PÁZMÁNY
Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Számítógépes hálózatok
#06 – Transport layer protocols 1

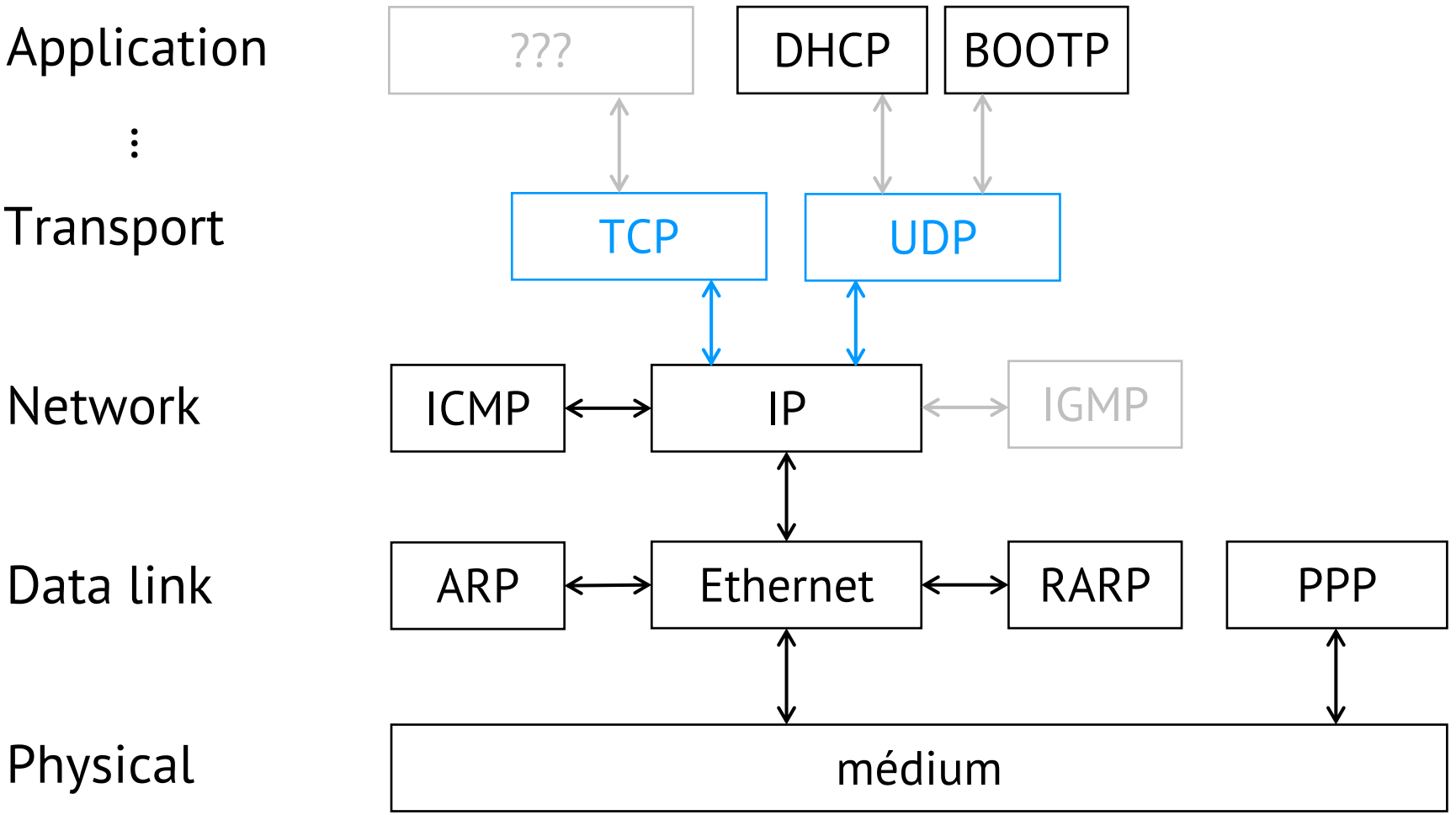
2024. október 18.

Naszlady Márton Bese

naszlady@itk.ppke.hu

#06/1 – A transport layer

Itt tartunk most



hálózati réteg beállítása, bootolás

adatszállítás, alkalmazások, portok, handshake

router, útvonalak, irányítás, logikai cím (IP cím)

bridge, switch, fizikai cím (MAC) hibakezelés, ütközés, VLAN

vezeték, repeater, hub, elektromos jelek, zaj

A transport layer szerepe

Feladata az adat átvitele két alkalmazás (program) között.

Tipikusan ennek a rétegnek a feladata:

- megoldani, hogy az adat egyetlen darabkája se vesszen el,
- megoldani, hogy az adat jó ütemben, megbízhatóan kerüljön átvitelre,
- megoldani, hogy az üzenetet a címzett hálózati eszközön belül a kívánt alkalmazás kapja meg.

A transport layer szerepe

Feladata az adat átvitele két alkalmazás (program) között.

Legfontosabb jellemzői:

- elrejti az alsóbb rétegek jellemzőit,
- a fizikai hálózattól függetlenül üzemel,
- csak a végponti berendezésekben van jelen,
- az alkalmazások fejlesztői számára szabványos eljárásokat definiál.

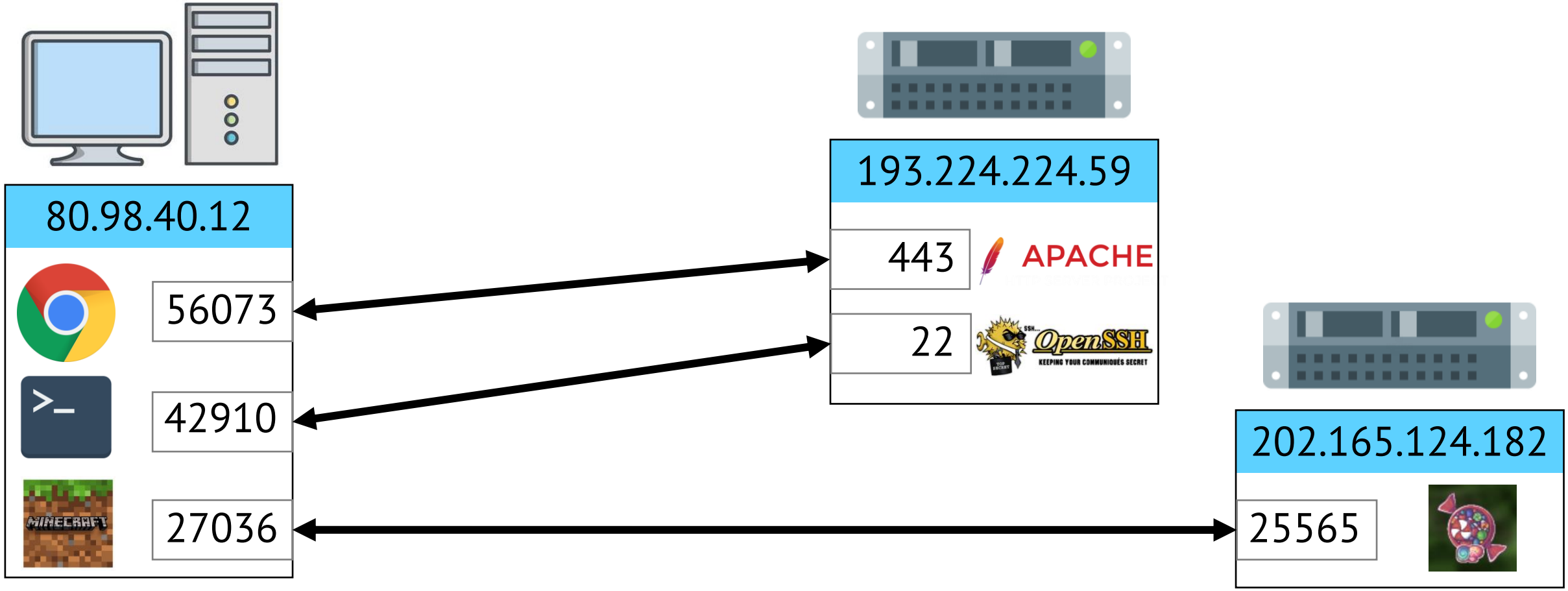
Multiplexálás és demultiplexálás

A hálózati eszközön egyidőben több folyamat (process) is futhat, melyek mindegyike szeretne valamilyen hálózati szolgáltatást használni.

Meg kell különböztetni, hogy az adott üzenet feladója és címzettje melyik hálózati eszközön belül melyik folyamat (process) volt!

Multiplexálás és demultiplexálás

A multiplexálás/demultiplexálás segítése érdekében az operációs rendszerben implementált transport layer címkézi, folyamathoz rendeli az üzeneteket.



Port

Olyan azonosító, ami egyértelműen meghatároz egy adott alkalmazáshoz vagy folyamathoz tartozó kommunikációs végpontot.

Számokkal jelöljük, 16 biten ábrázoljuk (0 – 65535)

0–1024: *well-known* portok, az alkalmazáshoz rendelésük világállandó

1024-től szabadon használható, csak a szokásjog érvényes

Fontos példák:

22 Secure Shell Protocol (SSH)

25 Simple Mail Transfer Protocol (SMTP)

53 Domain Name System (DNS)

80 Hypertext Transfer Protocol (HTTP)

Portok használata a transport layer szintjén

Transport layer szintű kommunikáció során a feladó és a címzett is **IP címmel és portszámmal** azonosítja magát.

Jelölni az IPv4 cím után írt kettősponttal szokás:

küldő		címzett
80.98.40.12:56073	→	193.224.224.59:443

Socket

A transzport réteg által megvalósított kapcsolat az operációs rendszerekben futó programok számára ún. socketek formájában láthatók.

Socket: egy IP cím, a port száma és a használt protokoll határozza meg.

A socket címe az IP címből és a port számából áll.

Helyi socketcím: a helyi IP cím és a helyi portszám
Távoli socketcím: a távoli IP cím és a távoli portszám

Az egymáshoz rendelt helyi és távoli socketek együtt egy socketpárt alkotnak.

Egy socketet egy alkalmazás nyit és zár, és az operációs rendszer nyilvántartja, hogy melyik socket melyik folyamathoz tartozik.

Socket állapotok

A socketek többféle állapotban lehetnek:

Listen: A szolgáltatásokat nyújtó alkalmazások (szerver-folyamatok) socketeket nyitnak, amiken figyelik, hogy érkezik-e bármilyen kérés feléjük

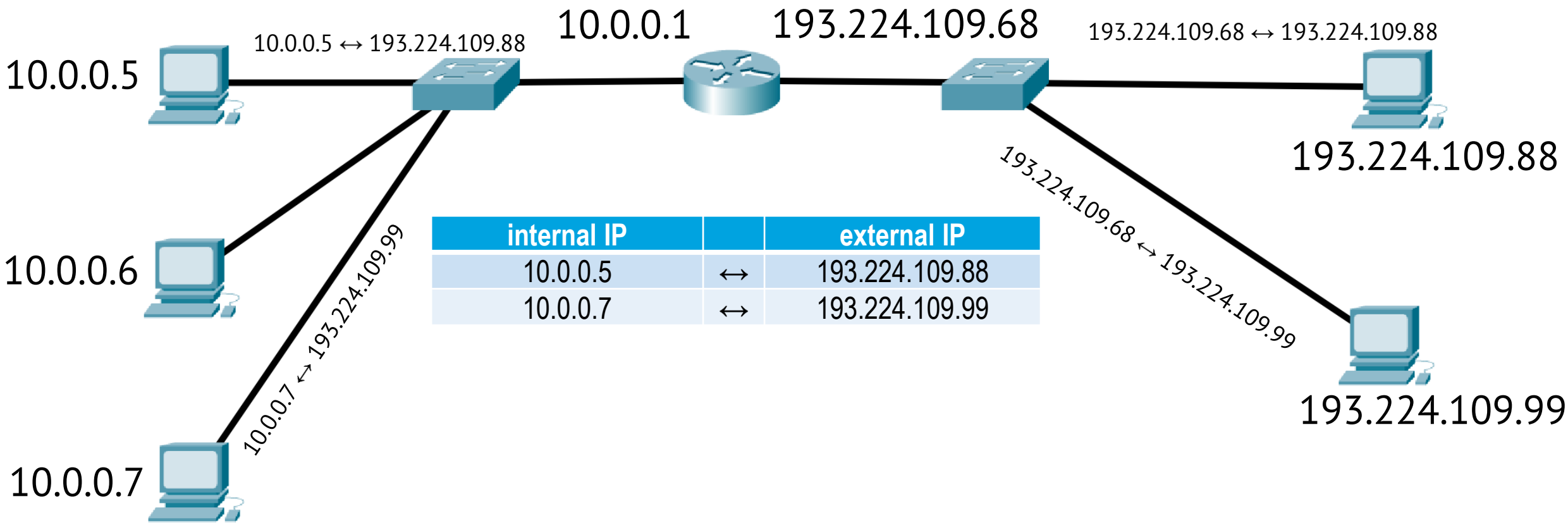
Established: Ha kérés érkezik, és kiépül a kapcsolat, akkor a socket established állapotba kerül. Létezhet egyazon időben több socket is ugyanazzal a címmel, amik más-más (al)folyamatokhoz vannak rendelve.

Closed: A kommunikáció végeztével/lezárásával a socket closed állapotba léphet.

Network Address Translation (ismétlés)

NAT-olás esetén a routernek emlékezni kell, hogy mit mire cserélt ki.

Táblázatot kell vezetni az élő kapcsolatokról.

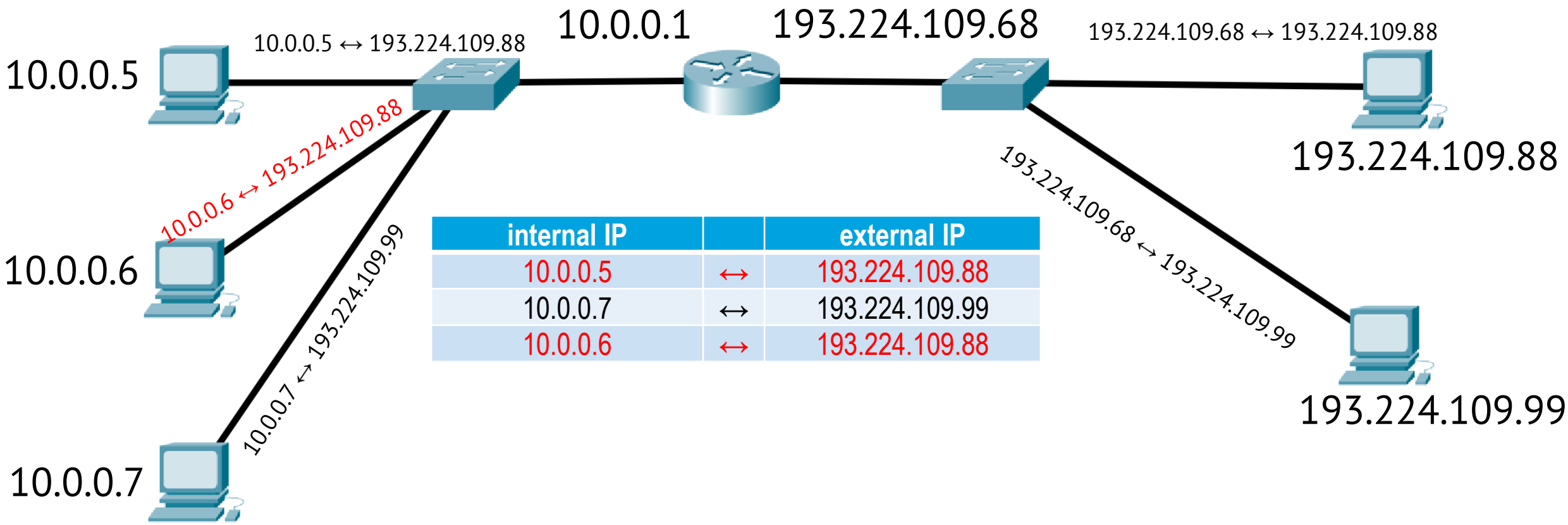


internal IP		external IP
10.0.0.5	↔	193.224.109.88
10.0.0.7	↔	193.224.109.99

NAT probléma (ismétlés)

Mi van akkor, ha több belső cím is ugyanazt a külső címet akarja elérni?

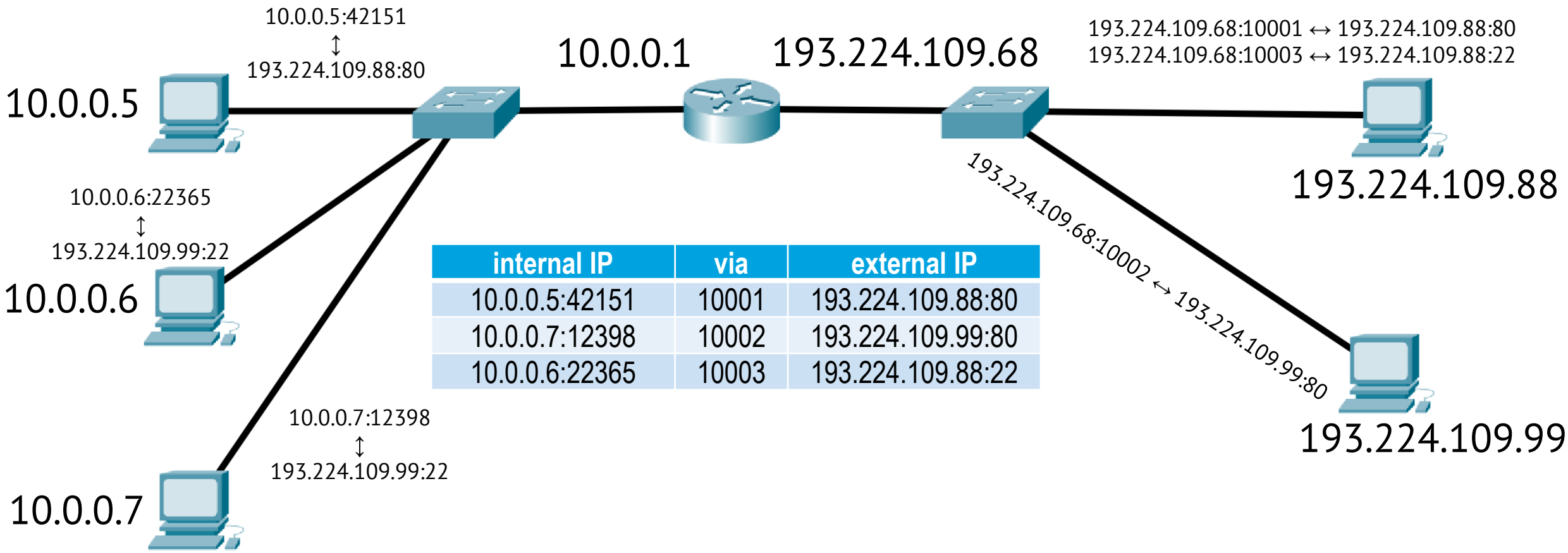
Honnan tudja a NAT-ot végző eszköz, hogy merre továbbítsa a csomagot?



internal IP		external IP
10.0.0.5	↔	193.224.109.88
10.0.0.7	↔	193.224.109.99
10.0.0.6	↔	193.224.109.88

Network Address and Port Transaltion (NAPT)

Megoldás: Az L4 szinten létrejövő socketek mintájára nem csak IP-IP párokat képez, hanem IP:port-IP:port párokat, és szükség esetén módosítja a külső portszámot is.

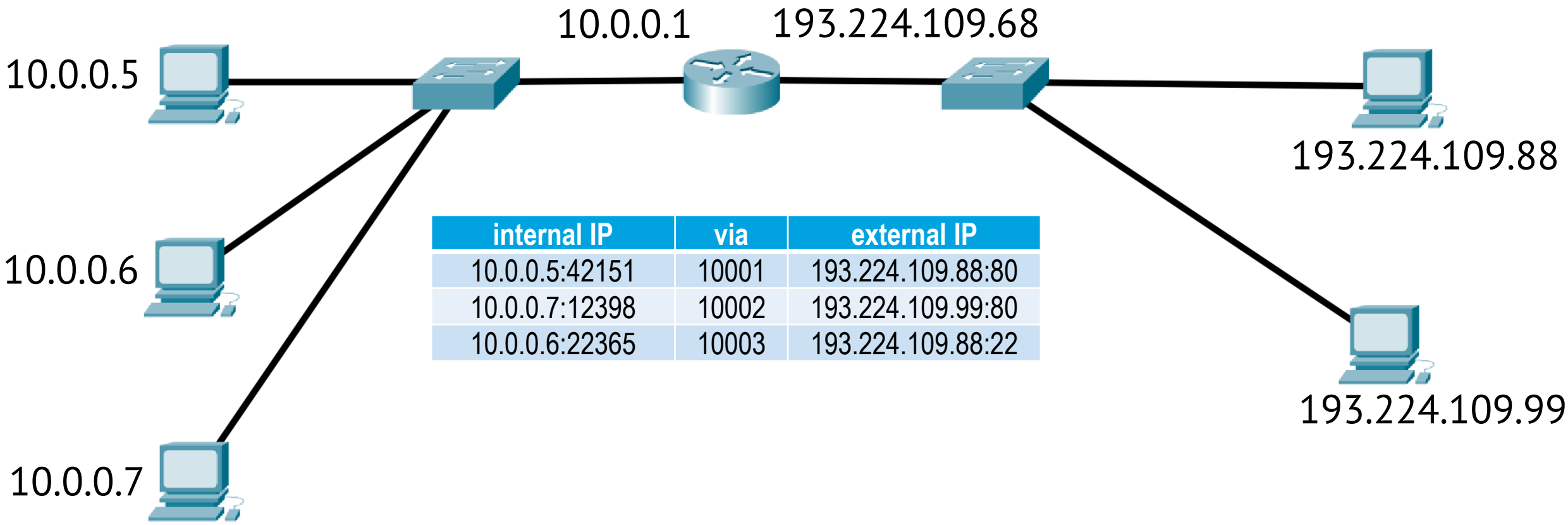


internal IP	via	external IP
10.0.0.5:42151	10001	193.224.109.88:80
10.0.0.7:12398	10002	193.224.109.99:80
10.0.0.6:22365	10003	193.224.109.88:22

Port forwarding

Ha belülről kifelé kezdeményezek, akkor egyszerű a táblázat kitöltése (tanult bejegyzések).

Hogyan kezelem a kívülről befelé irányuló kéréseket?

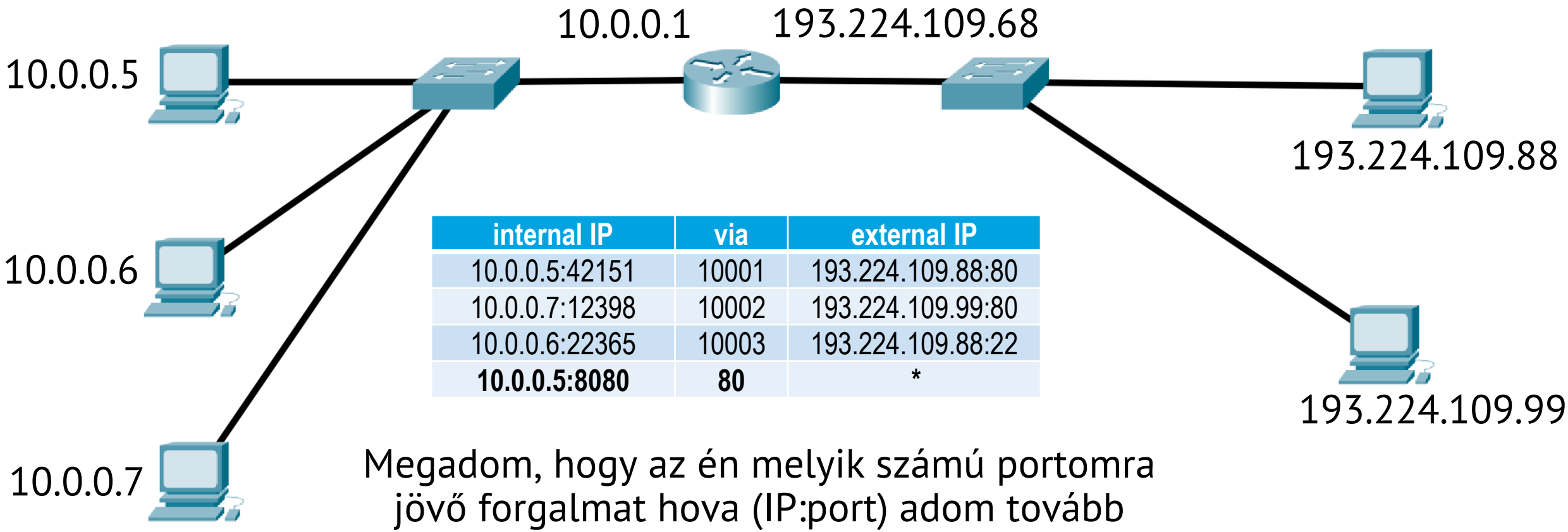


internal IP	via	external IP
10.0.0.5:42151	10001	193.224.109.88:80
10.0.0.7:12398	10002	193.224.109.99:80
10.0.0.6:22365	10003	193.224.109.88:22

Port forwarding

Ha belülről kifelé kezdeményezek, akkor egyszerű a táblázat kitöltése (tanult bejegyzések).

Hogyan kezelem a kívülről befelé irányuló kéréseket? -> **statikus bejegyzésekkel**



internal IP	via	external IP
10.0.0.5:42151	10001	193.224.109.88:80
10.0.0.7:12398	10002	193.224.109.99:80
10.0.0.6:22365	10003	193.224.109.88:22
10.0.0.5:8080	**80**	*

Megadom, hogy az én melyik számú portomra jövő forgalmat hova (IP:port) adom tovább

A transport layer implementációi

User Datagram Protocol (UDP)

- Összeköttetés nélküli protokoll
- Nem megbízható, mert nincs benne visszajelzés a kézbesítésről.
- Nincs sorrendtartás.
- Rendkívül gyors és hatékony.

„valakire rákiabálás”

Transmission Control Protocol (TCP)

- Összeköttetés alapú protokoll
- Megbízható, mivel visszajelzést ad a szegmensek megérkezéséről.
- Sorrendtartó átvitel.
- Lassúbb az összeköttetés felépítése, de utána gyors az adatátvitel.

