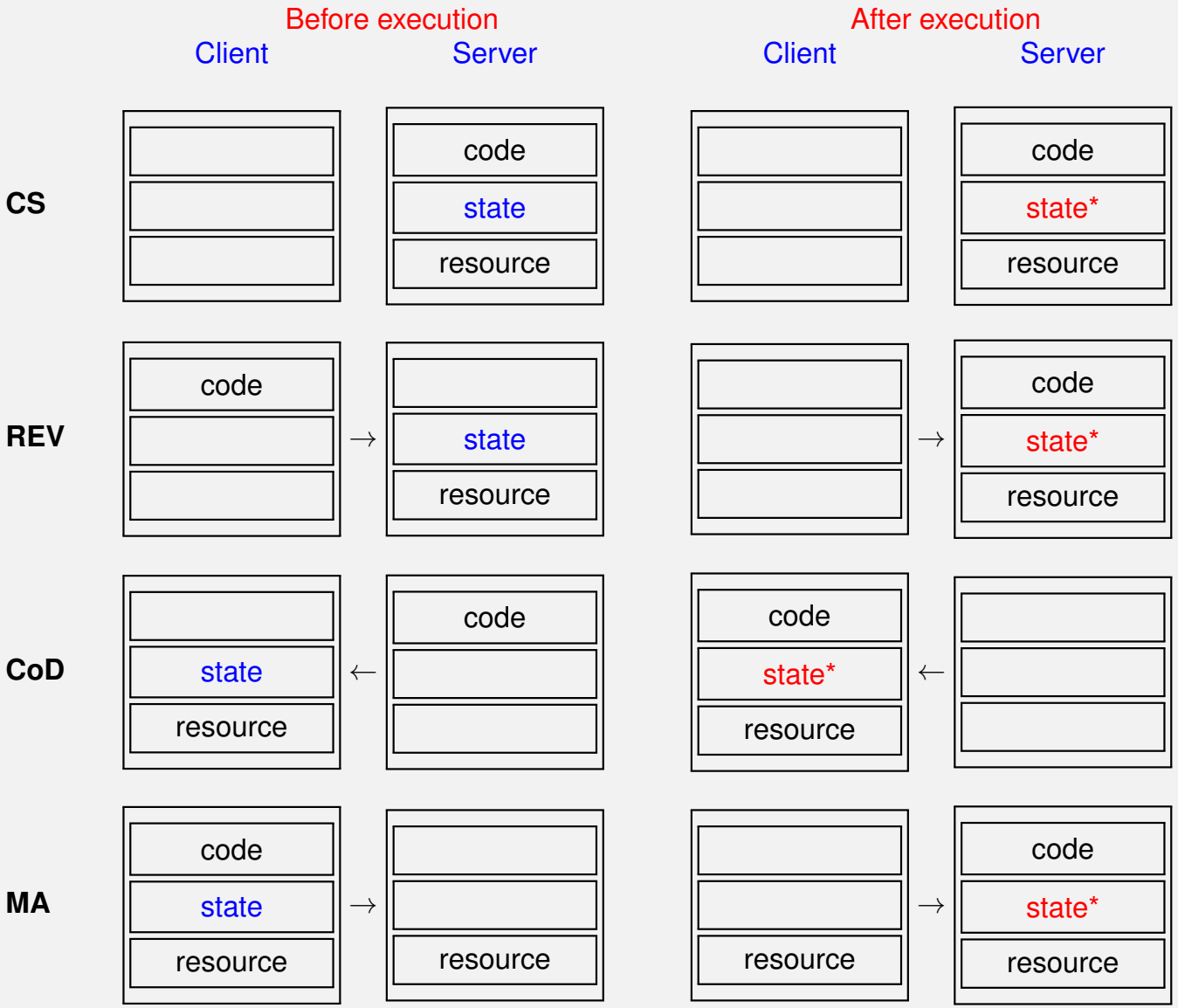
## Jellemző feladatok

Néhány jellemző feladat, amelyhez kódmigrációra van szükség.[a]

- Client-Server: a szokásos kliens-szerver kommunikáció, nincsen kódmigráció
- Remote Evaluation: a kliens feltölti a kódot, és a szerveren futtatja
- Code on Demand: a kliens letölti a kódot a szerverről, és helyben futtatja
- Mobile Agent: a mobil ágens feltölti a kódját és az állapotát, és a szerveren folytatja a futását

[a] A következő fólia ezeket a rövidítéseket tartalmazza.

# Kódmigráció: jellemző feladatok



# Kódmigráció: gyenge és erős mobilitás

## Objektumkomponensek

- Kódszegmens: a programkódot tartalmazza
- Adatszegmens: a futó program állapotát tartalmazza
- Végrehajtási szegmens: a futtató szál környezetét tartalmazza

## Gyenge mobilitás

A kód- és adatszegmens mozgatása (a kód újraindul):

- Viszonylag egyszerű megtenni, különösen, ha a kód hordozható
- Irány szerint: feltöltés (push, ship), letöltés (pull, fetch)

## Erős mobilitás

A komponens a végrehajtási szegmenssel együtt költözik

- Migráció: az objektum átköltözik az egyik gépről a másikra
- Klónozás: a kód másolata kerül a másik gépre, és ugyanabból az állapotból indul el, mint az eredeti; az eredeti is fut tovább

# Kódmigráció: az erőforrások elérése

## Erőforrások elérése

Az eredeti gépen található erőforrások költözés után a kód számára távoliakká válnak.

## Erőforrás–gép kötés erőssége

- Mozdíthatatlan: nem költöztethető (pl. fizikai hardver)
- Rögzített: költöztethető, de csak drágán (pl. nagy adatbázis)
- Csatolatlan: egyszerűen költöztethető (pl. gyorsítótár)

## Komponens–erőforrás kötés jellege

Milyen jellegű erőforrásra van szüksége a komponensnek?

- Azonosítókapcsolt: egy konkrét (pl. a cég adatbázisa)
- Tartalomkapcsolt: adott tartalmú (pl. bizonyos elemeket tartalmazó cache)
- Típuskapcsolt: adott jellegű (pl. színes nyomtató)

# Kódmigráció: az erőforrások elérése

## Kapcsolat az erőforrással

Hogyan tud a komponens kapcsolatban maradni az erőforrással?

- Típuskapcsolt erőforrás esetén a legkönnyebb újrakapcsolódni egy lokális, megfelelő típusú erőforráshoz
- Azonosítókapcsolt vagy tartalomkapcsolt esetben:
  - rendszerszintű hivatkozást létesíthetünk az eredeti erőforrásra,
    - mozdíthatatlan erőforrások esetén ez az egyetlen lehetőség
    - minden más esetben is szóba jöhet, de általában van jobb megoldás
  - azonosítókapcsolt erőforrást érdemes áthelyezni[a]
  - tartalomkapcsolt erőforrást érdemes lemásolni[a]

[a] ha nem túl költséges

# Kódmigráció: heterogén rendszerben

## Nehézségek

- A célgép nem biztos, hogy képes futtatni a migrált kódot
- A processzor-, szál- és/vagy folyamatkörnyezet nagyban függ a lokális hardvertől, oprendszertől és futtatókörnyezettől

## Megoldás problémás esetekben

Virtuális gép használata: akár process VM, akár hypervisor. Természetesen a virtuális gépnek elérhetőnek kell lennie mindkét környezetben.