„személyes beszélgetés”

#06/1 – Összefoglalás

Fogalom	Transport layer szerepe
	Port, well-known port példák
	Socket, socket állapotok
Protokoll	UDP és TCP általános jellemzői

#06/2 – Transmission Control Protocol

TCP

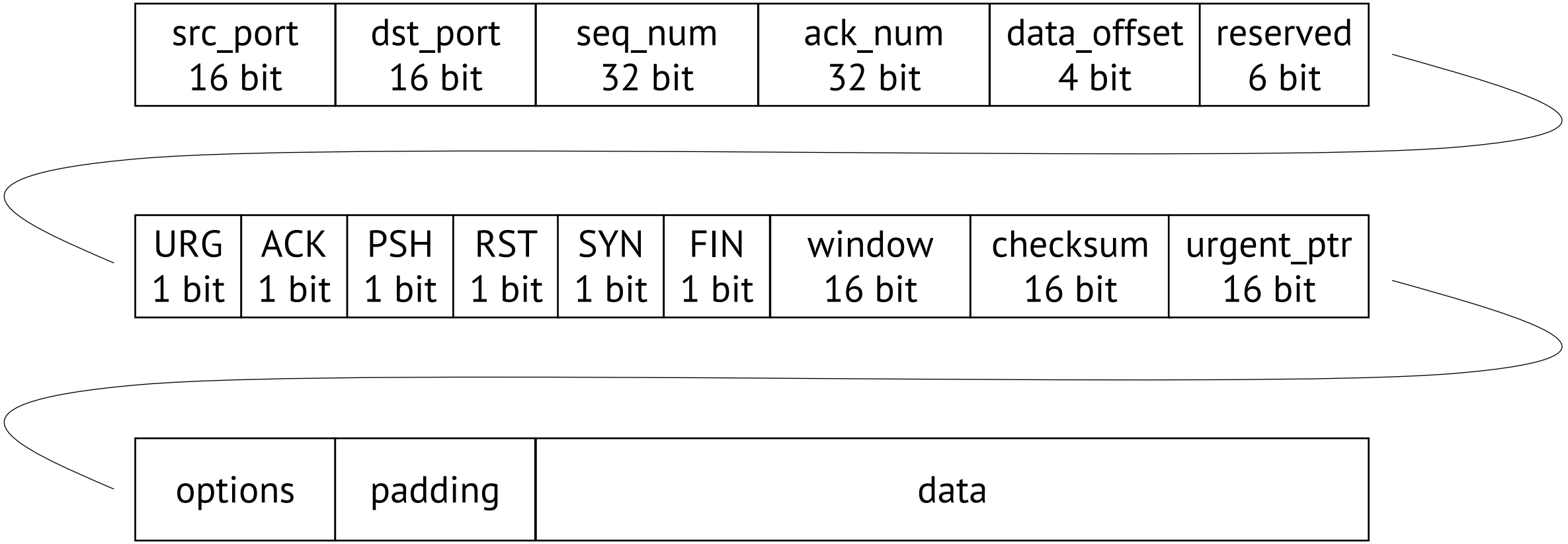
A leggyakoribb L4 szintű protokoll

- Kapcsolatorientált, azaz mindig pontosan két partner közt valósul meg (csak unicast).
- A megbízhatatlan IP szolgáltatás fölött egy megbízható szolgáltatást hoz létre.
- Alkalmazás-alkalmazás kommunikációt valósít meg (vö. IP host-host között kapcsol).
- A TCP darabokra bontja az információt. Az IP rétegnek átadott darab neve ***segment***.
- A TCP egy bitfolyamot visz át.
  „Amit az egyik oldalon beleöntünk, pontosan az folyik ki a másik oldalon.”

TCP

- A TCP minden segmentre nyugtát vár. Ha nem érkezik nyugta, akkor újra elküldi a segmentet. (Nem feltétlenül minden segmentet külön-külön, és nem feltétlenül mindet azonnal.)
- Minden segmentet ellenörző összeg véd. Ha ez hibás, akkor eldobja azt. Ha eldobta, akkor nem küld nyugtát, tehát újra meg fogja kapni.
- Az IP csomagok nem feltétlenül sorrendben érkeznek meg. A TCP helyreállítja a sorrendet.
- Az IP csomagok duplázódhatnak is. A TCP kiszűri a duplikátumokat is.
- A TCP folyamatvezérlést (flow control) alkalmaz: az adatáramlás sebességét fékezni és gyorsítani is tudja a rendelkezésre álló erőforrások függvényében.

TCP üzenetformátum



TCP üzenetformátum

- src_port — a forrás oldali port száma
- dst_port — a céloldali port száma
- seq_num — sequence number; a byte-folyamban az ebben a segmentben küldött első byte sorszáma, kezdetben sem nulla
- ack_num — acknowledgement number; a vett byte-folyamot eddig a sequ. numberig nyugtázom; a következő venni kívánt sorszámot tartalmazza
- data_offset — a fejrész hosszát adja meg 4 byte-os egységekben; a fejrész hossza az options rész változó hossza miatt változhat
- reserved — nem használható, fenntartott bitek

TCP üzenetformátum

Flagek:

- URG urgent; érvényes az urgent_ptr (sürgős adat mutató) rész
- ACK acknowledgement; érvényes az ack_num rész
- PSH push; azonnal add föl a felső rétegnek ezt a segmentet
- RST reset; durva bontás (lecsaptam a telefont)
- SYN syncronization; a kapcsolat indítása, szinkronizáljuk a seq_num értékeket
- FIN finish; befejezem az adatküldést (szépen leteszem a telefont)

TCP üzenetformátum

- window — az ack_num értéktől kezdve ennyi byte fogadására állok készen
- checksum — a teljes segmentre ÉS egy pszeudo-fejlécre számolt ellenőrző összeg. A pszeudo-fejléc is tartalmazza a forrás és a cél IP címét, ezzel a rossz helyre küldött TCP csomagok is kiszűrhetők.
- urgent_ptr — ha be van állítva az URG flag, akkor az ebben a mezőben lévő értékig az adatban sürgős adat van (pl. szakítsd meg a letöltést)
- options — különböző egyéb beállítások
pl. Maximum Segment Size (MSS), Window scale factor, ...
- padding — ha az options rész nem jön ki 4 byte-ra, akkor kiegészítjük
- data — nem kötelező, előfordul, hogy nincs adat. Pl. ha csak egy vett adatot nyugtázunk, akkor üres.

#06/2 – Összefoglalás

Protokoll	Transmission Control Protocol rendeltetése
Jellemzők	TCP feladatai
Képesség	TCP segment részeit azonosítani

#06/3 – Nyugtázási módszerek

Nyugtázási módszerek

Stop-and-Wait protocol

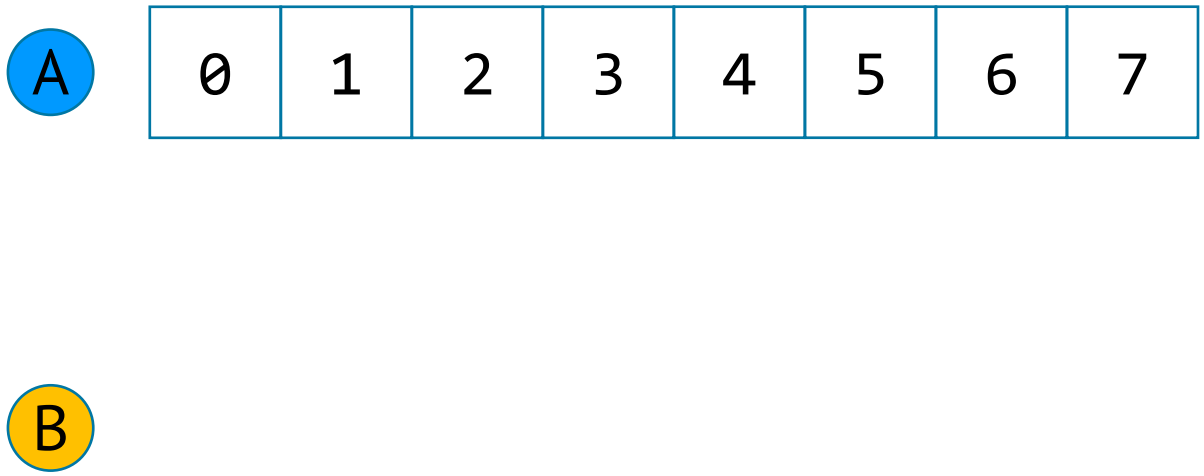
A segmenteket egyesével küldi el. Miután elküldött egy segmentet, egészen addig vár, amíg a segment fogadásáról nem érkezett visszajelzés. A következő segmentet csak akkor küldi, ha tudja, hogy az előző célba ért.

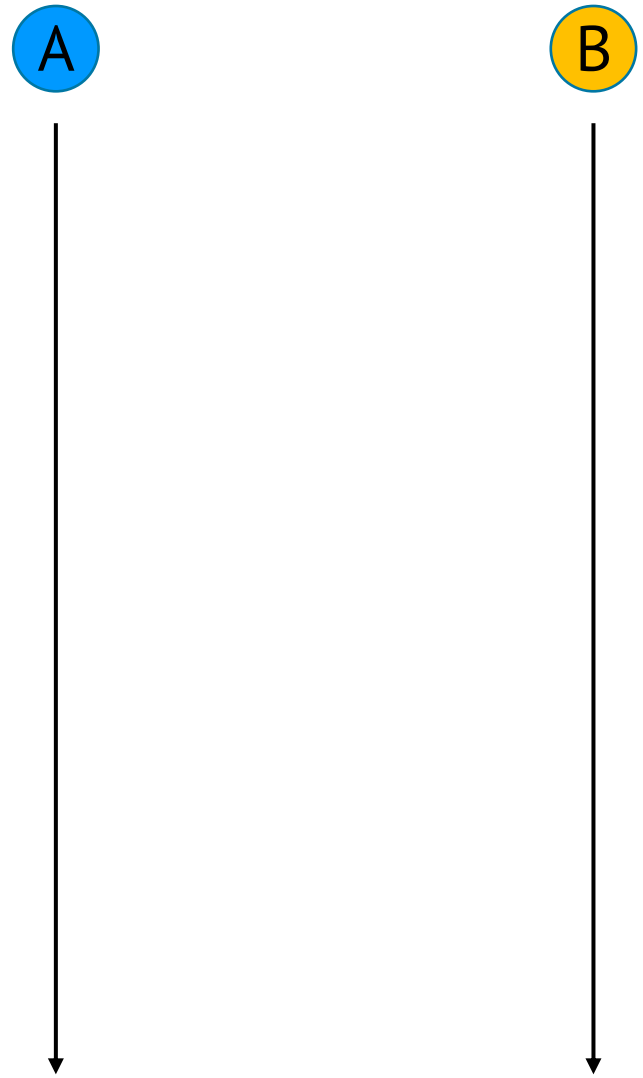
Sliding window protocol

Több segmentet küld egyszerre, egy bizonyos mennyiségig nem érdekli, hogy a fogadó oldal visszajelzett-e. Egy idő után (ablakméretnyi kiküldött segment után) azért ő is vár a nyugtázásra, és csak akkor folytatja a küldést, hamegkapja a sikerességről szóló visszajelzést.

Nyugtázási módszerek

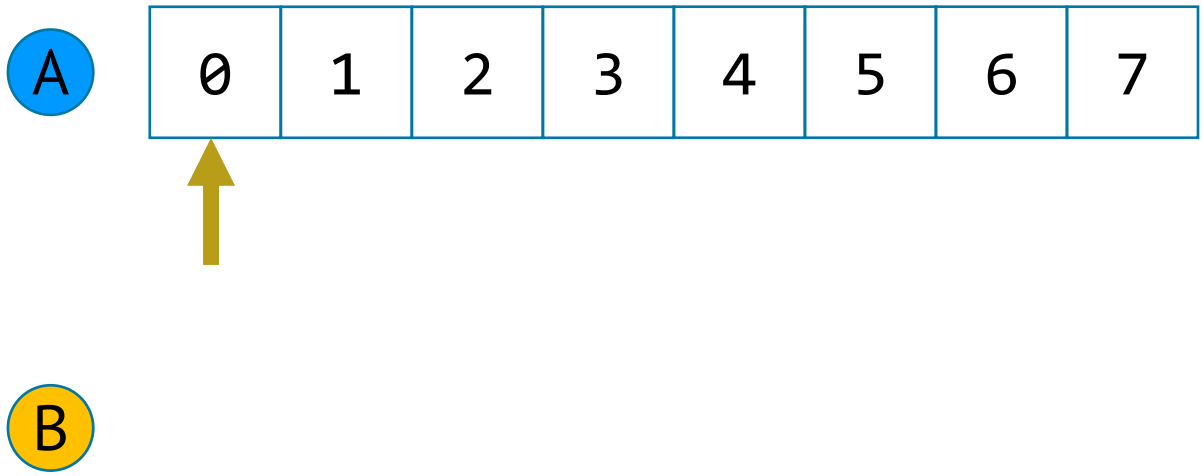
Stop-and-Wait protocol

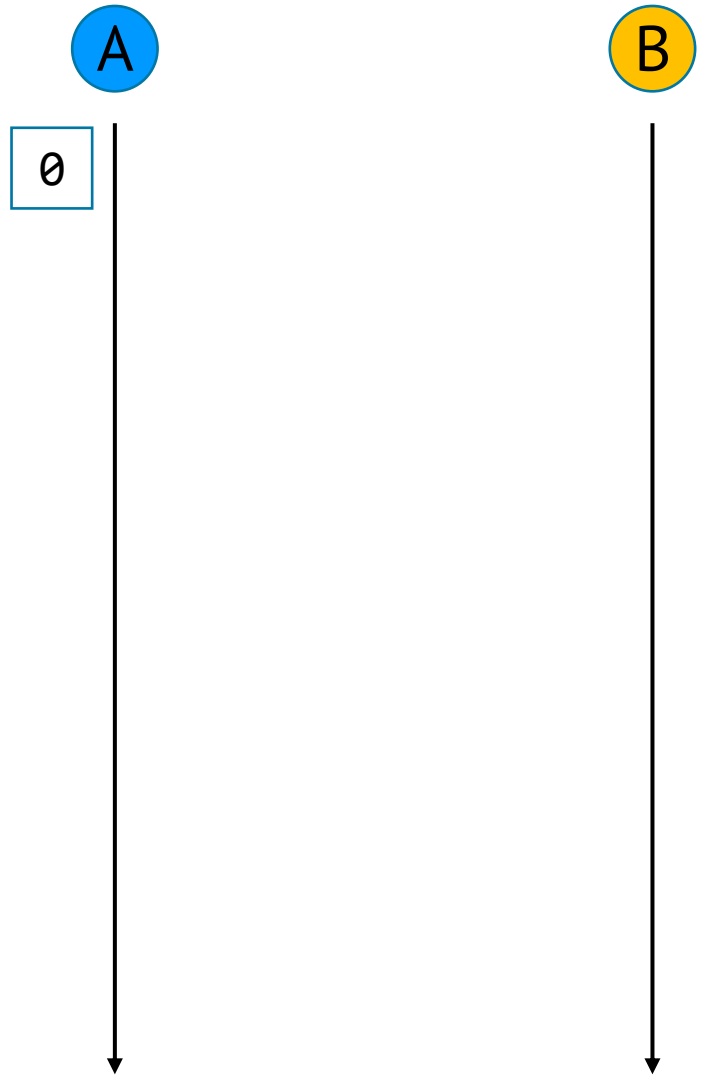




Nyugtázási módszerek

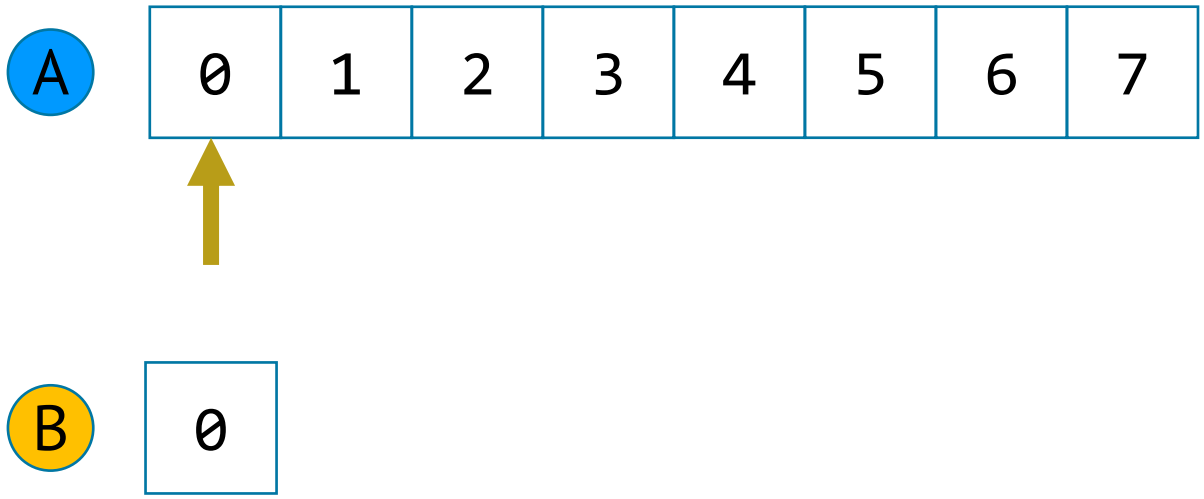
Stop-and-Wait protocol

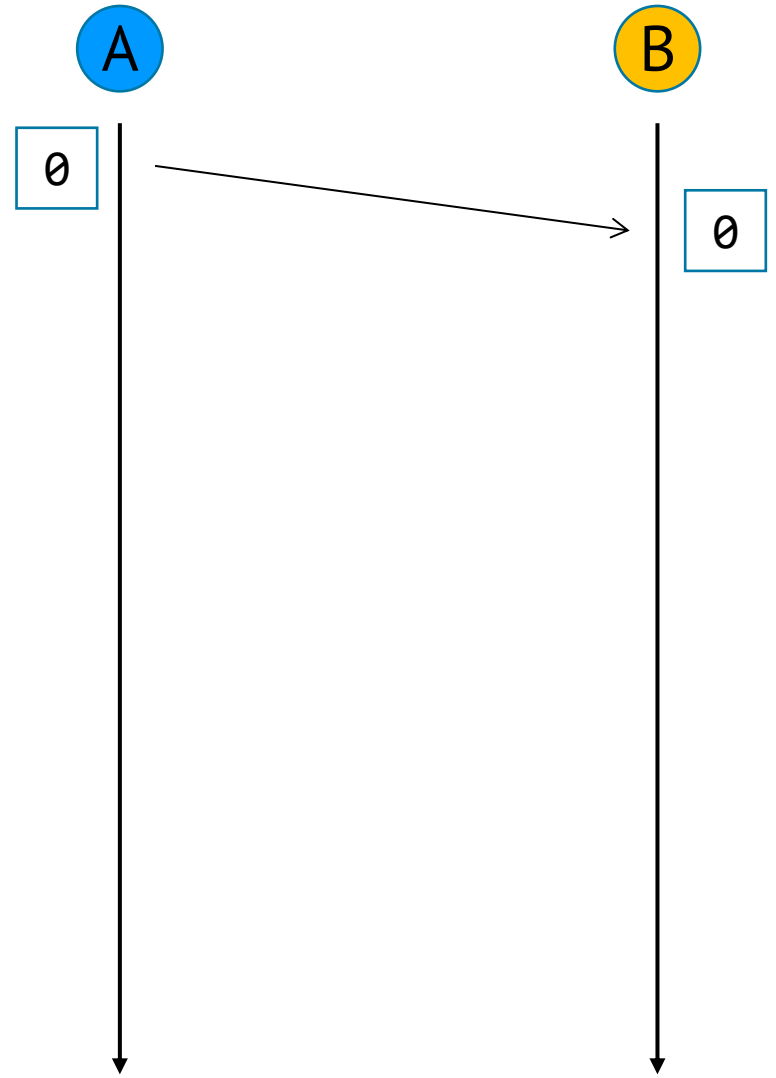




Nyugtázási módszerek

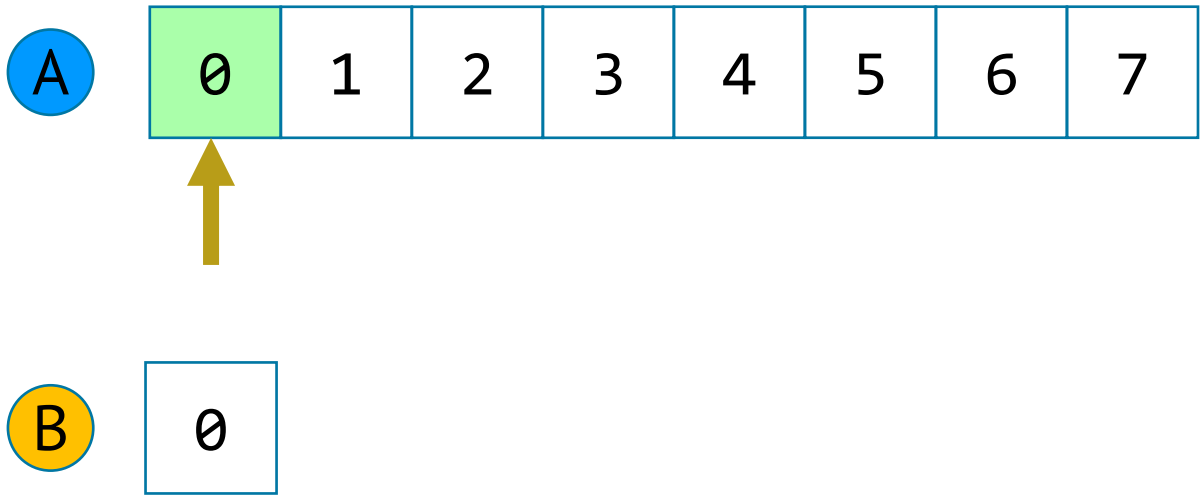
Stop-and-Wait protocol

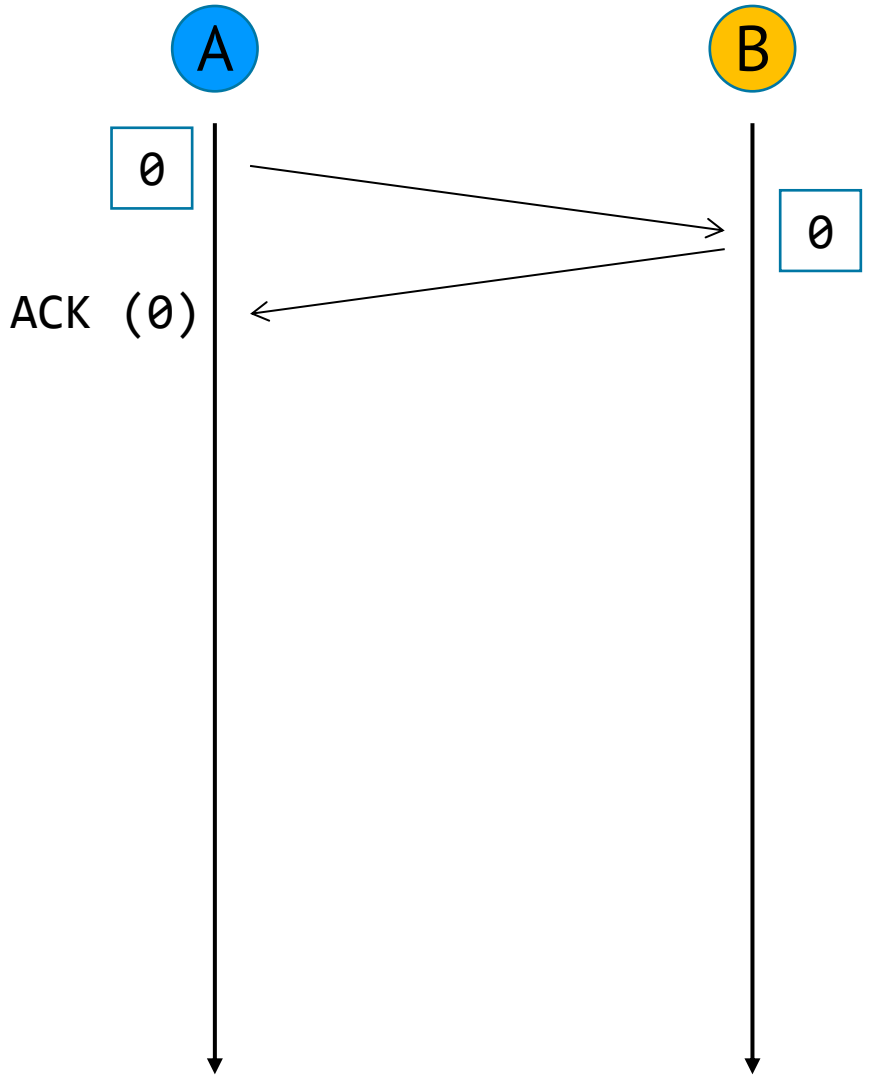




Nyugtázási módszerek

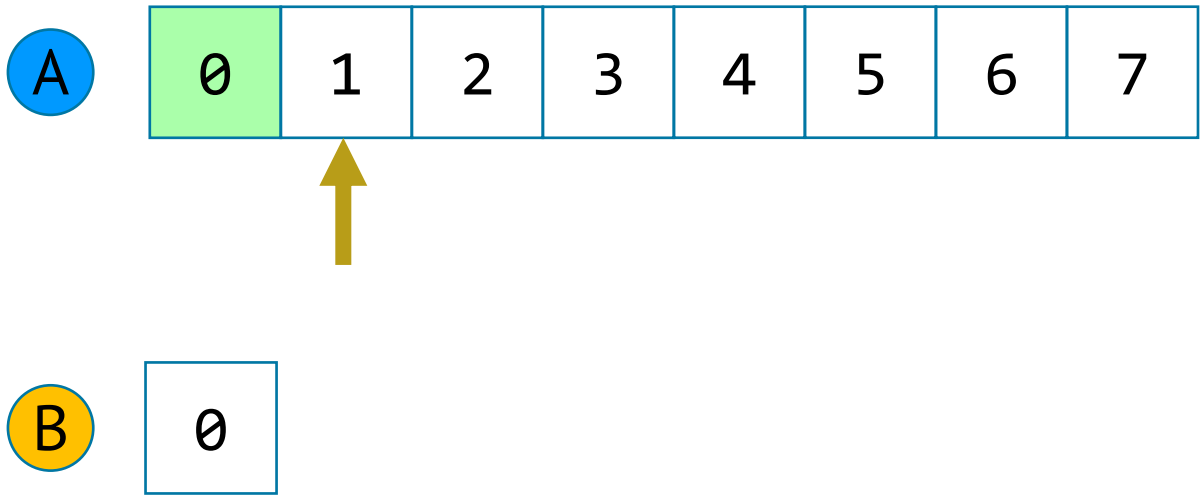
Stop-and-Wait protocol

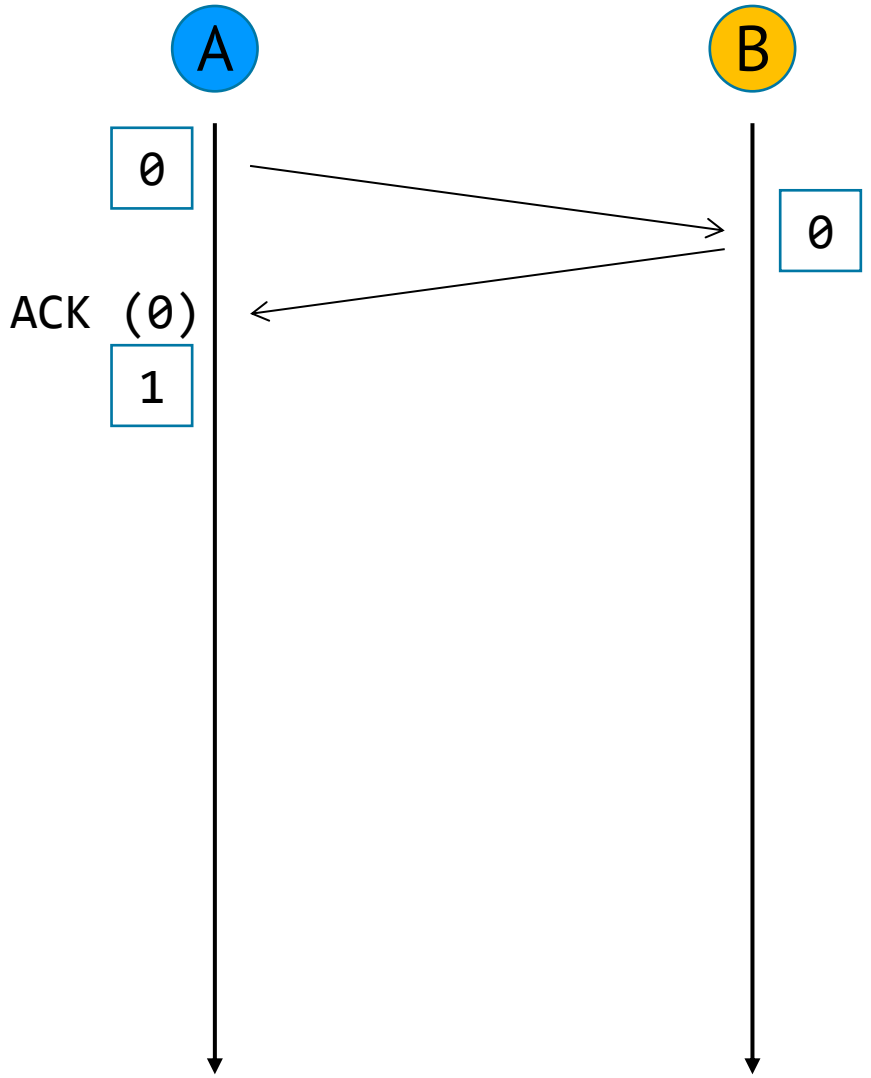




Nyugtázási módszerek

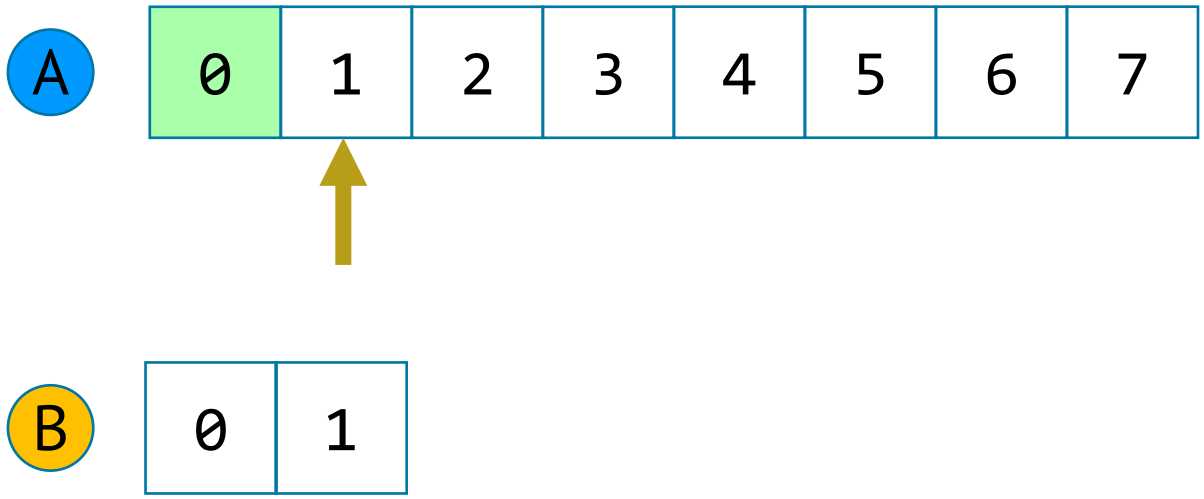
Stop-and-Wait protocol

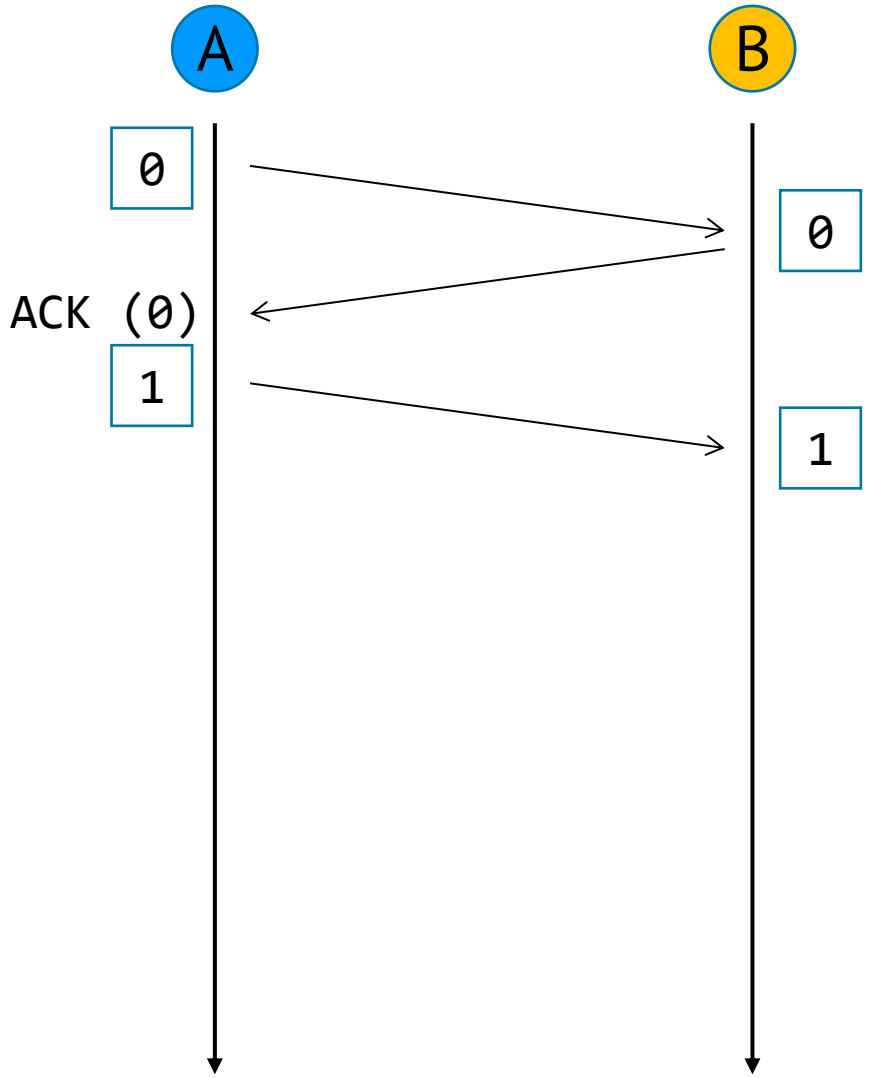




Nyugtázási módszerek

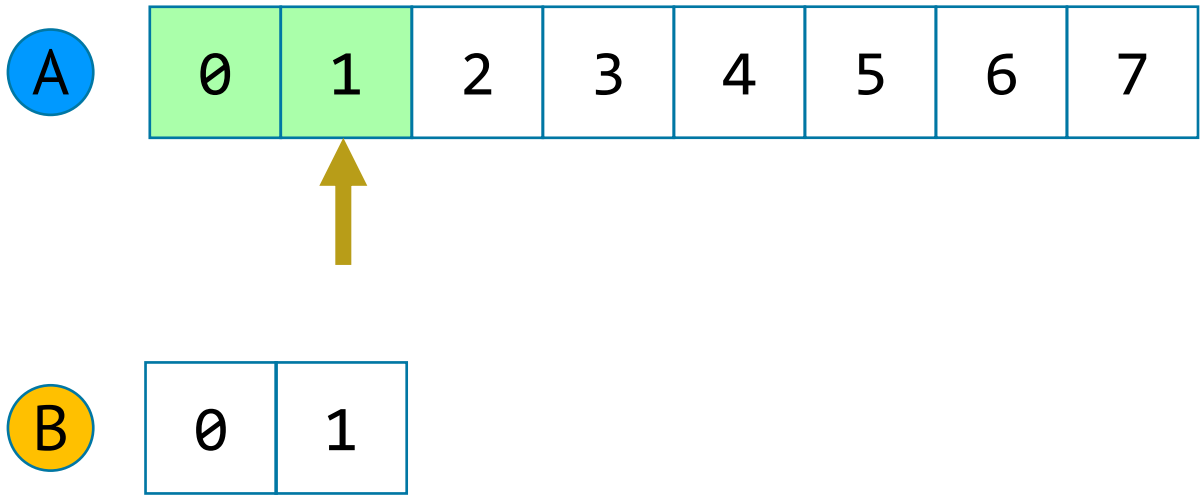
Stop-and-Wait protocol

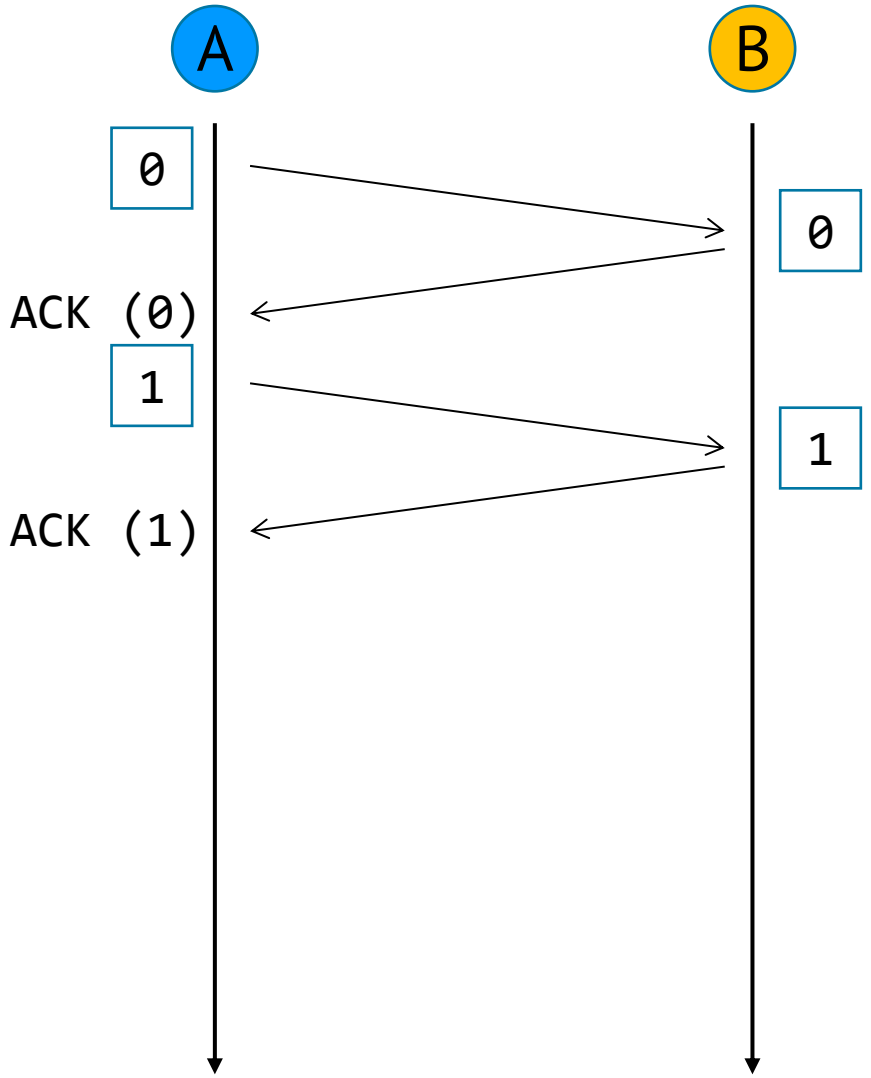




Nyugtázási módszerek

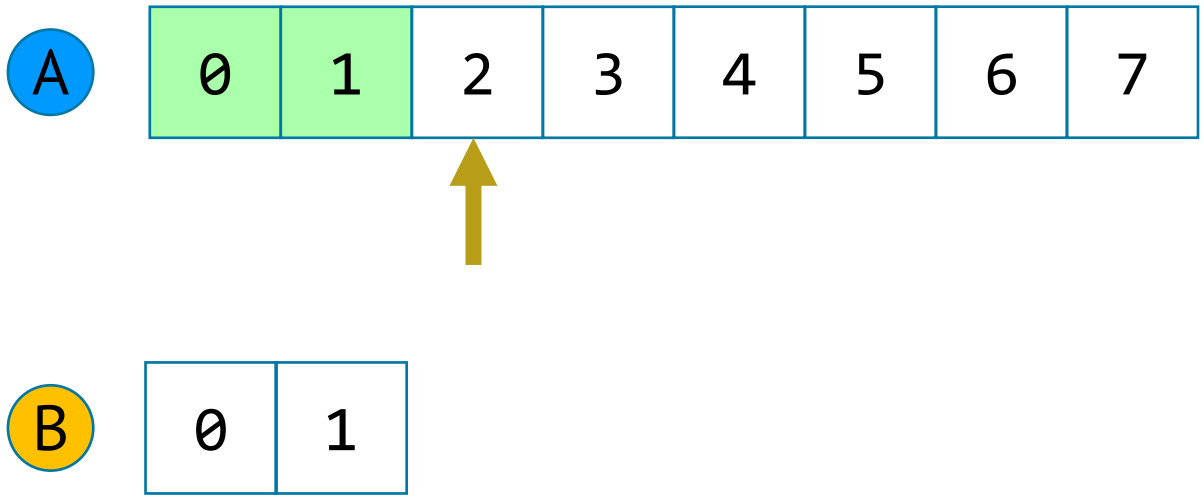
Stop-and-Wait protocol

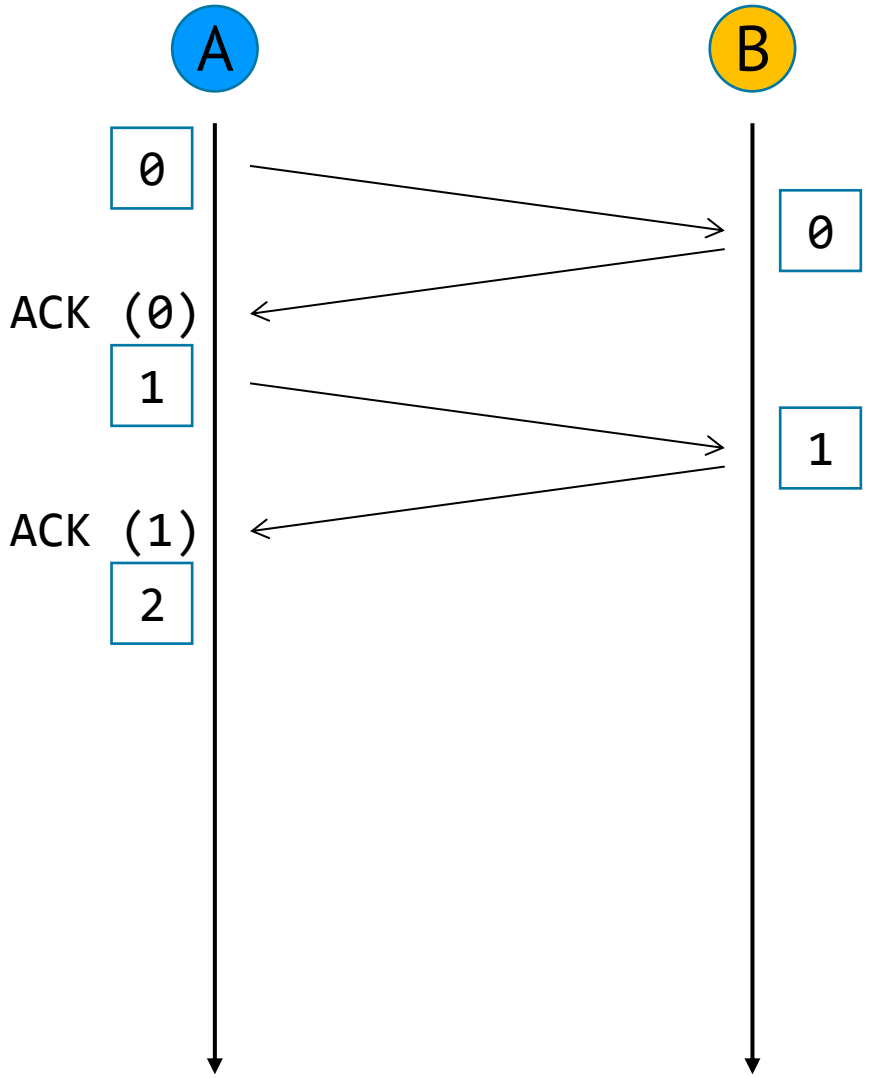




Nyugtázási módszerek

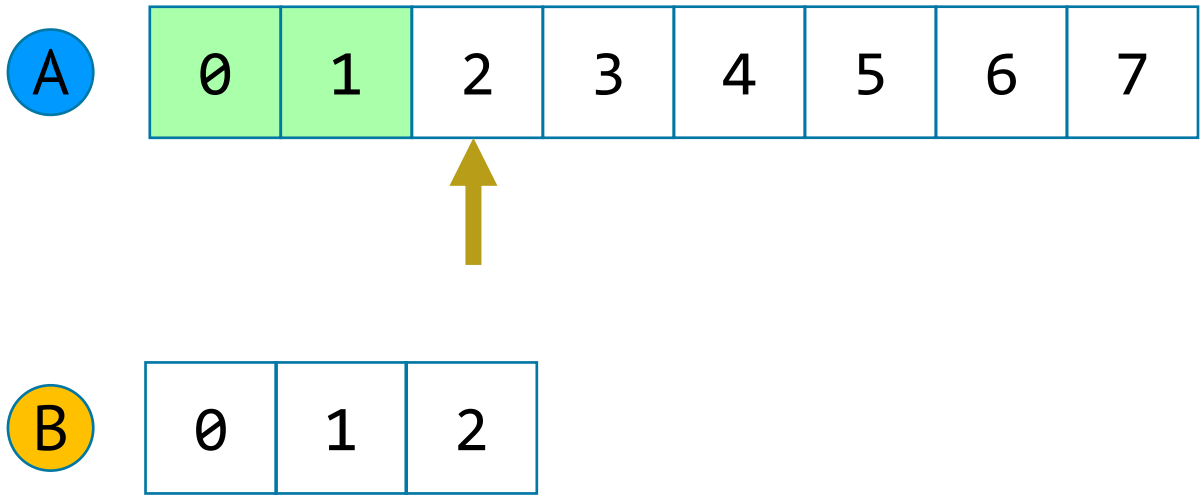
Stop-and-Wait protocol

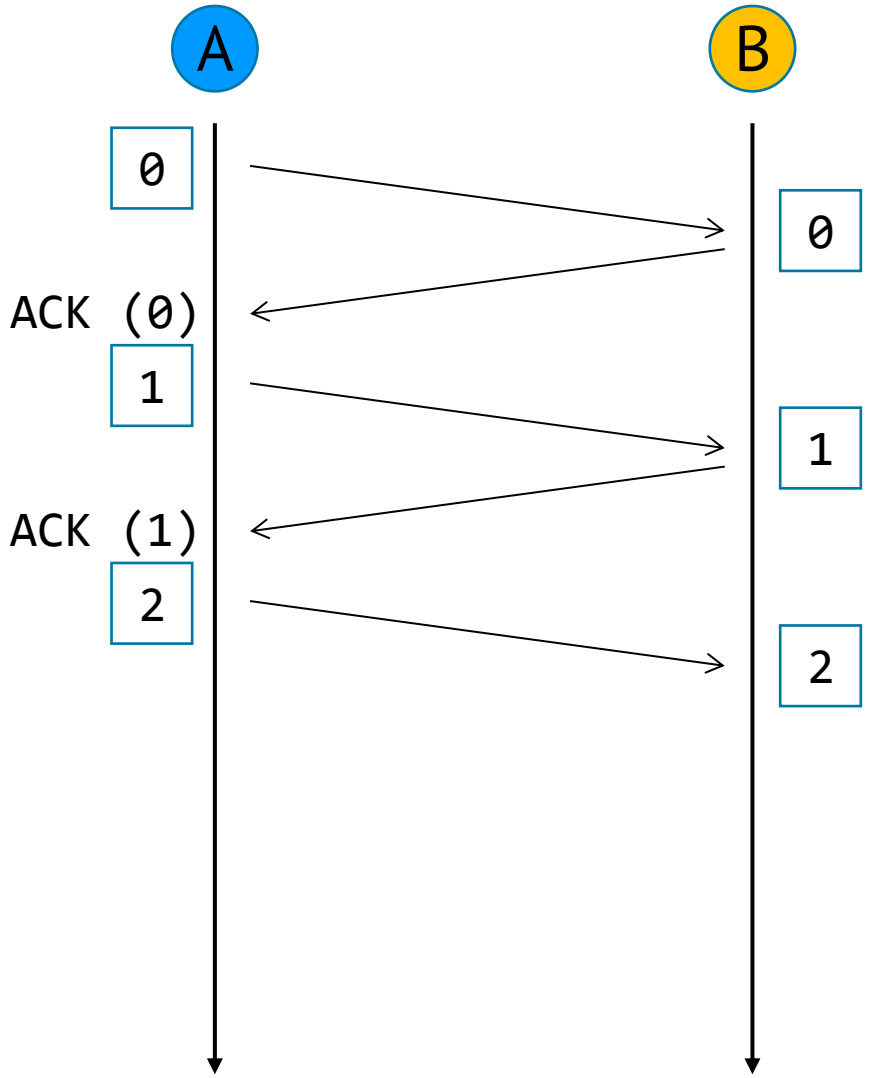




Nyugtázási módszerek

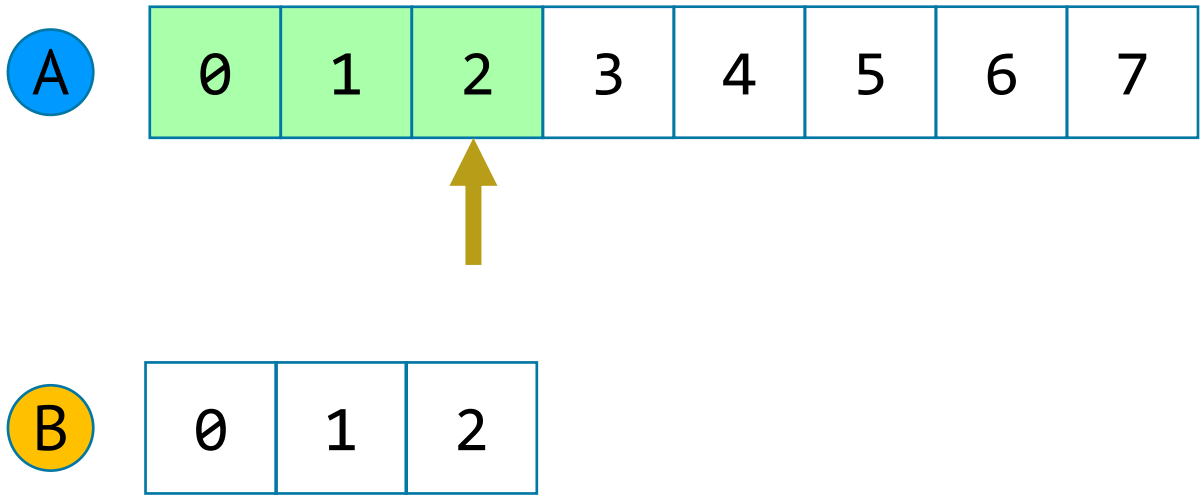
Stop-and-Wait protocol

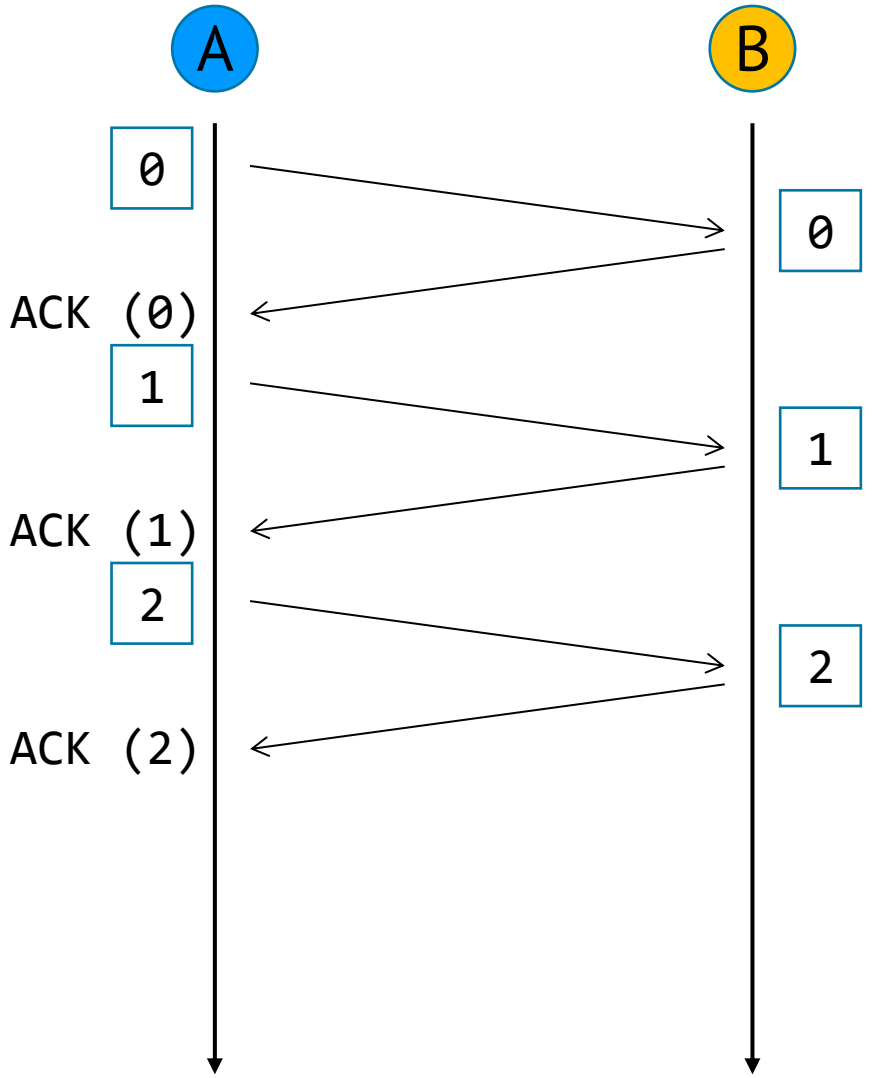




Nyugtázási módszerek

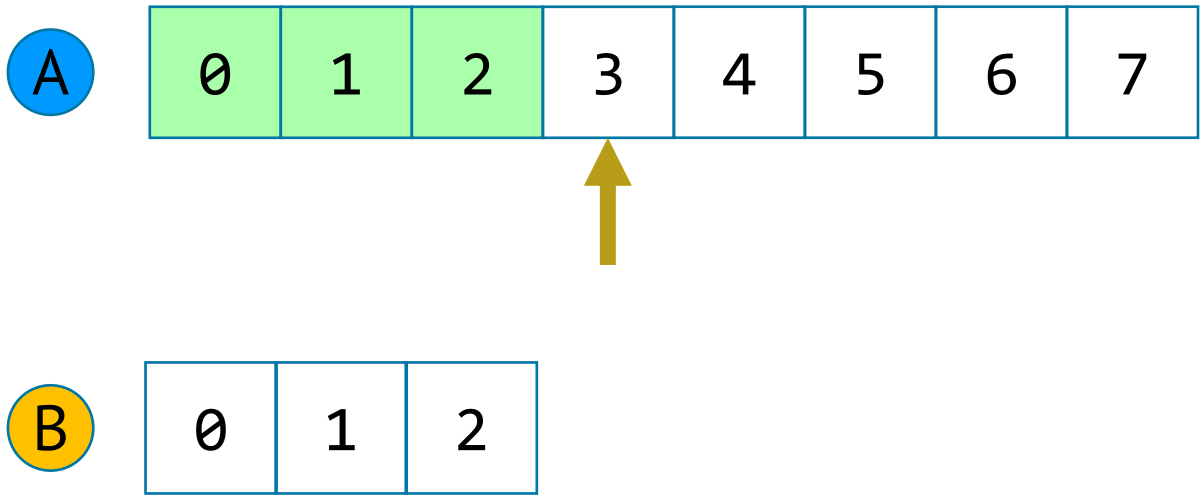
Stop-and-Wait protocol

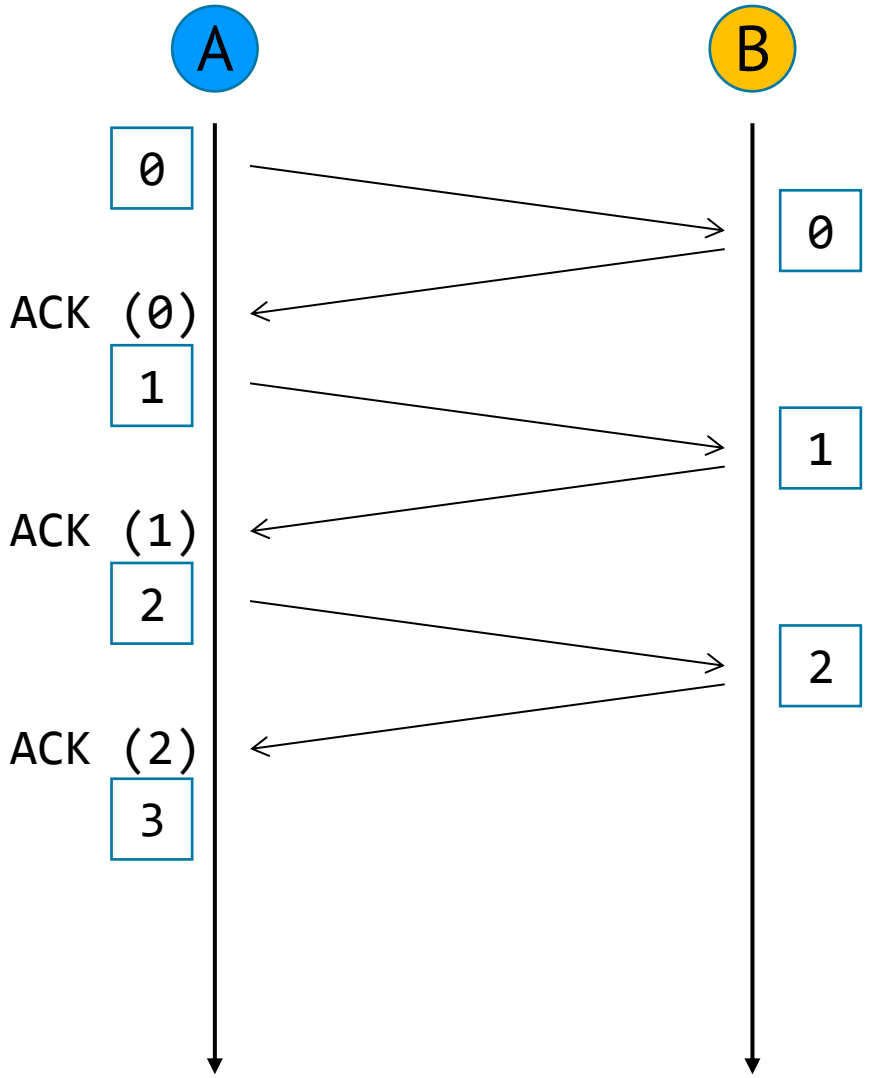




Nyugtázási módszerek

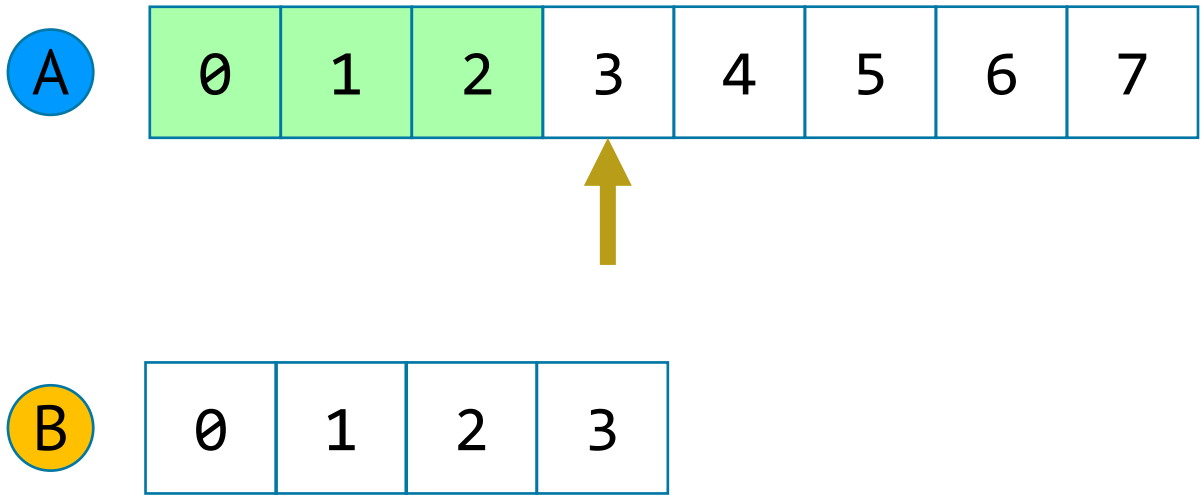
Stop-and-Wait protocol

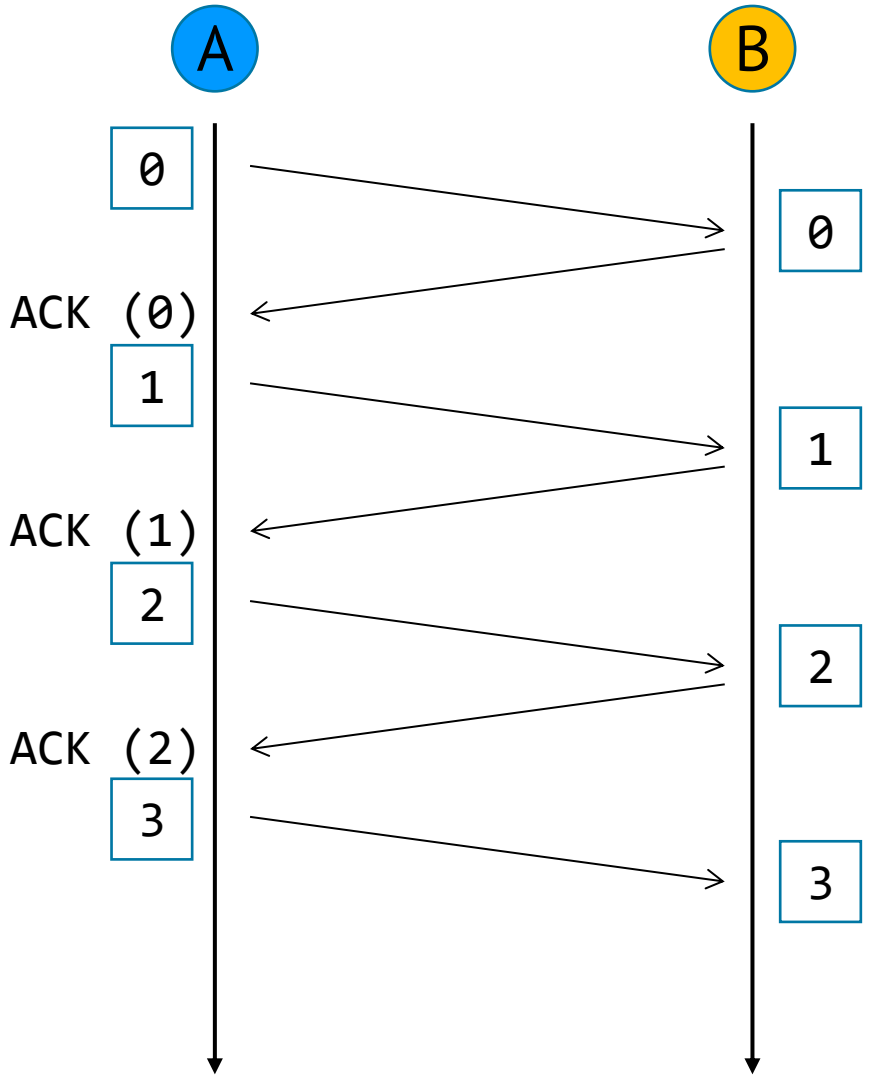




Nyugtázási módszerek

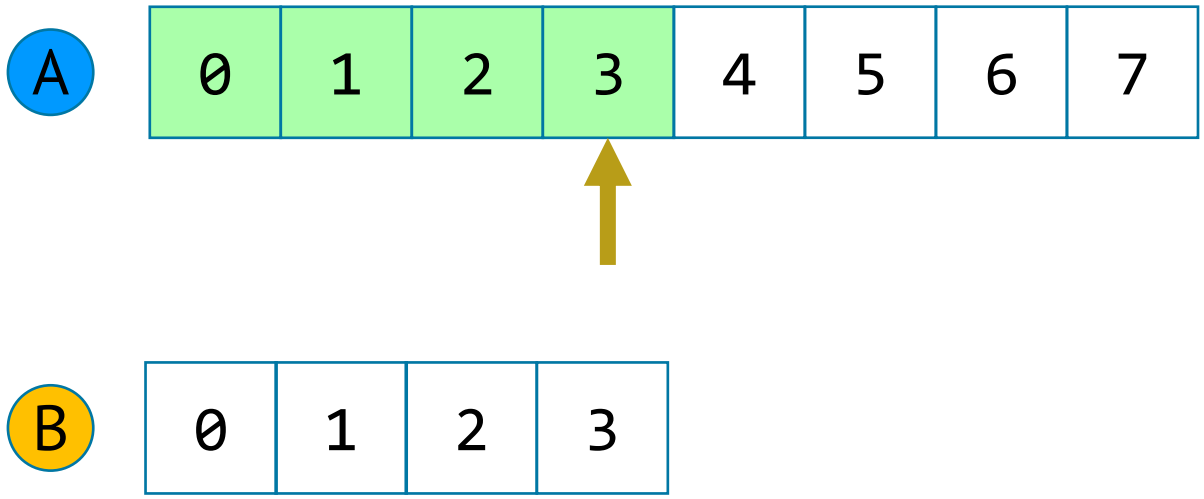
Stop-and-Wait protocol

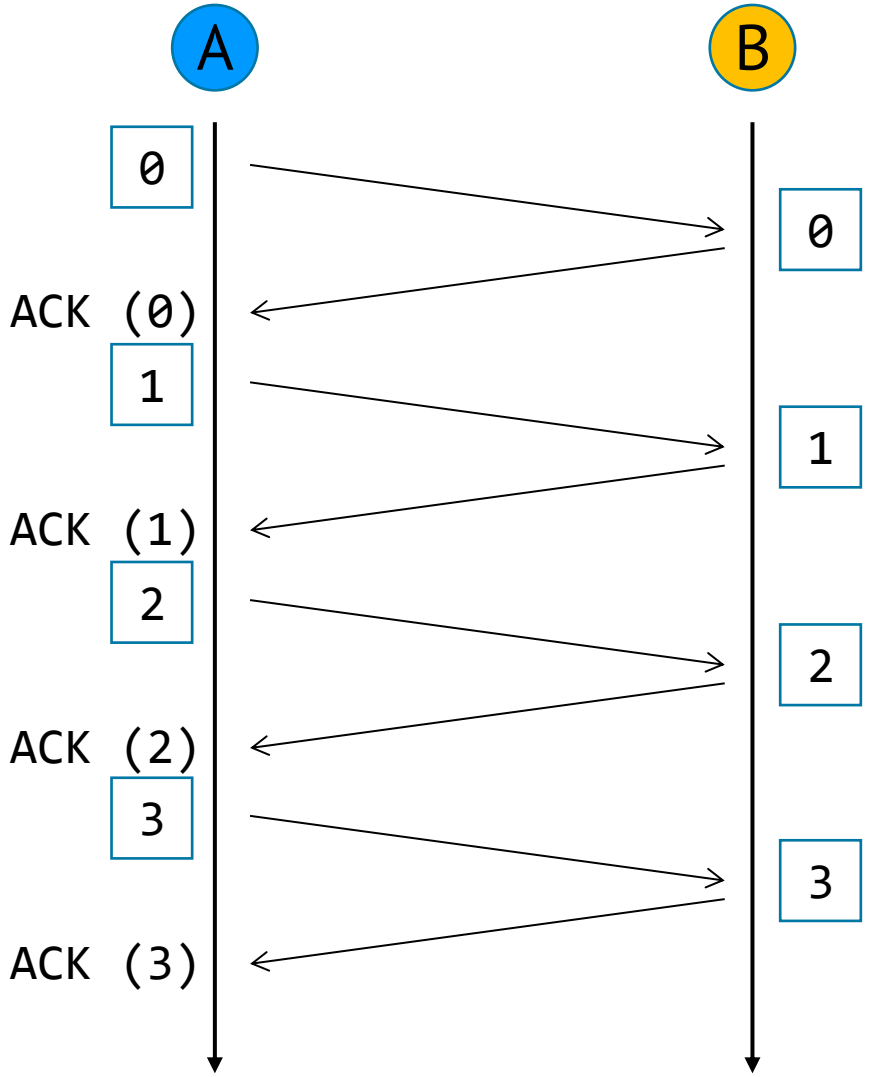




Nyugtázási módszerek

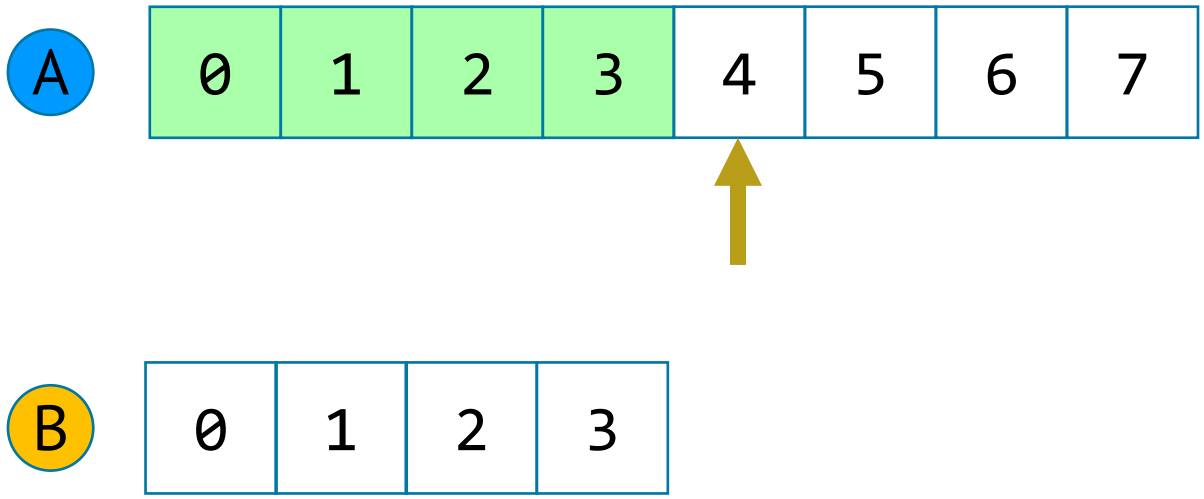
Stop-and-Wait protocol

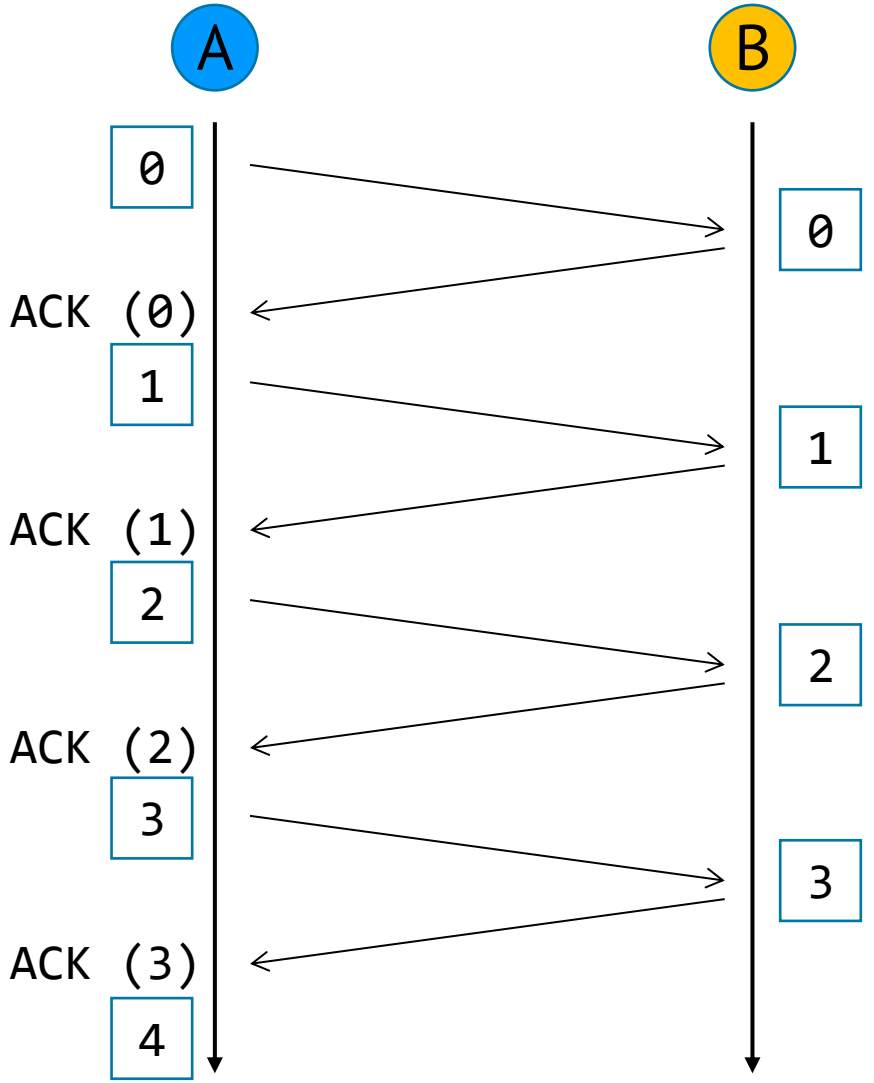




Nyugtázási módszerek

Stop-and-Wait protocol





Stop-and-Wait protocol megállapítások

Mivel egyesével küld, ezért lassú.

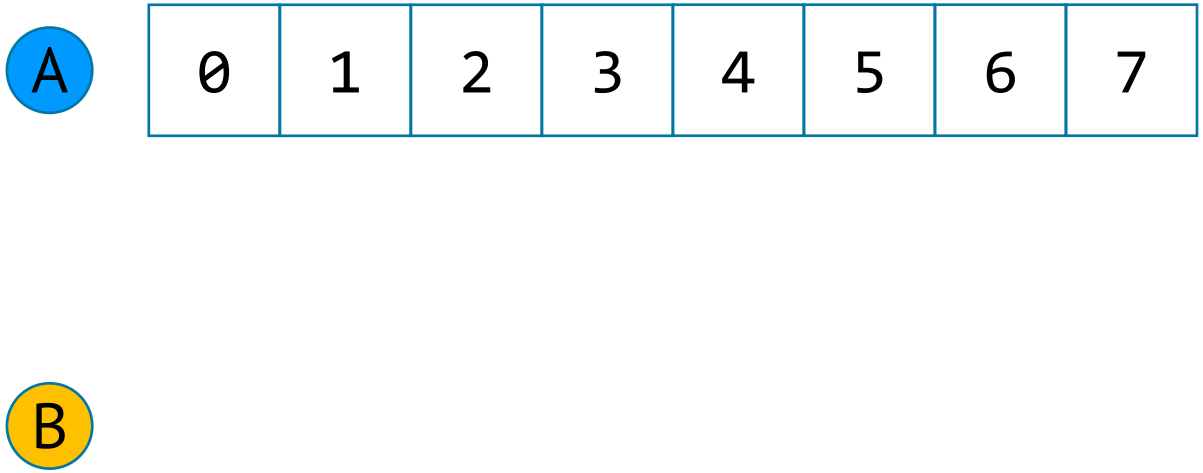
Mivel egyesével küld és vár, ezért nem használja ki a sávszélességet.

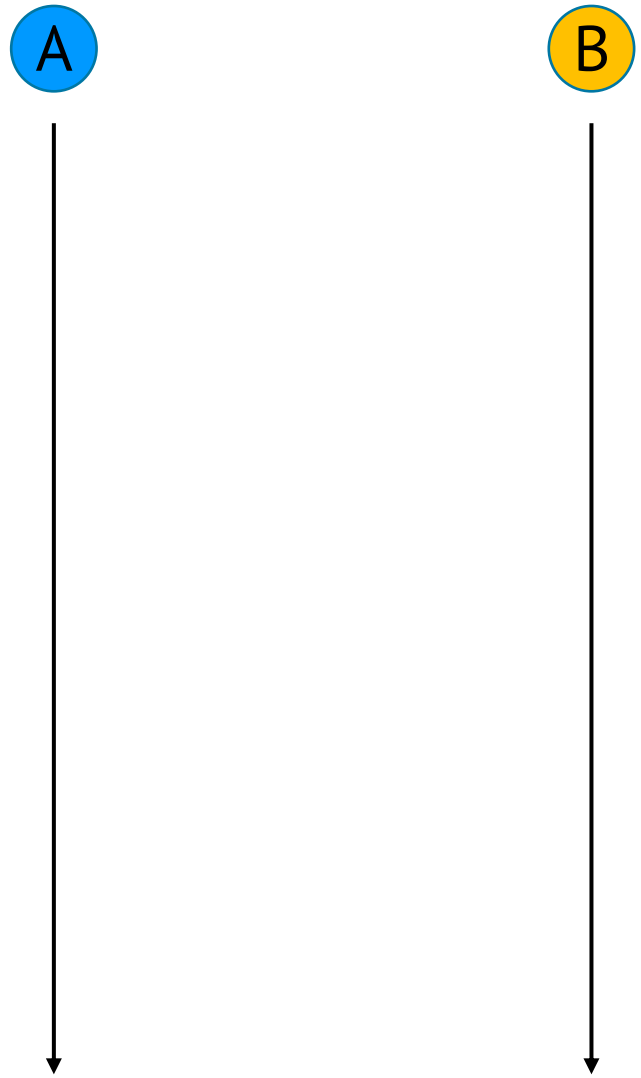
Nem beszél „fölöslegesen”, csak akkor küld, ha a másik megértette az előzőt.

Van nála jobb megoldás is → sliding window protocol

Nyugtázási módszerek

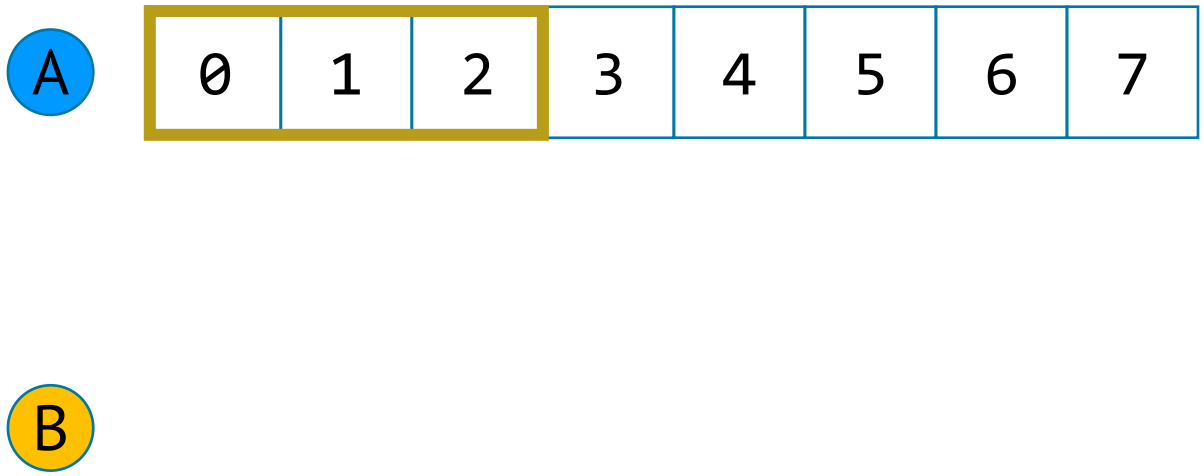
Sliding window protocol

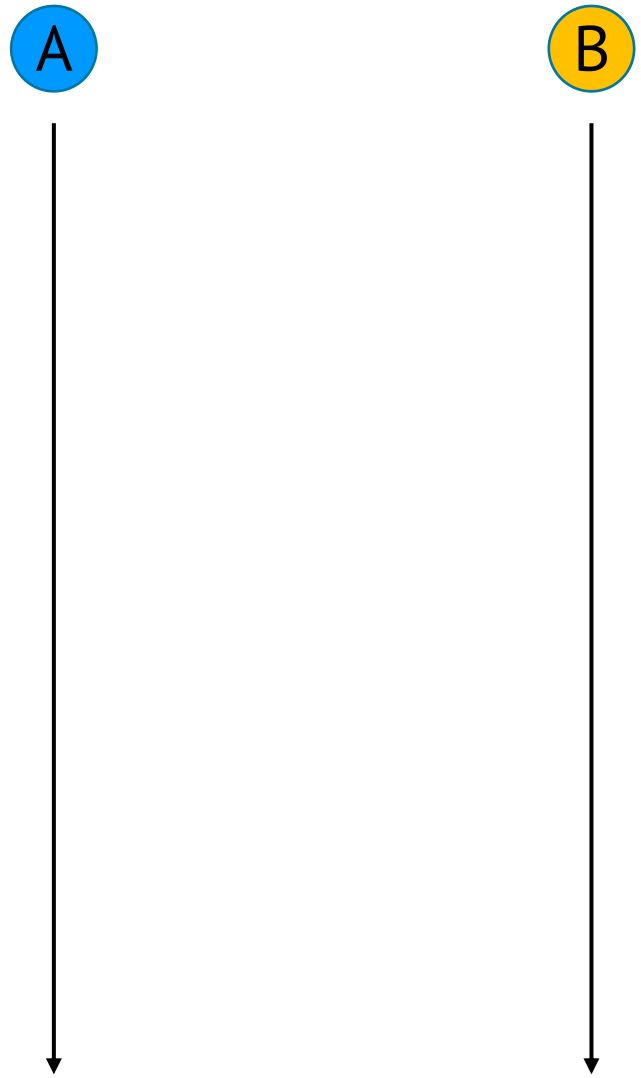




Nyugtázási módszerek

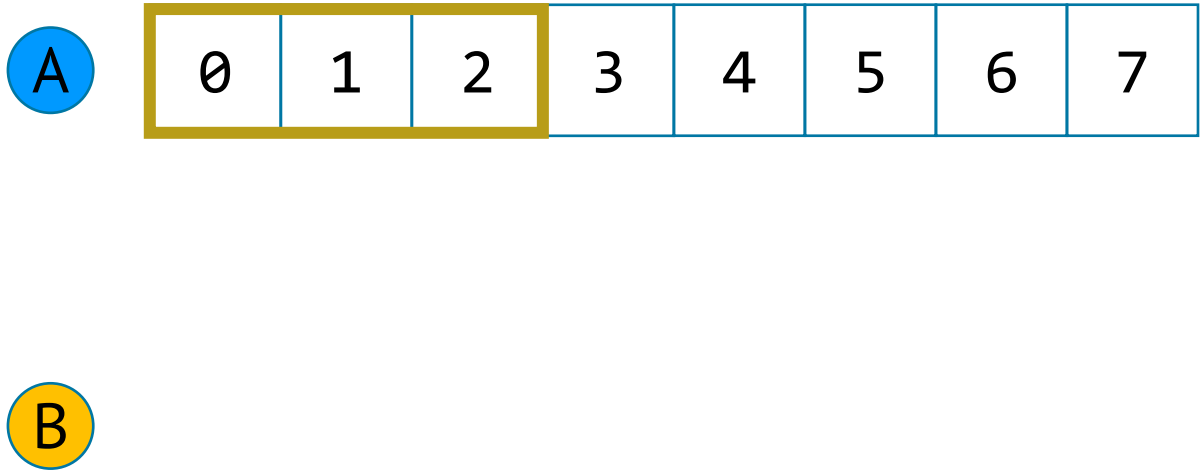
Sliding window protocol

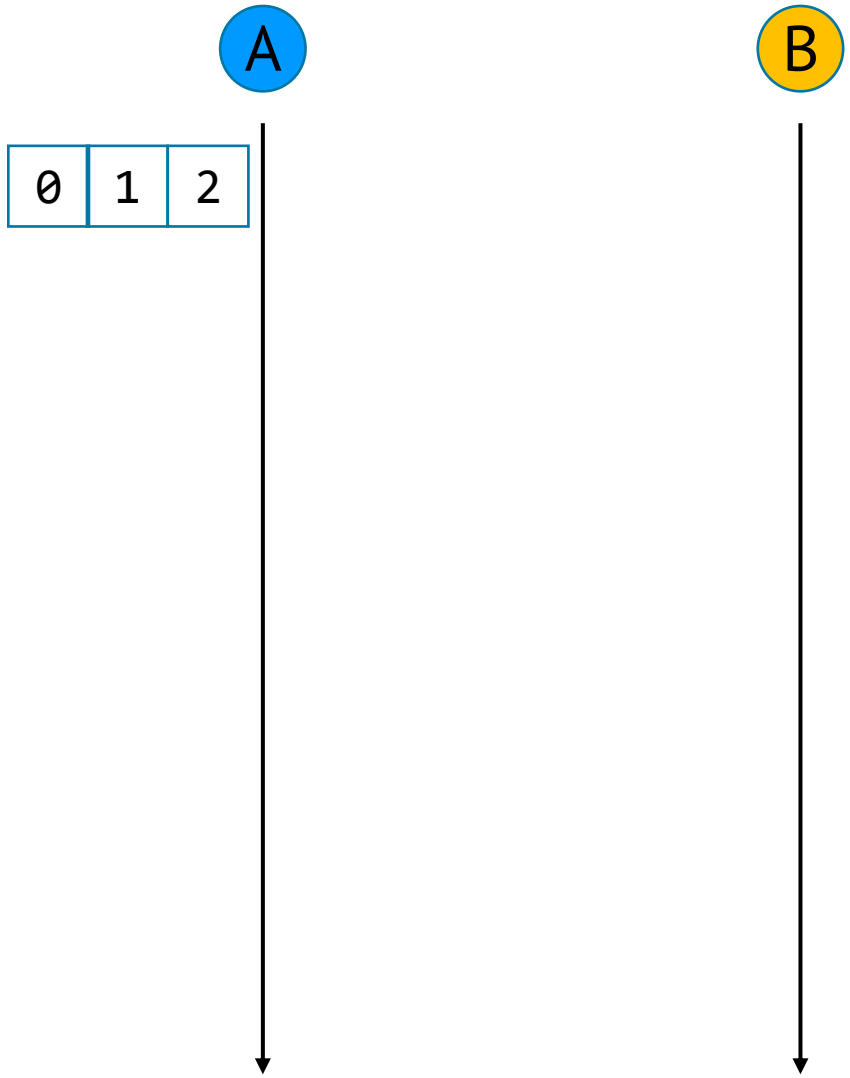




Nyugtázási módszerek

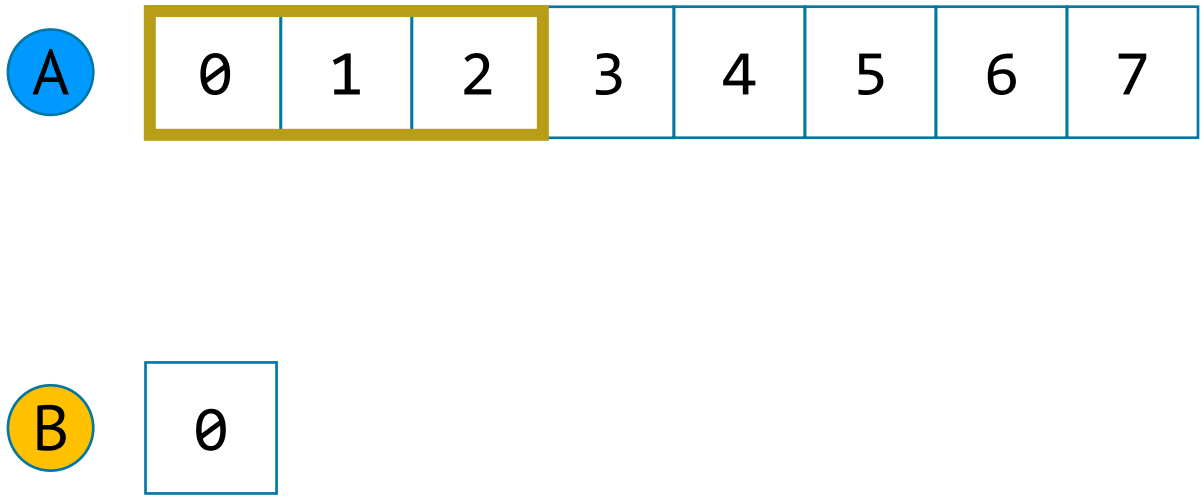
Sliding window protocol

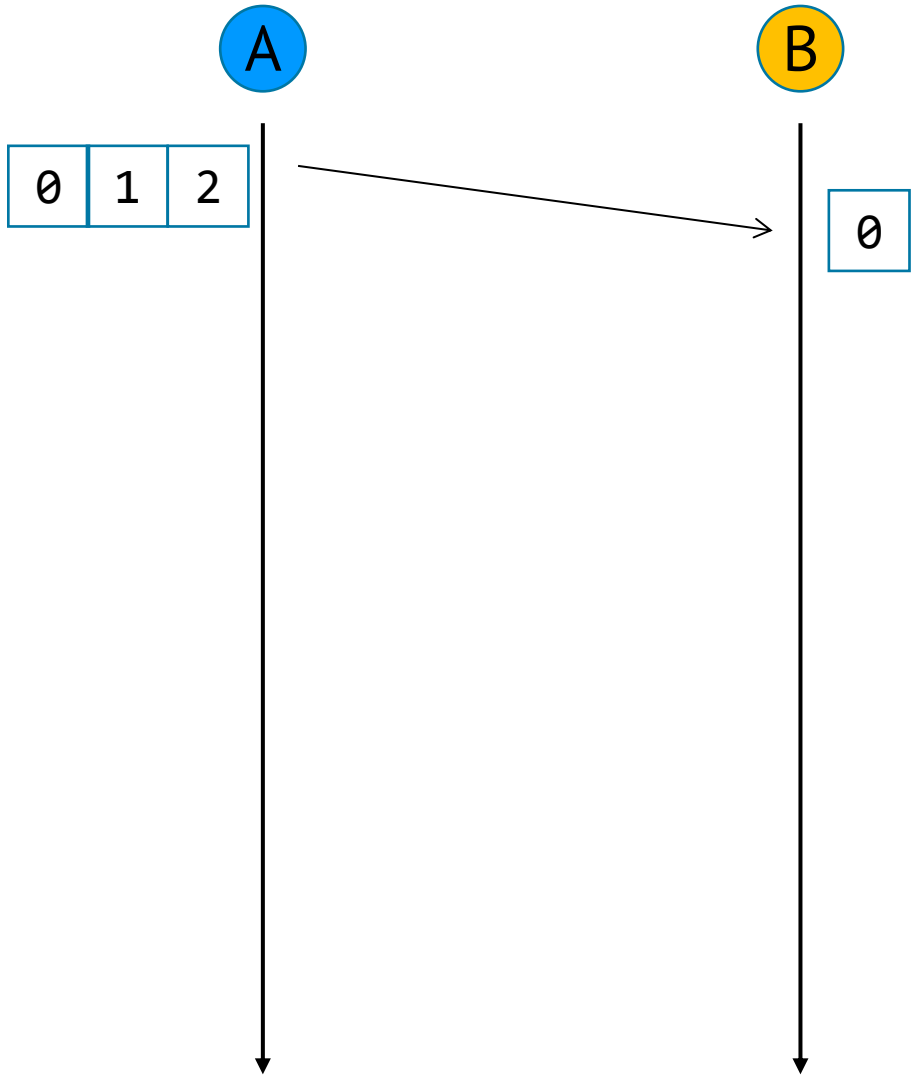




Nyugtázási módszerek

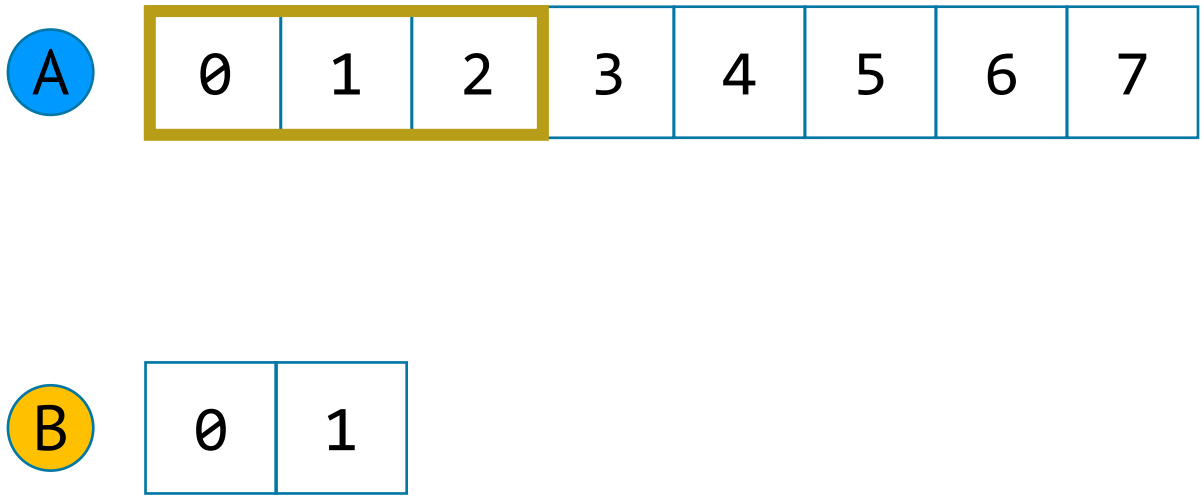
Sliding window protocol

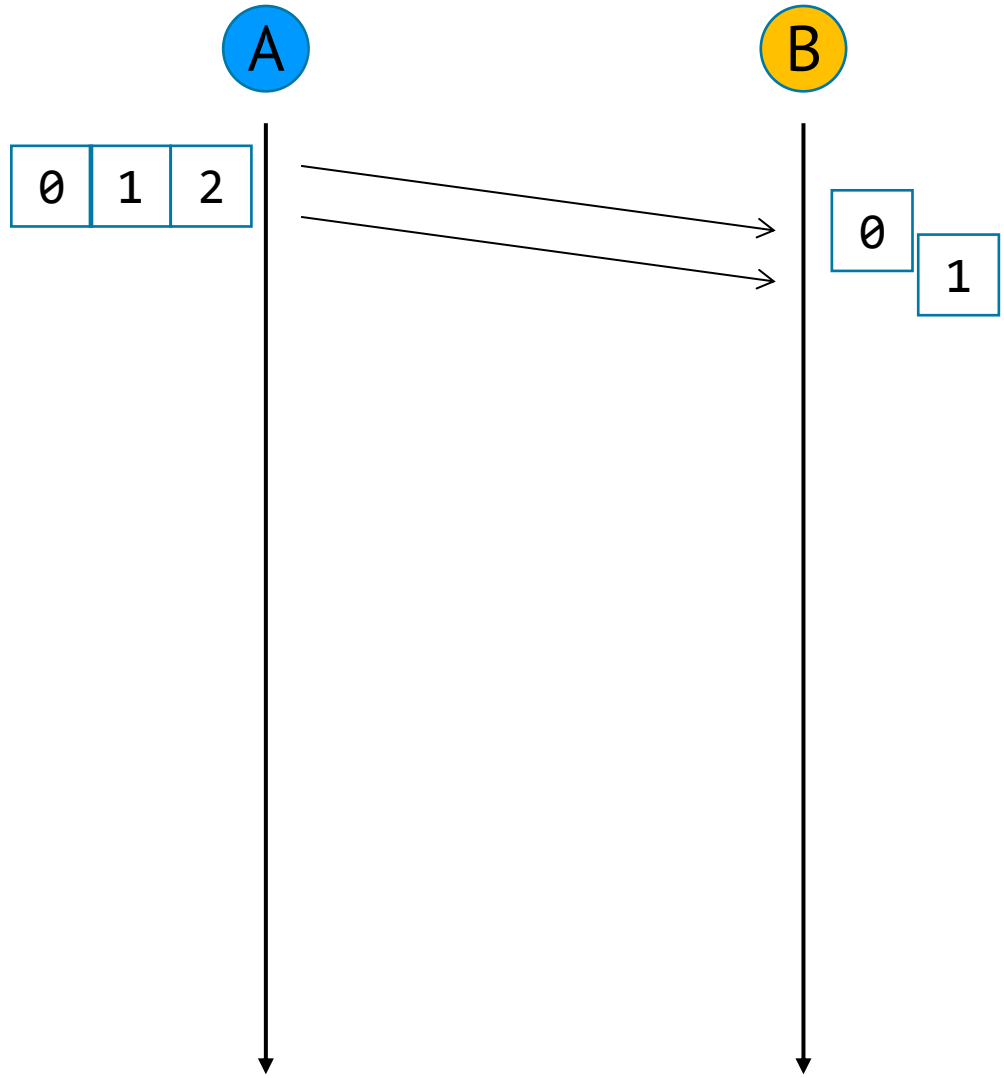




Nyugtázási módszerek

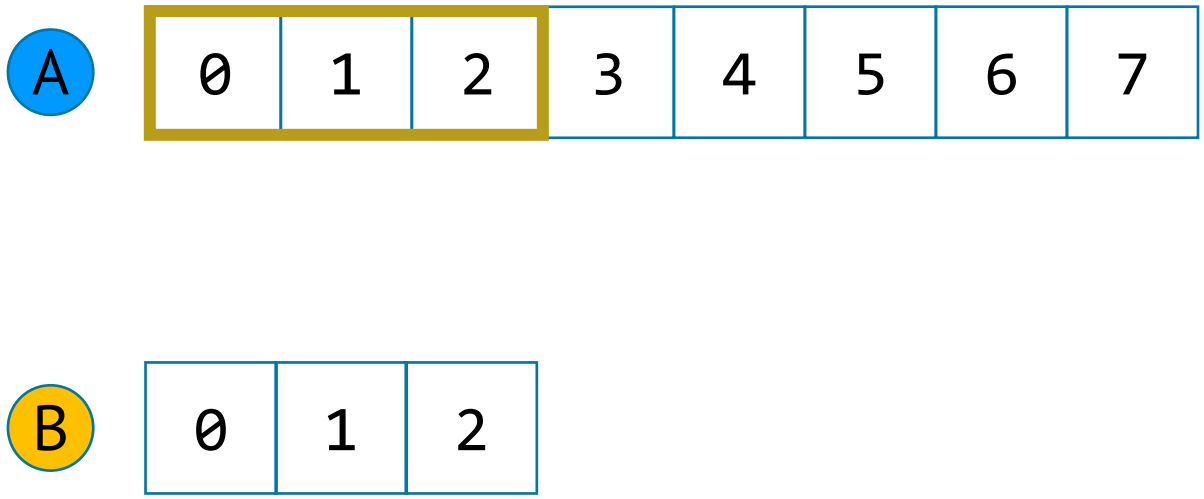
Sliding window protocol

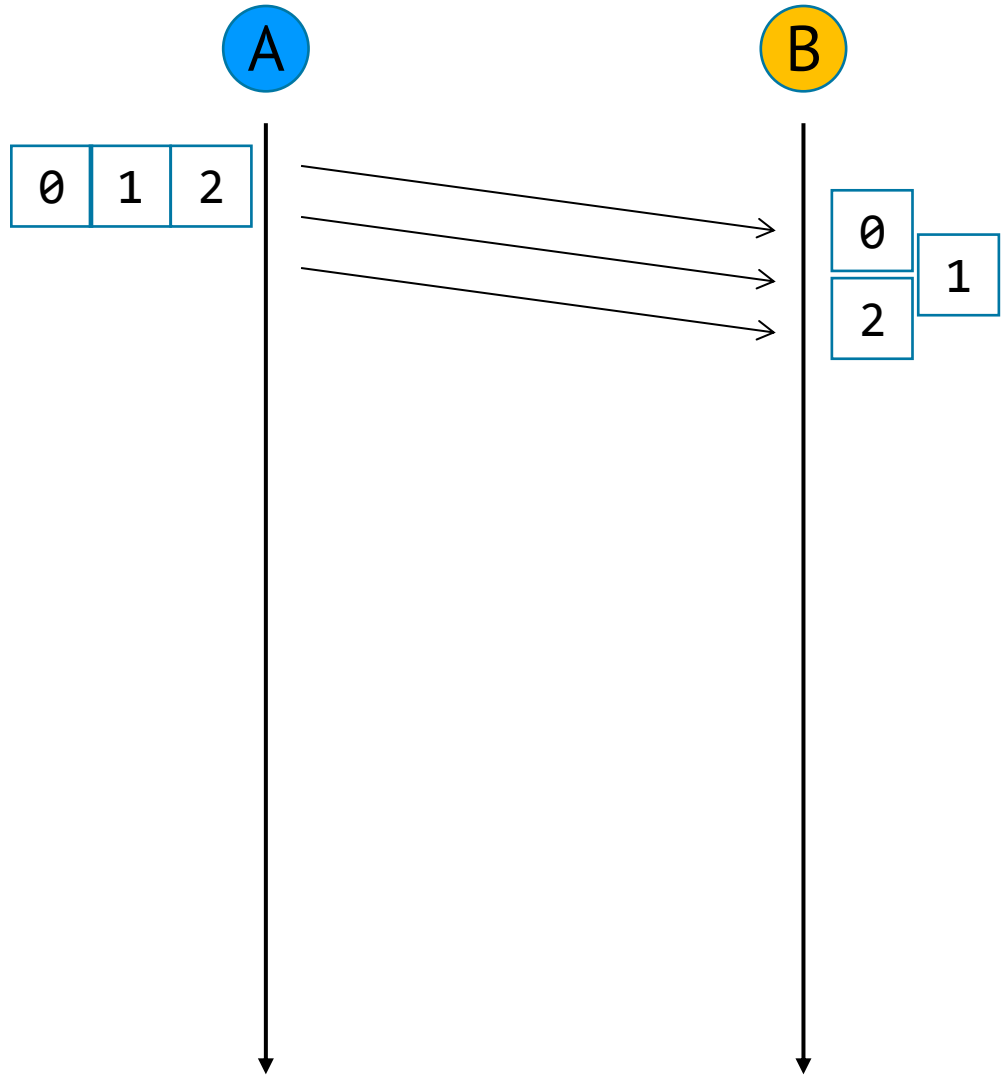




Nyugtázási módszerek

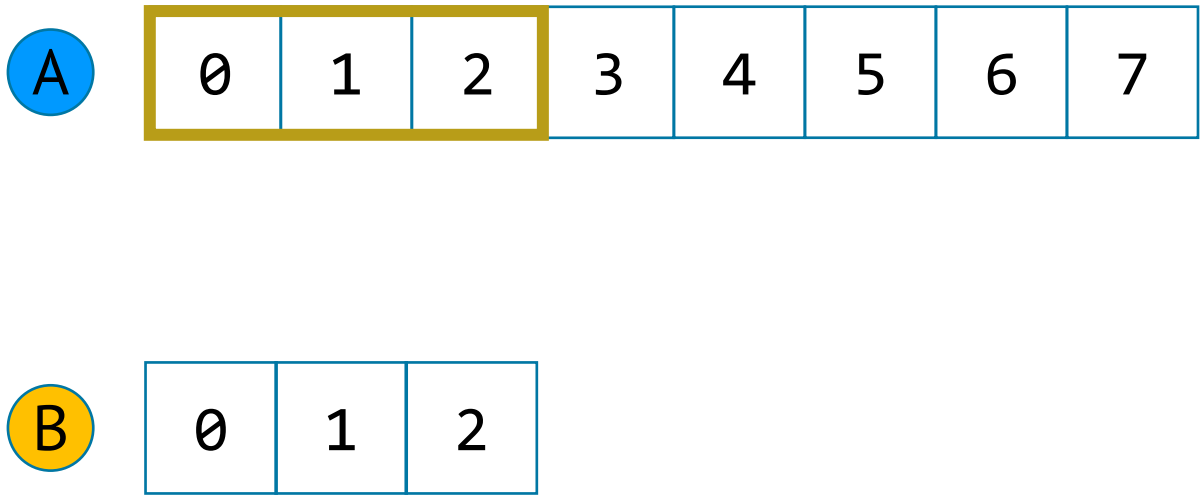
Sliding window protocol

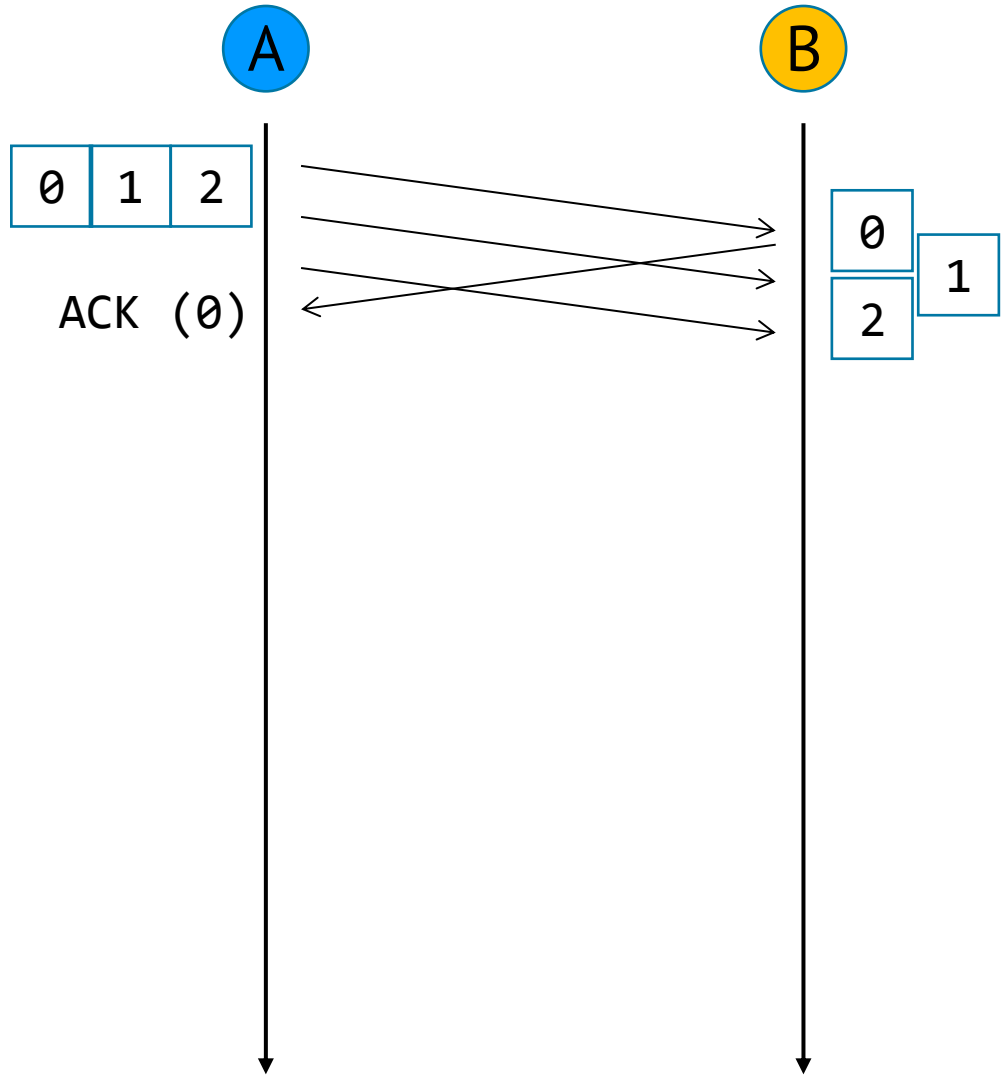




Nyugtázási módszerek

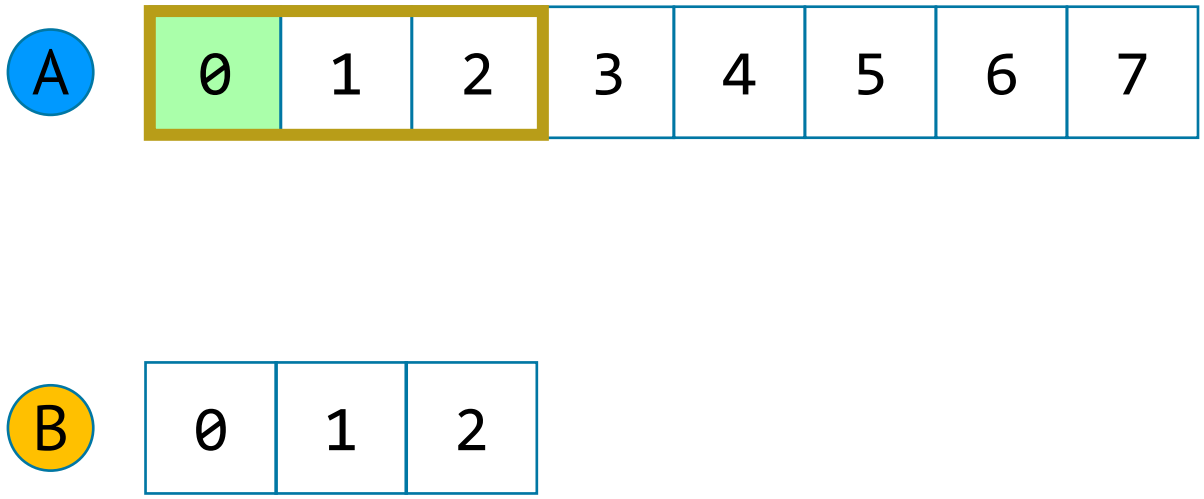
Sliding window protocol

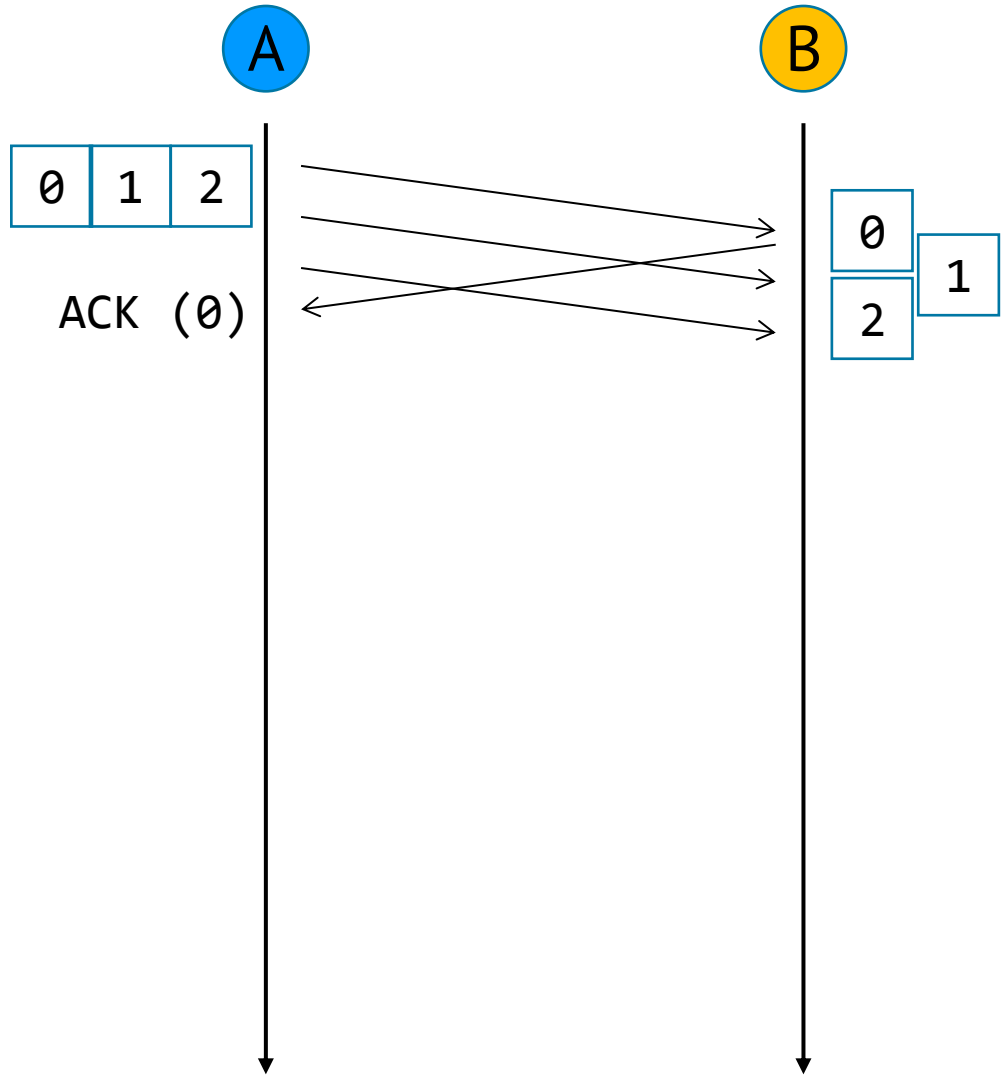




Nyugtázási módszerek

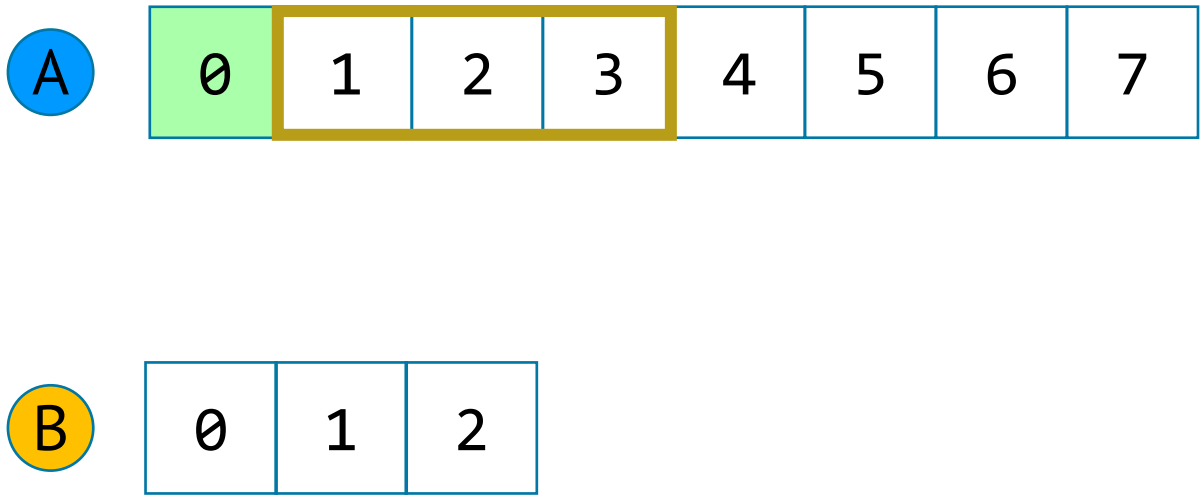
Sliding window protocol

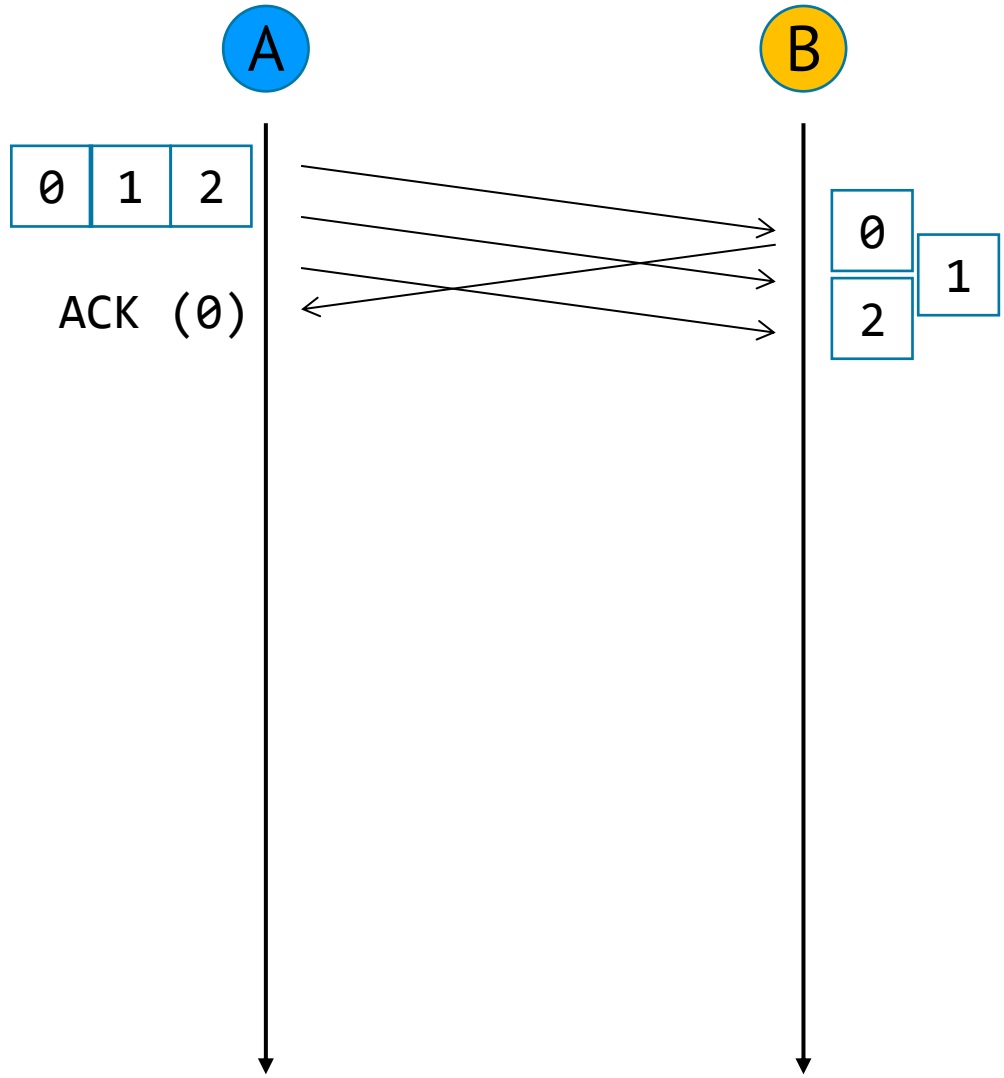




Nyugtázási módszerek

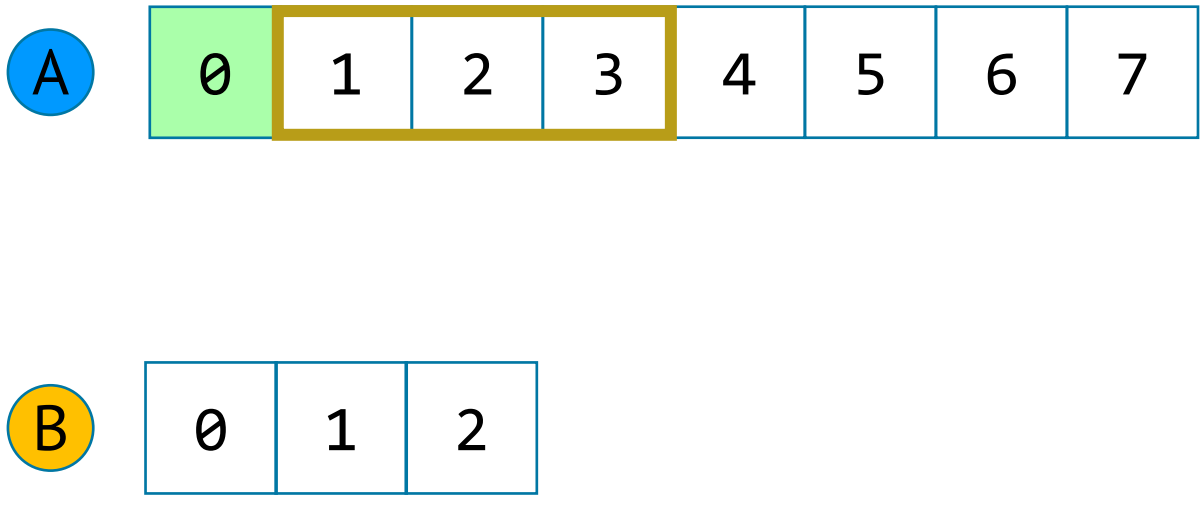
Sliding window protocol

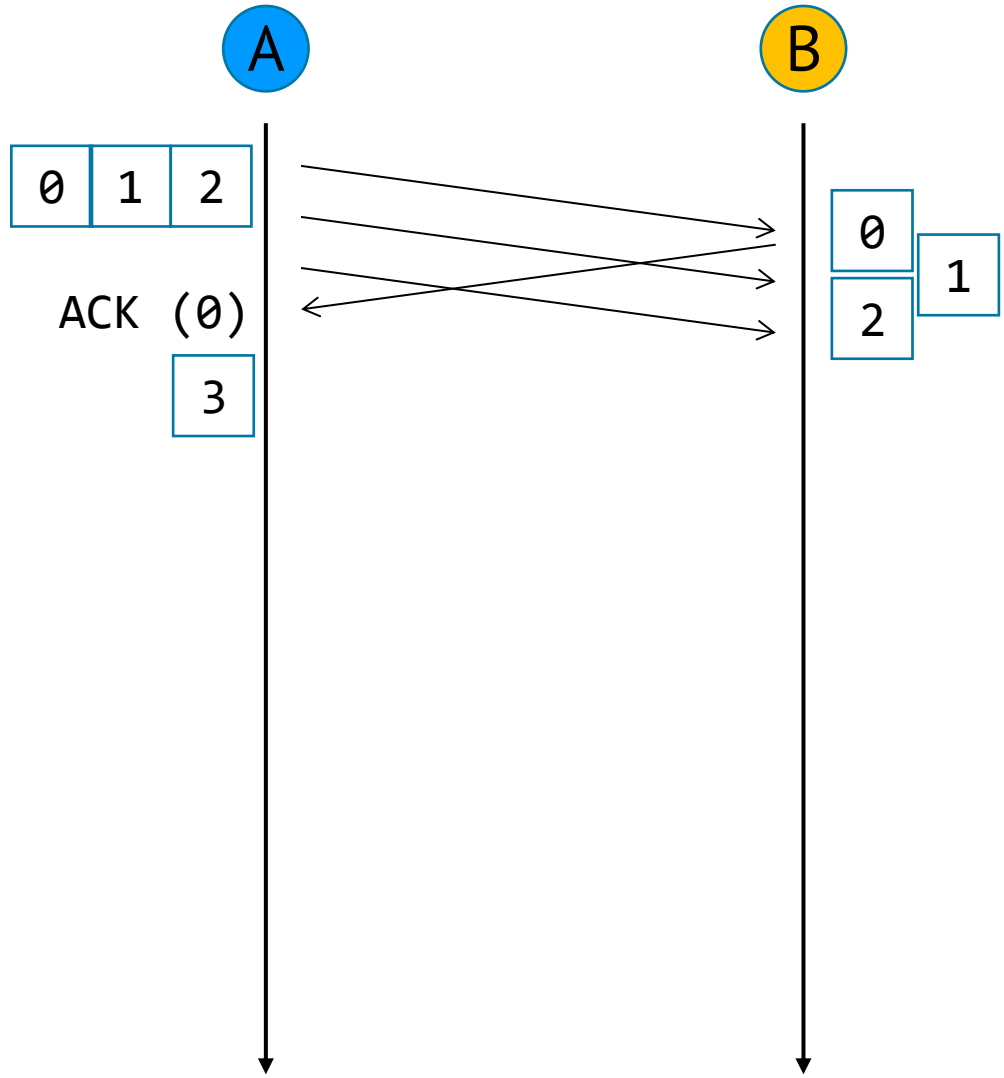




Nyugtázási módszerek

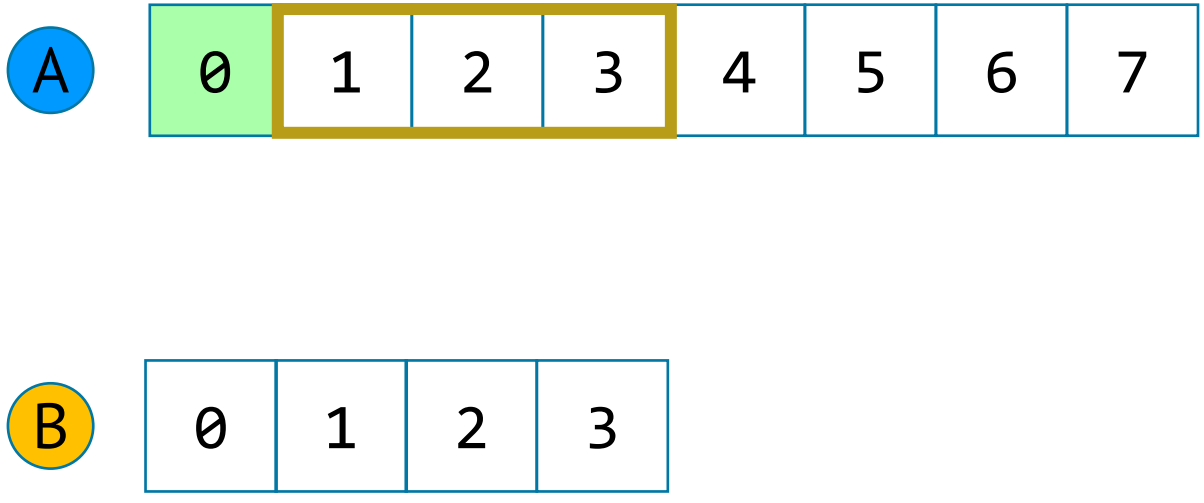
Sliding window protocol

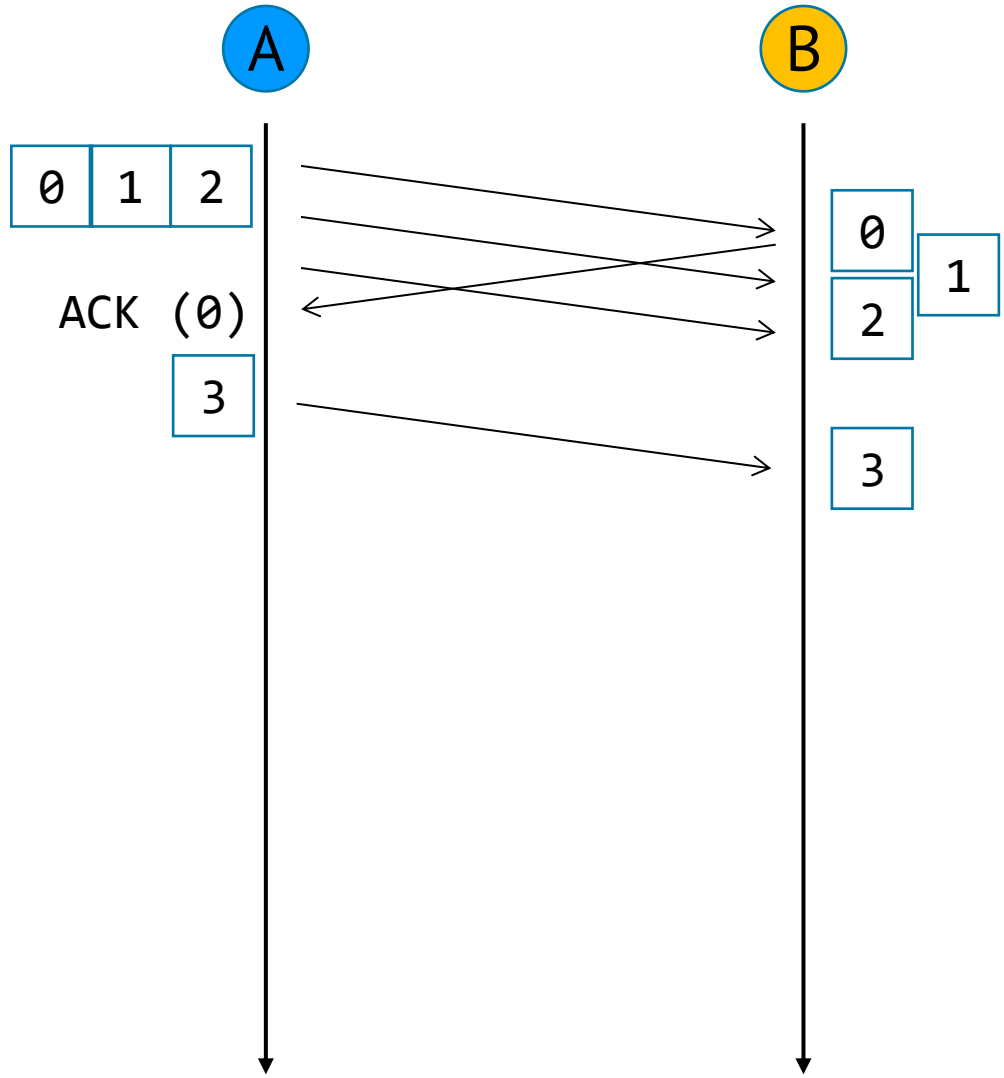




Nyugtázási módszerek

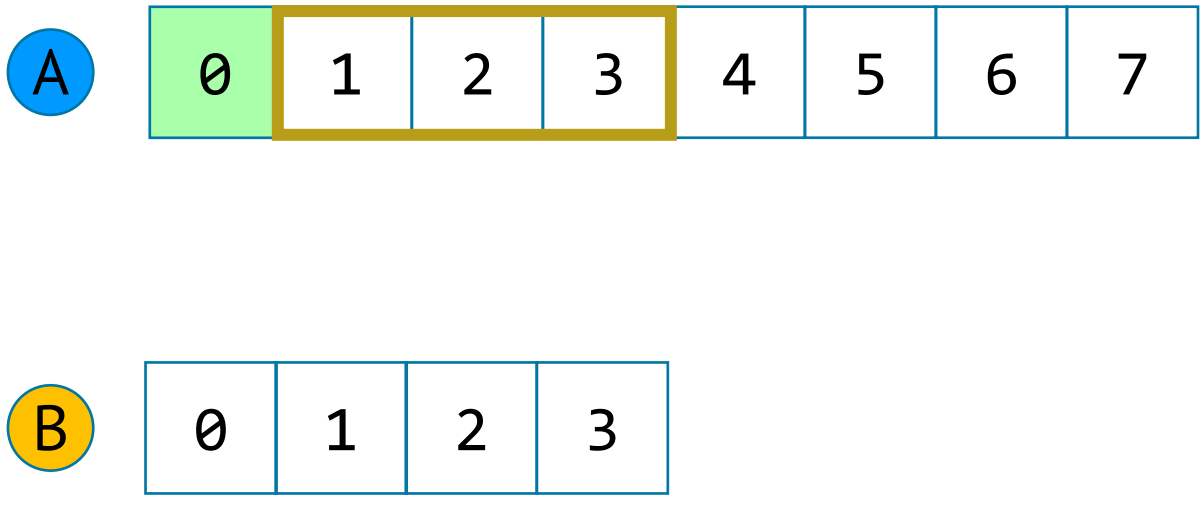
Sliding window protocol

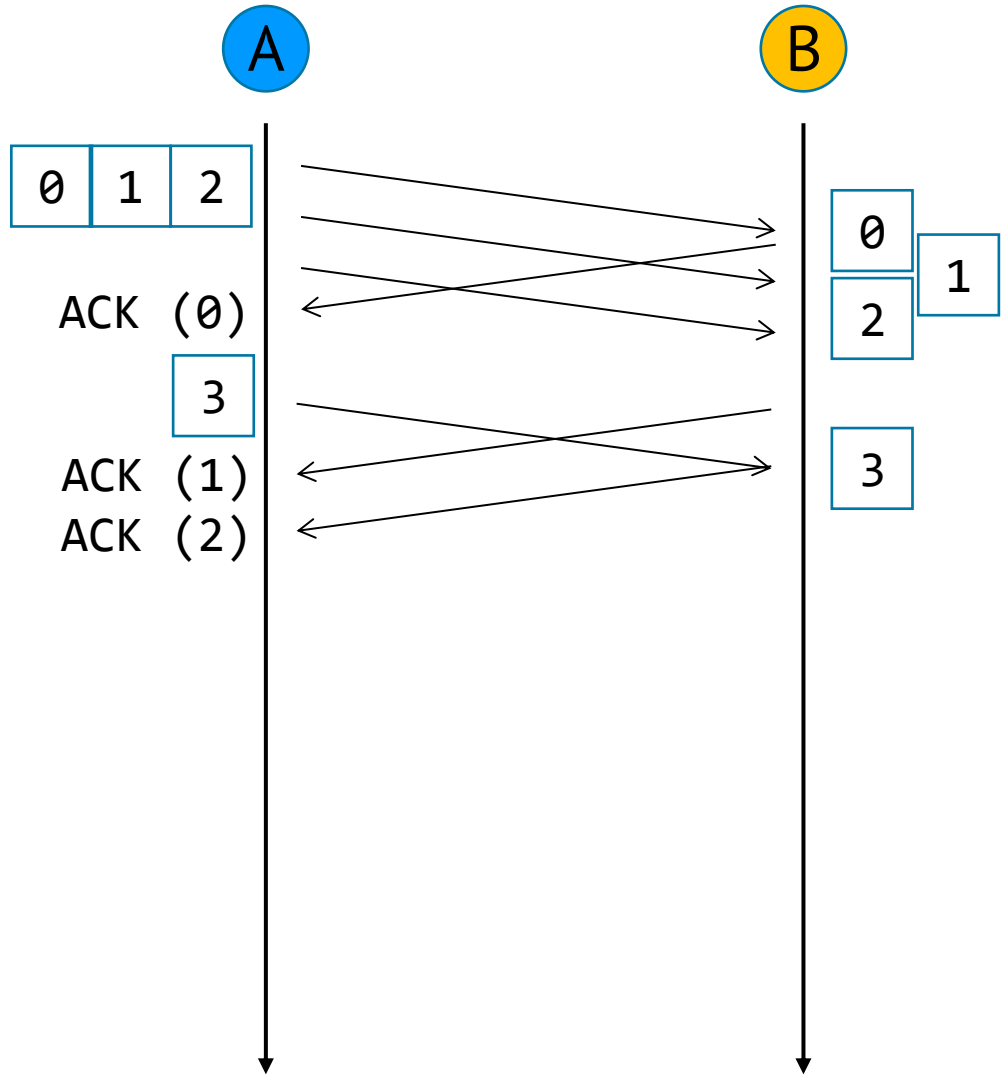




Nyugtázási módszerek

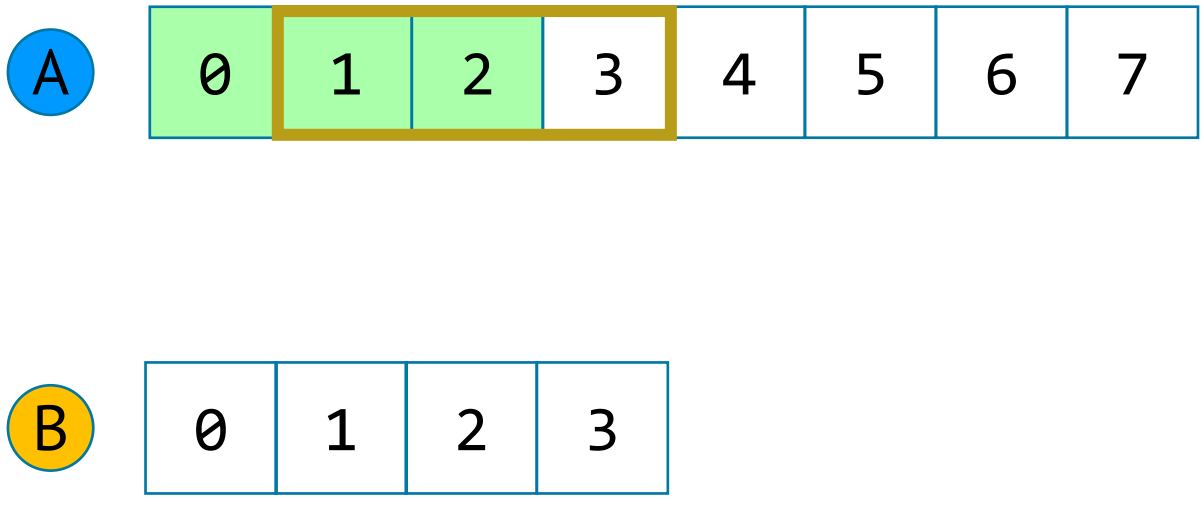
Sliding window protocol

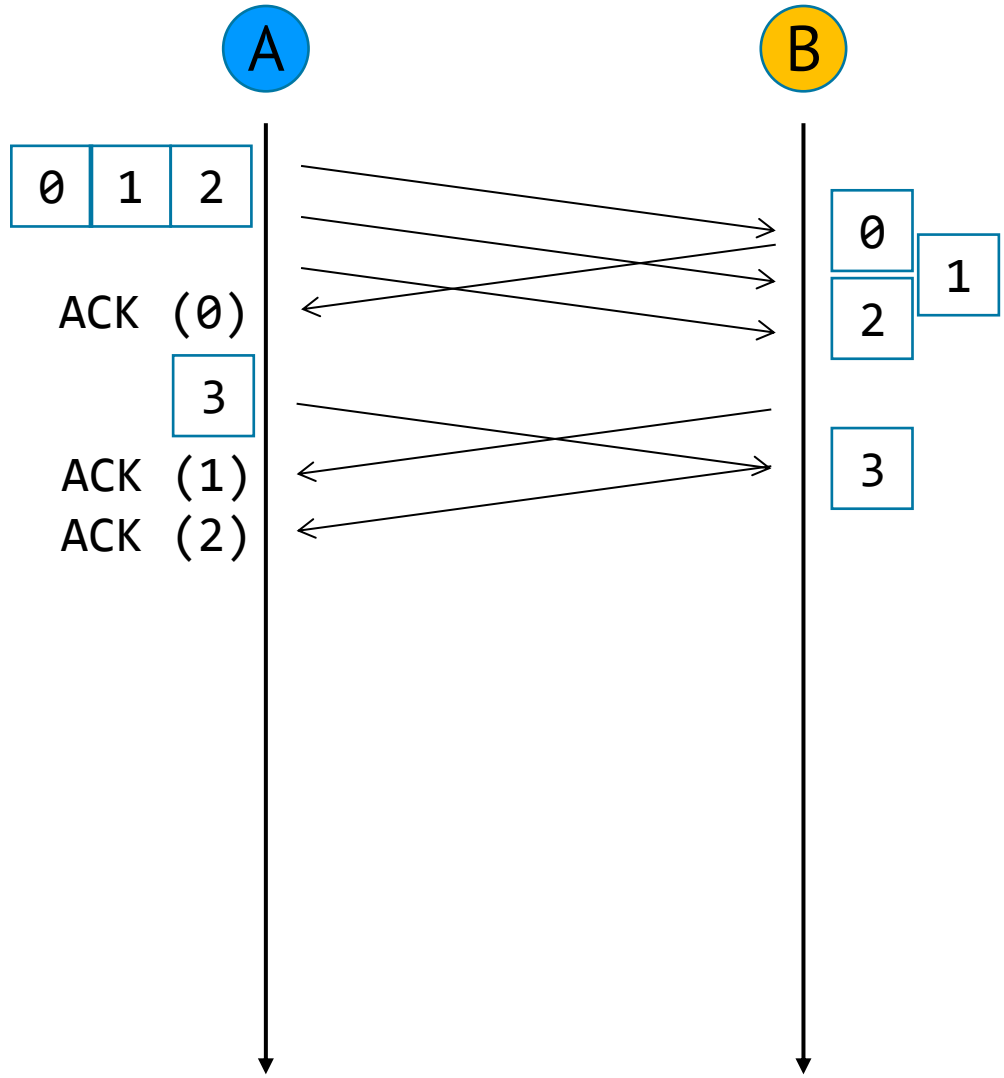




Nyugtázási módszerek

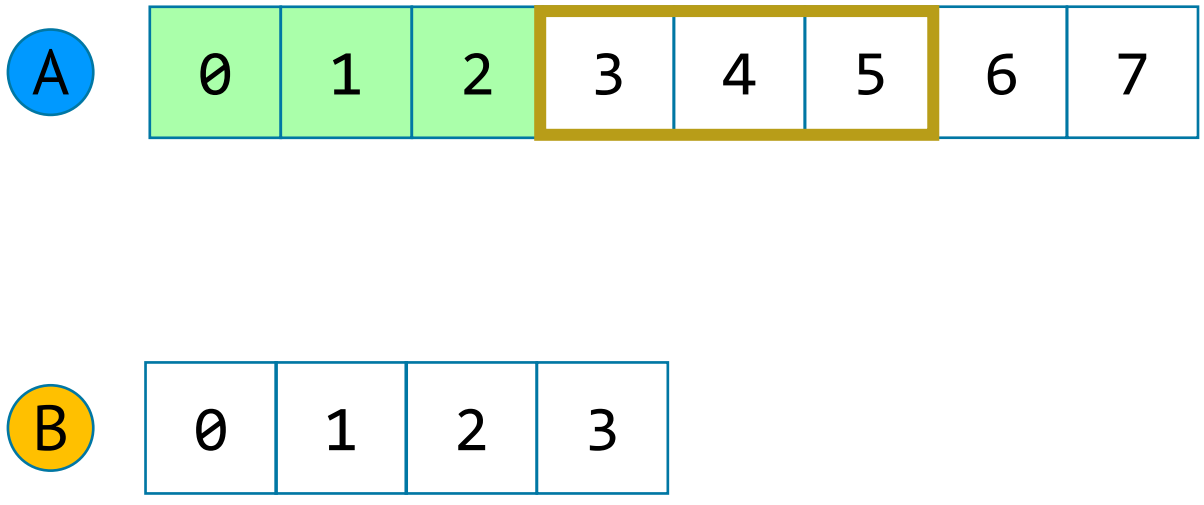
Sliding window protocol

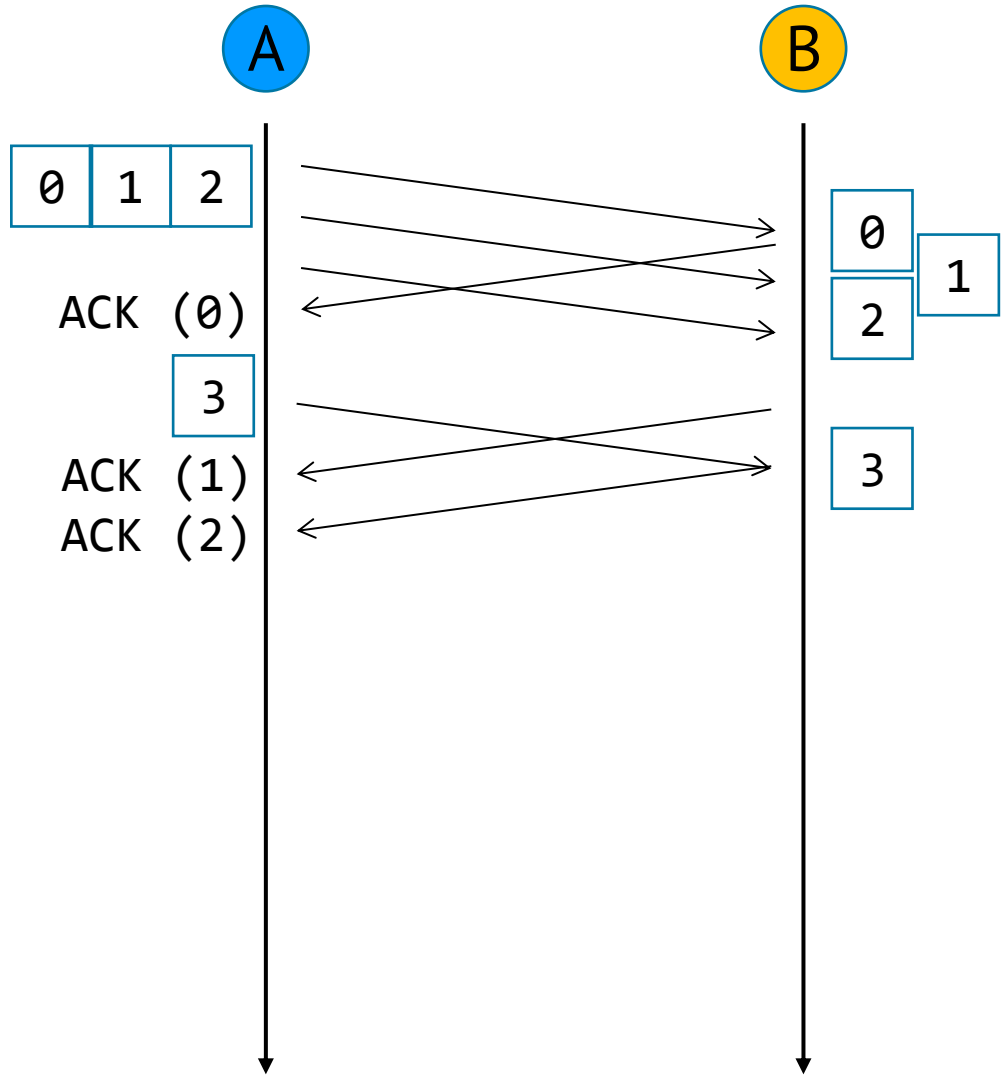




Nyugtázási módszerek

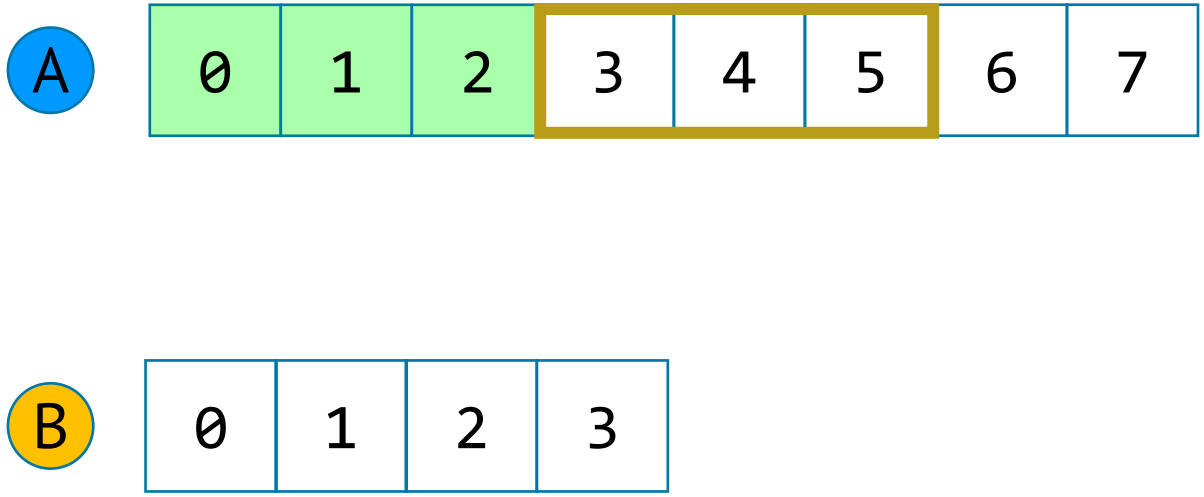
Sliding window protocol

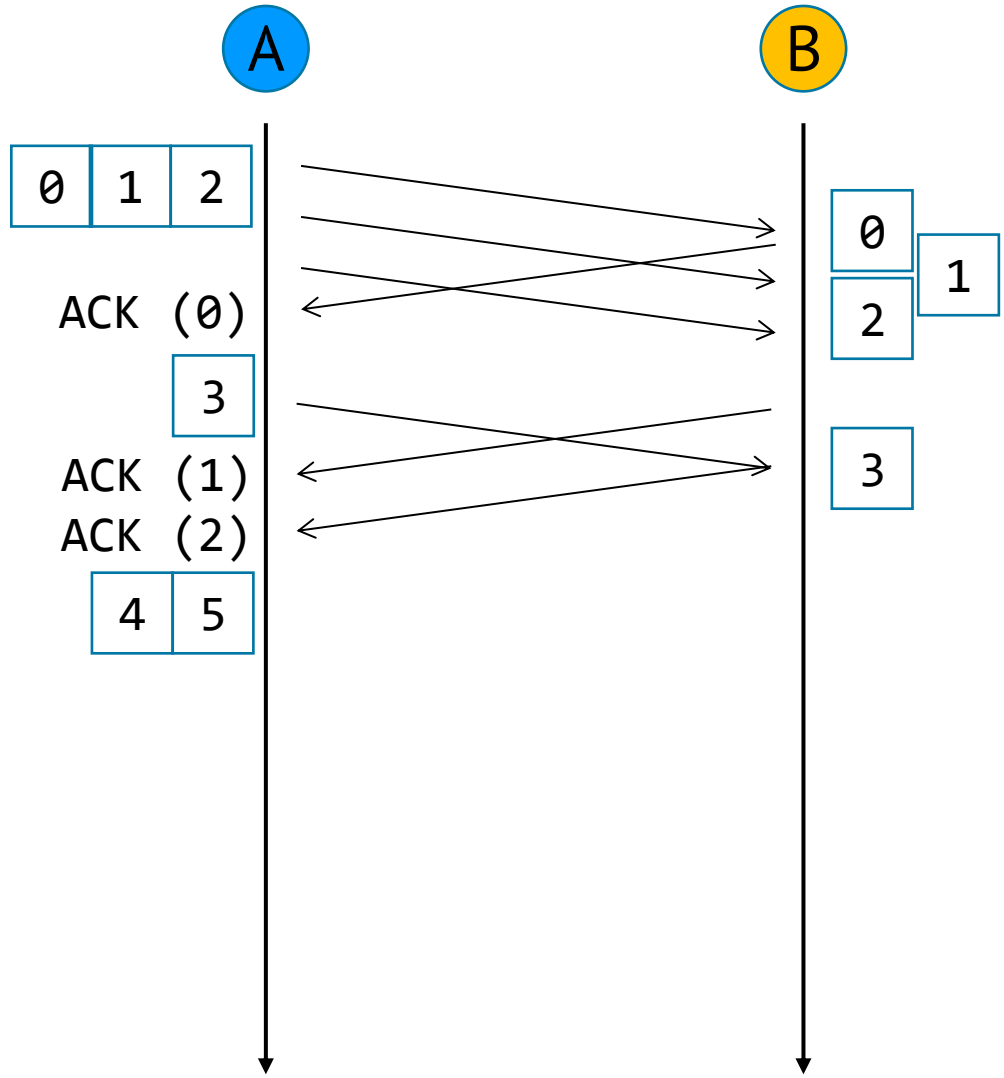




Nyugtázási módszerek

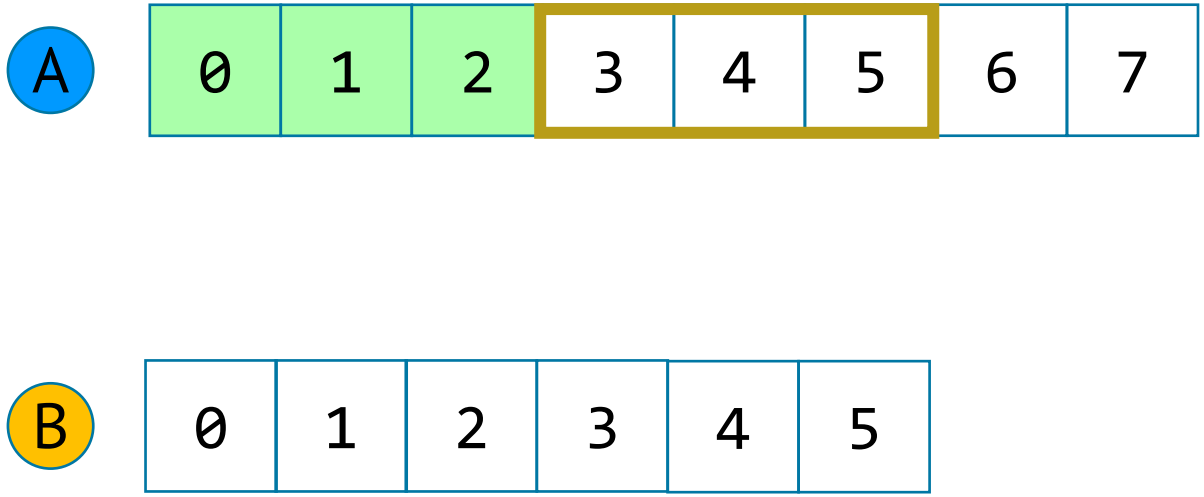
Sliding window protocol

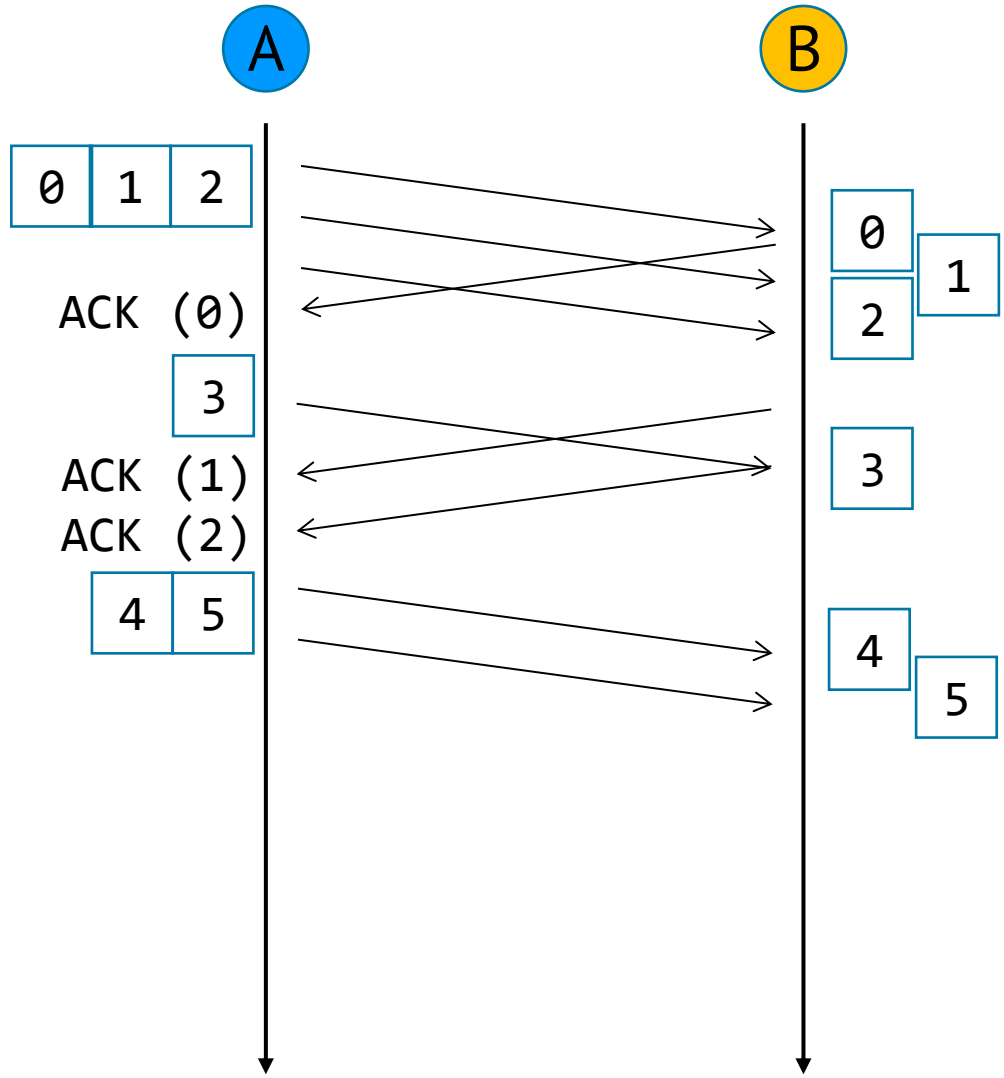




Nyugtázási módszerek

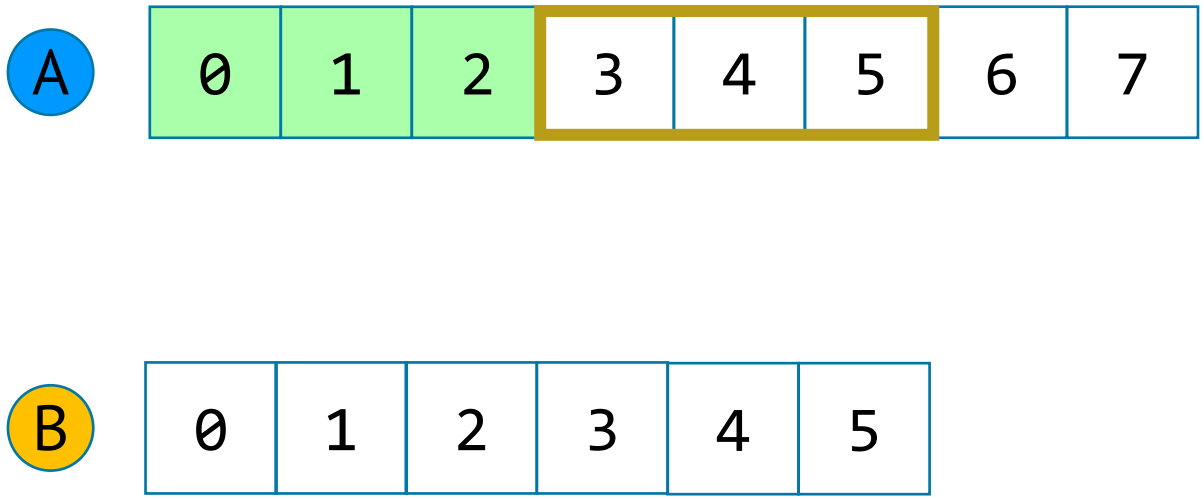
Sliding window protocol

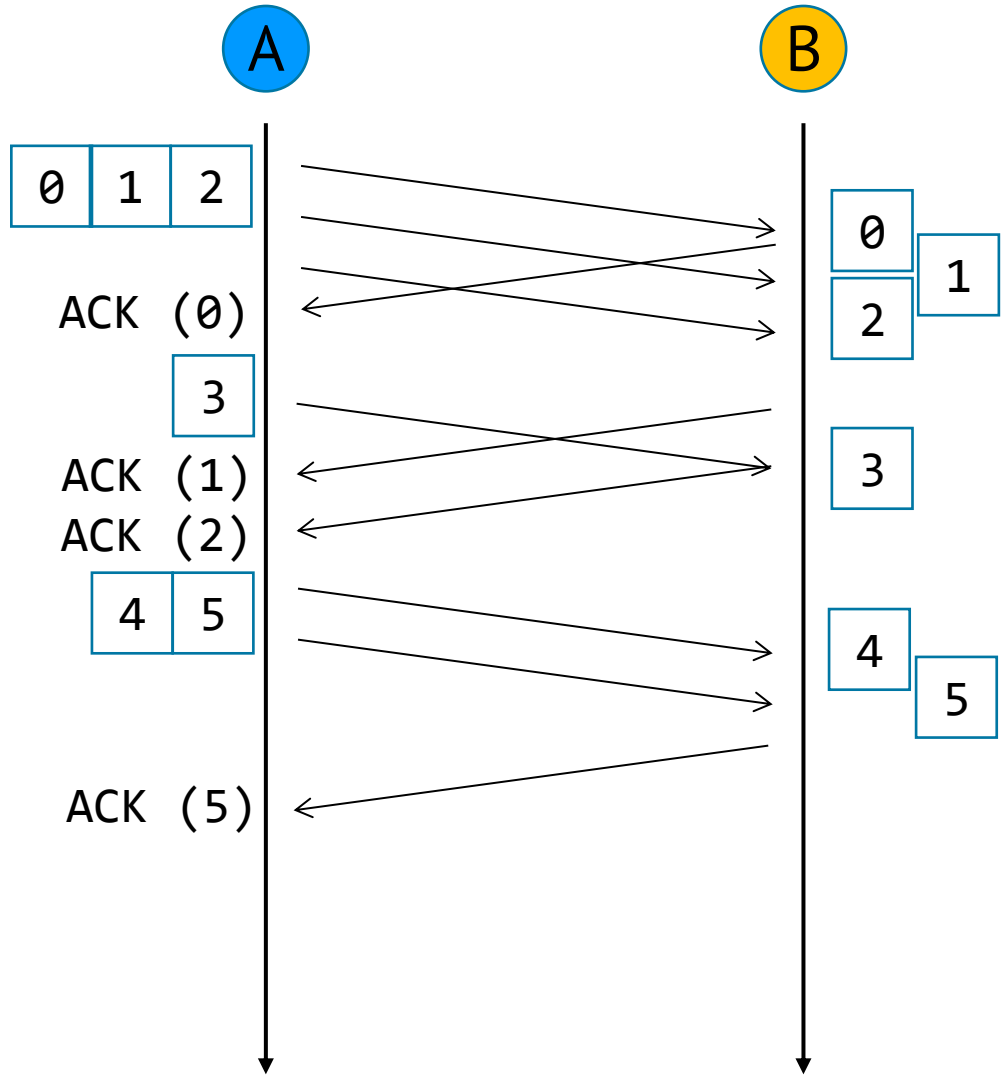




Nyugtázási módszerek

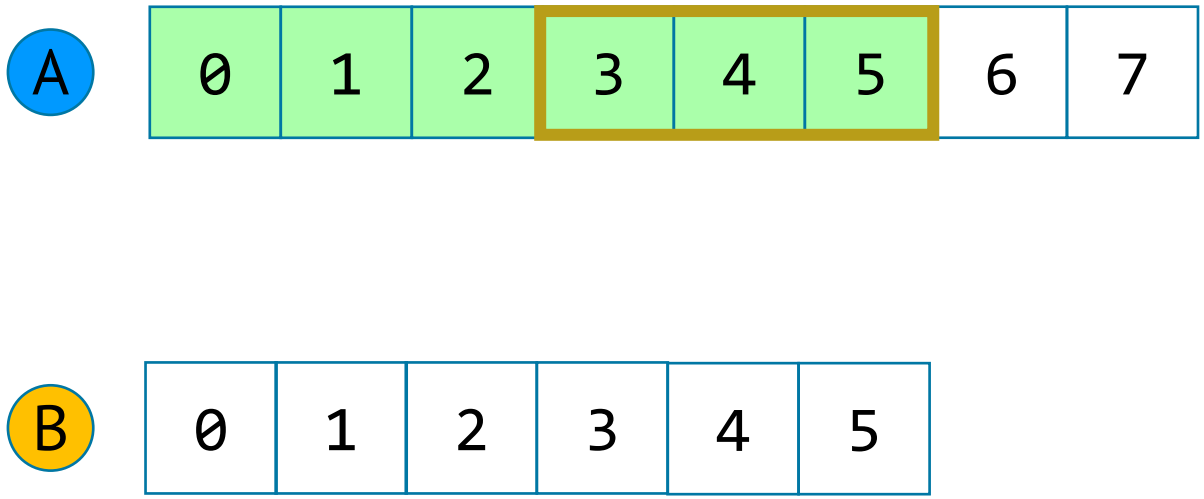
Sliding window protocol

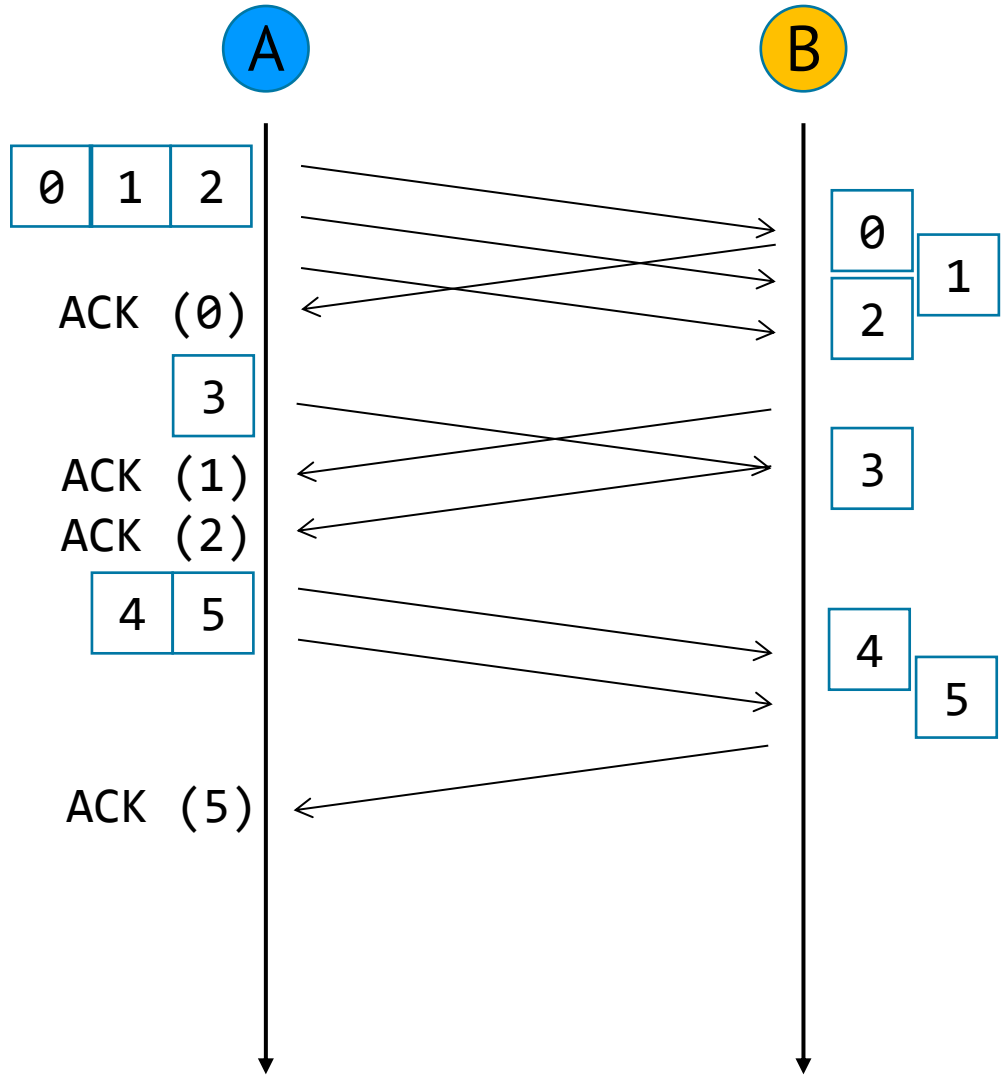




Nyugtázási módszerek

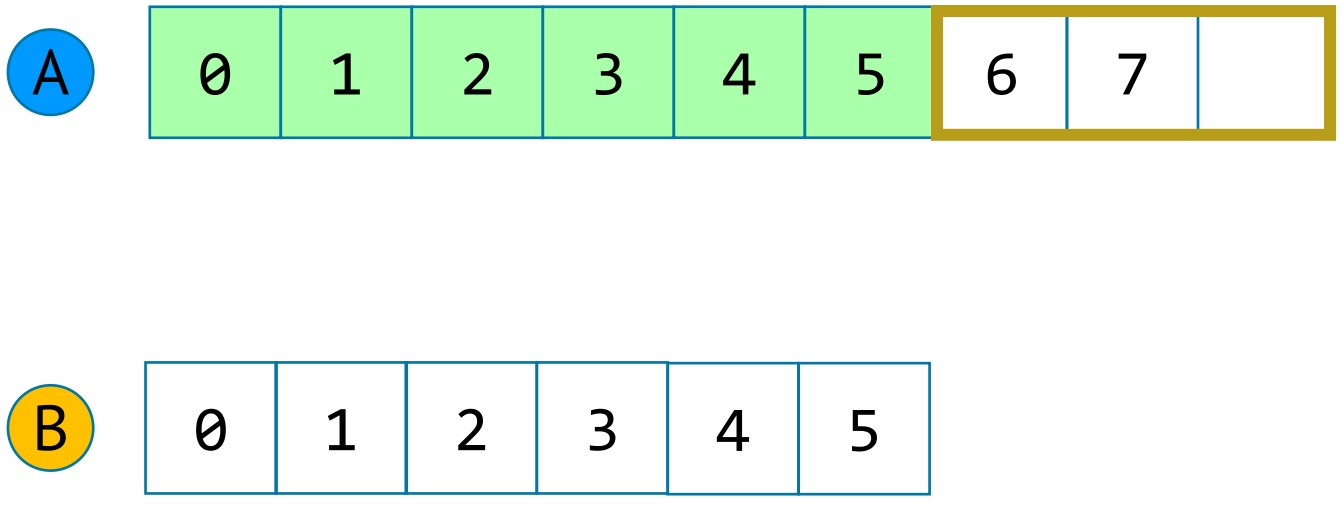
Sliding window protocol

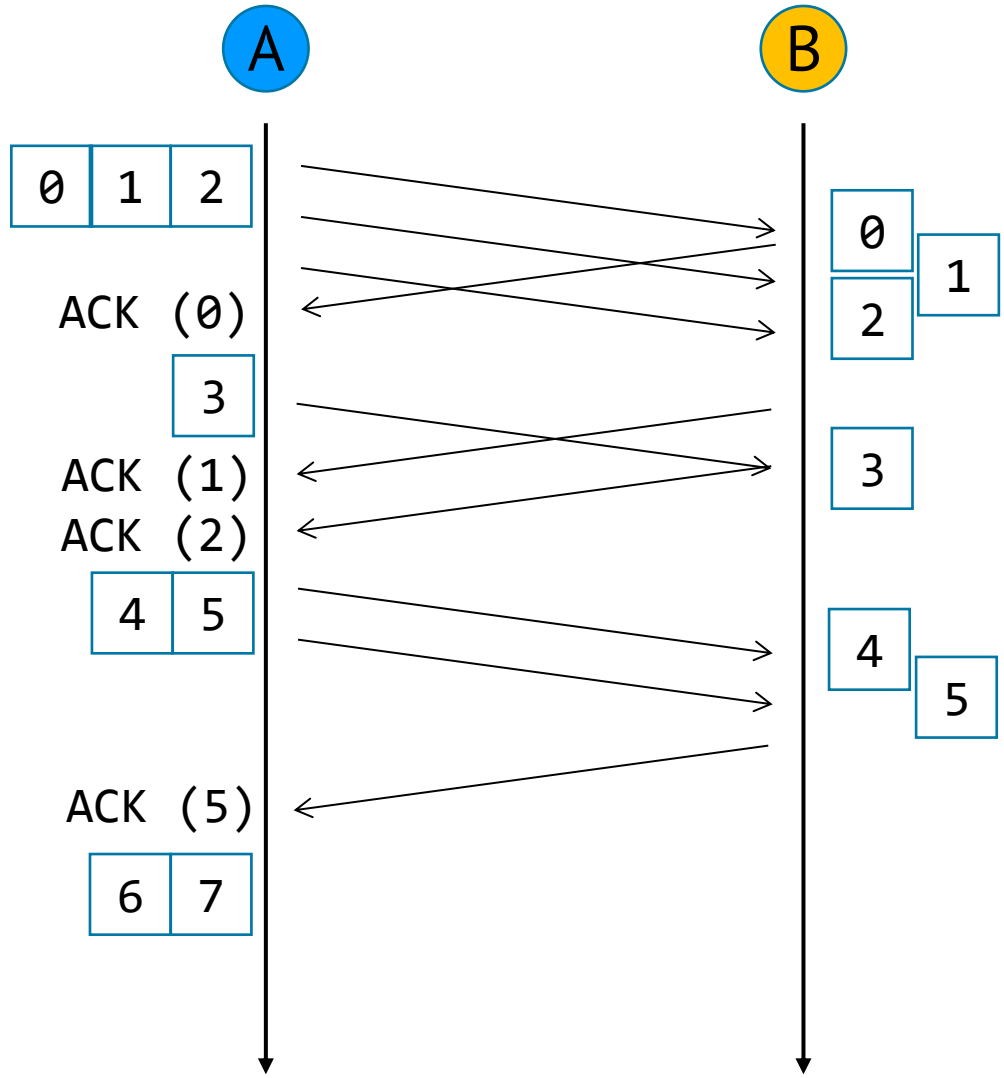




Nyugtázási módszerek

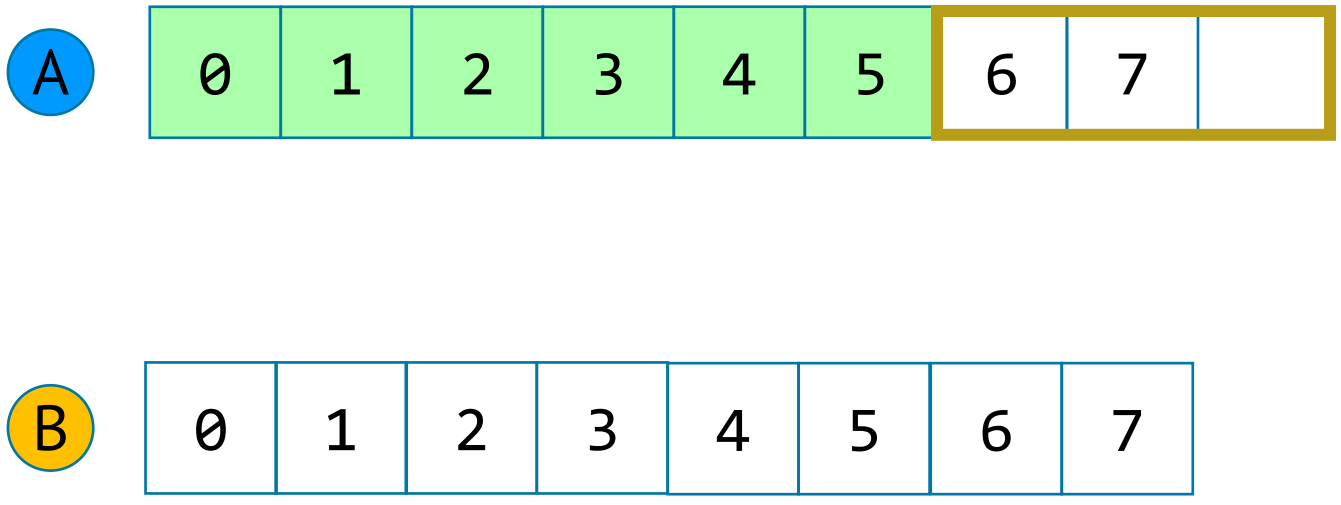
Sliding window protocol

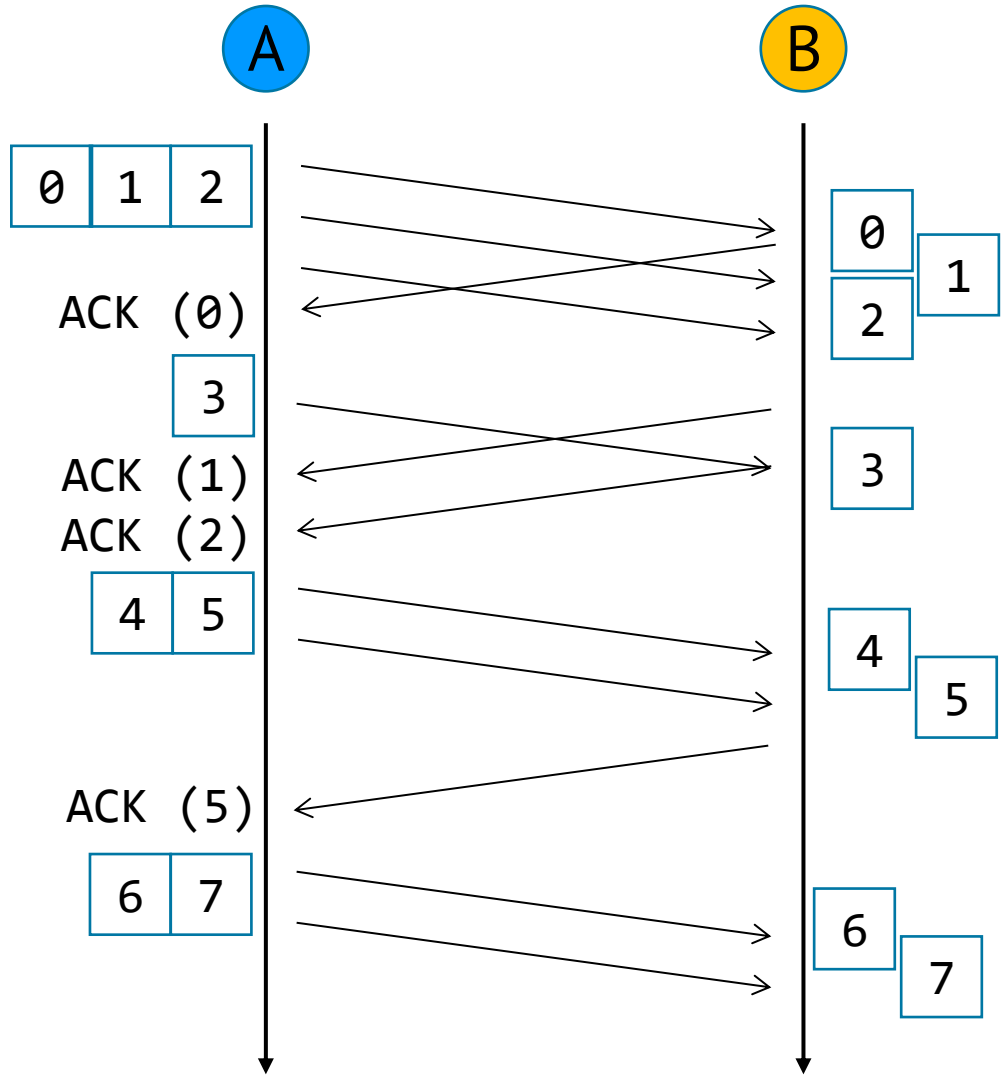




Nyugtázási módszerek

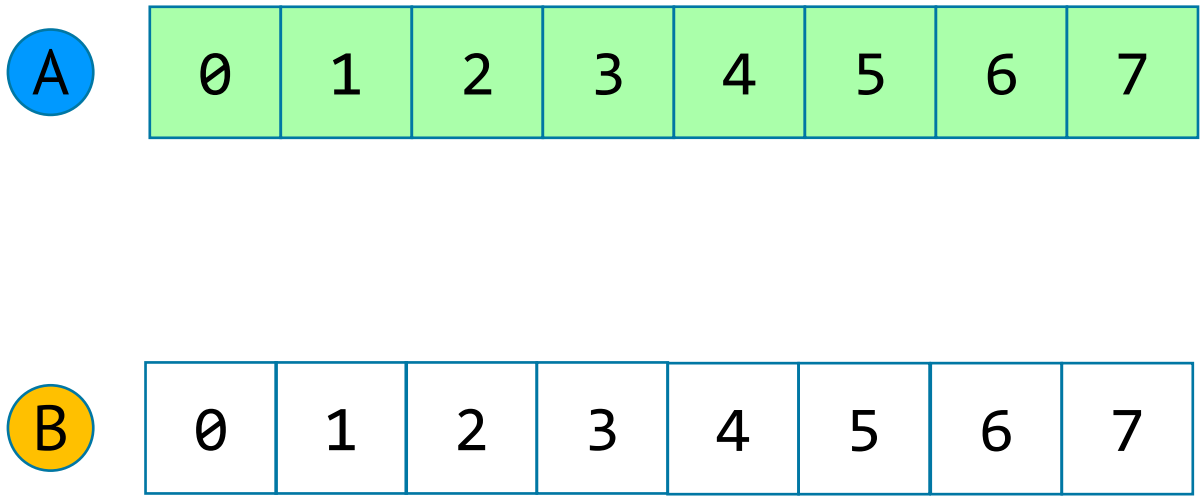
Sliding window protocol

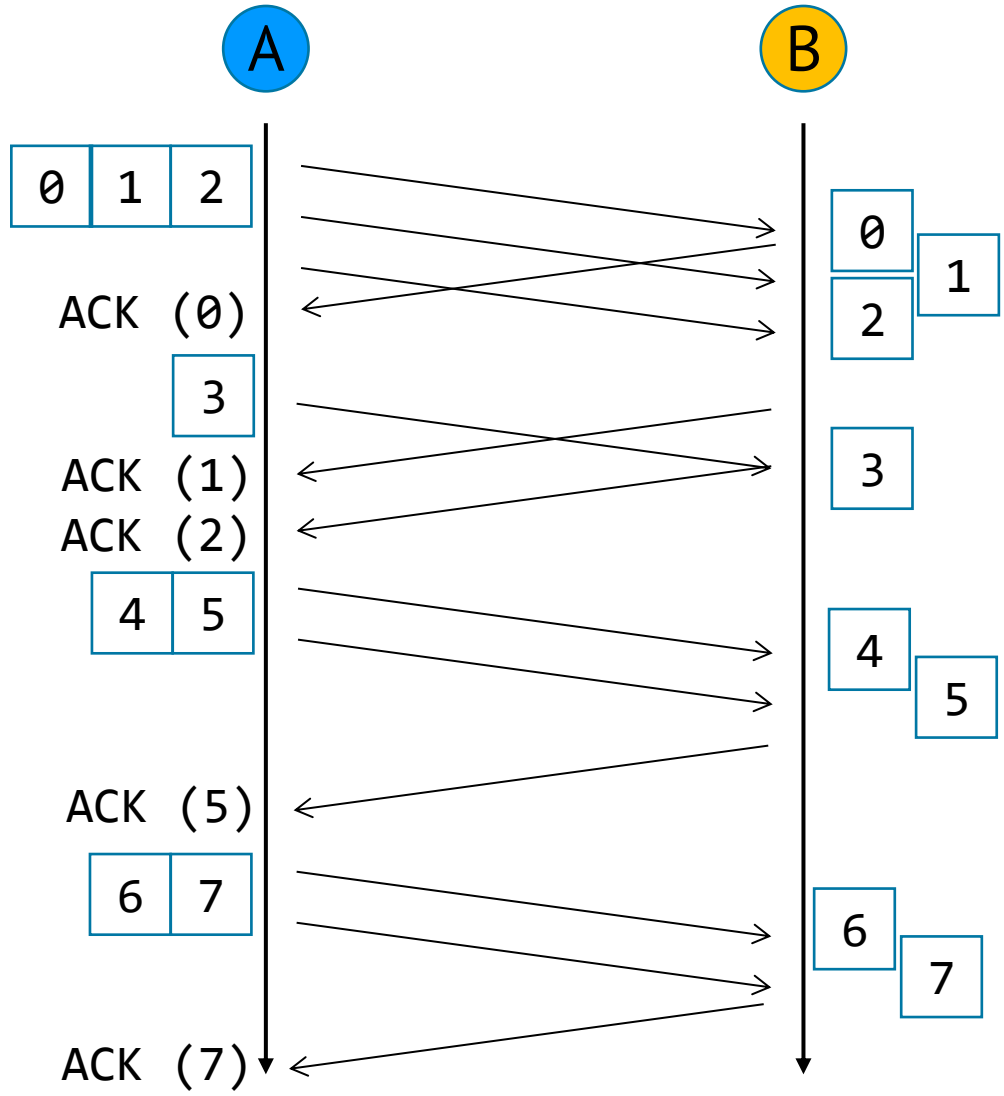




Nyugtázási módszerek

Sliding window protocol





Sliding window protocol megállapítások

Többet küld egyszerre, ezért gyorsabb.

A window size-tól függően akár folyamatosan is küldhet, jól kihasználja a sávszélességet.

Az window size az alkalmazás igényeitől függően dinamikusan változhat a kommunikáció során; pl. a fogadó csökkentheti/növelheti az ablak méretét.

Egy nyugta üzenet mindig az abban szereplő sequence numberig nyugtáz mindent, azaz az őt megelőző segmentek is nyugtázva lesznek általa. Ebből adódik, hogy nem feltétlenül kell minden segmentet egyesével nyugtázni.

Újraküldés

Az újraküldésnek két oka lehet:

- nem kaptam nyugtaüzenetet (timeout)
- nem jó sequence numberre kaptam nyugtaüzenetet (pl. ugyanazt a nyugtát kaptam meg többször is).

A saját oldalamon folyamatosan regisztrálom, hogy melyik segmentek lettek nyugtázva.

A stop-and-wait esetében egészen addig próbálkozom az utolsó segment elküldésével, amíg nyugtát nem kapok. A sliding window esetében újra elküldöm a window-ban lévő összes segmentet, ezt követően várom, hogy nyugtát kapjak (és eltolhassam az ablakot).

#06/3 – Összefoglalás

Folyamat Stop-and-wait protocol

Sliding window protocol

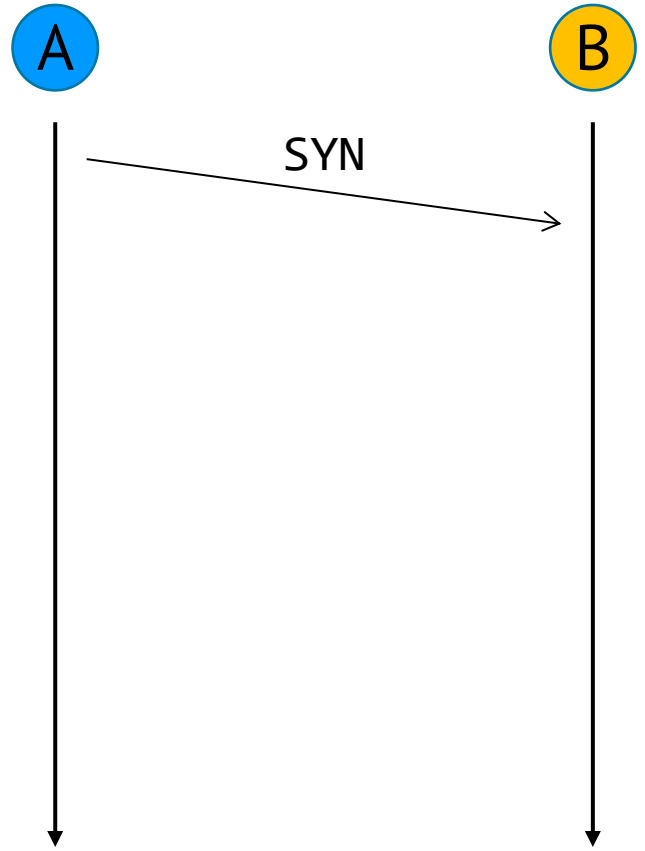
#06/4 – TCP kapcsolat építése és bontása

TCP kapcsolat létrehozása

Three-way handshake

1. Az A eszköz SYN csomagot küld B -nek.

„Szia másik eszköz, tudnál nyitni egy nekem egy TCP kapcsolatot?”



TCP kapcsolat létrehozása

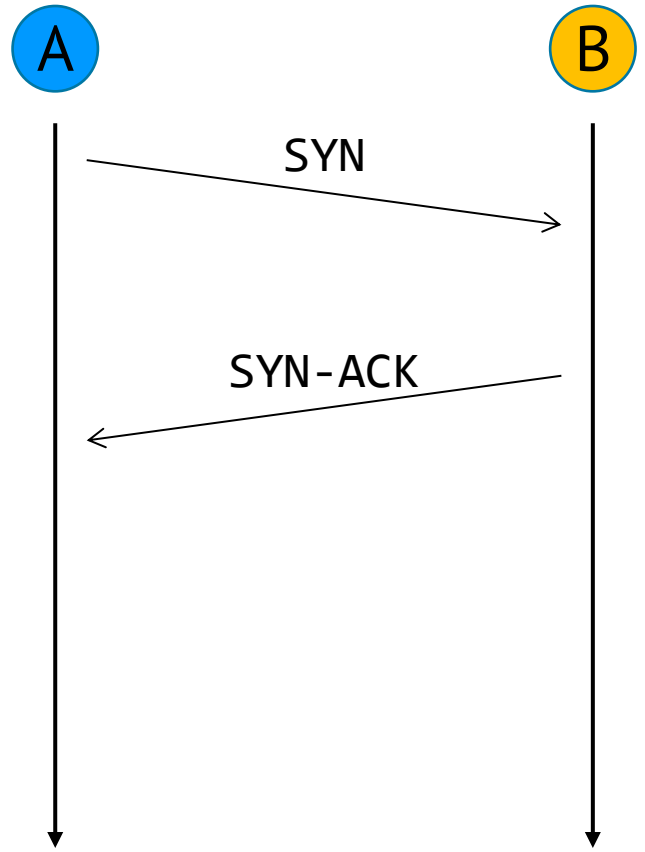
Three-way handshake

1. Az A eszköz SYN csomagot küld B -nek.

„Szia másik eszköz, tudnál nyitni egy nekem egy TCP kapcsolatot?”

2. A B eszköz erre egy SYN-ACK csomaggal válaszol.

„Szia, én tudok, és te is tudsz nyitni egyet nekem?”



TCP kapcsolat létrehozása

Three-way handshake

1. Az A eszköz SYN csomagot küld B -nek.

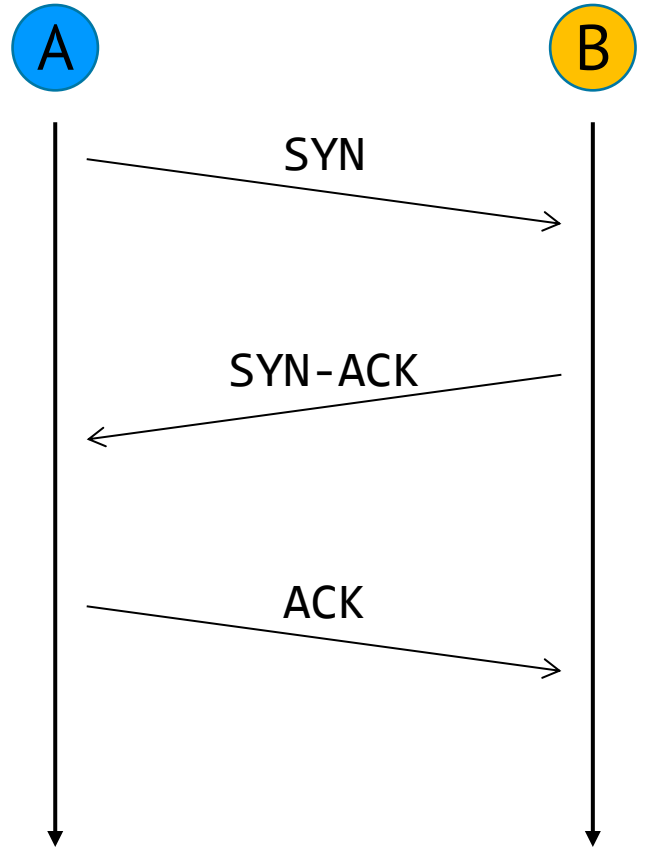
   „Szia másik eszköz, tudnál nyitni egy nekem egy TCP kapcsolatot?”

2. A B eszköz erre egy SYN-ACK csomaggal válaszol.

   „Szia, én tudok, és te is tudsz nyitni egyet nekem?”

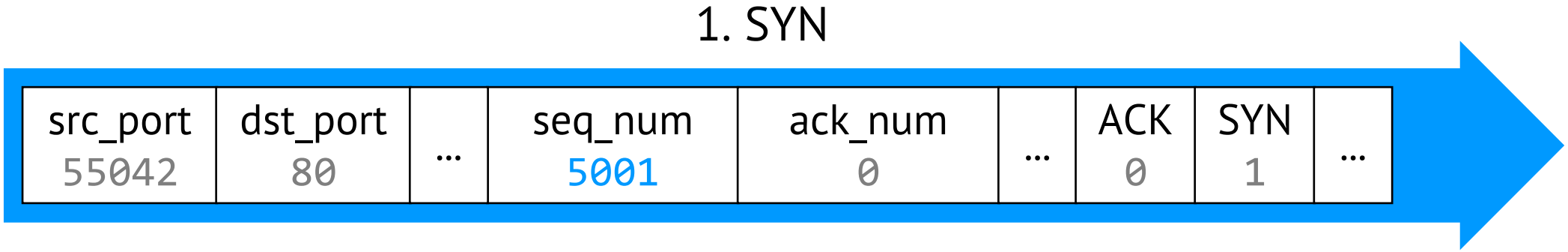
3. Az A eszköz is válaszol egy ACK-t.

   „Igen, én is nyitottam.”



TCP kapcsolat létrehozása

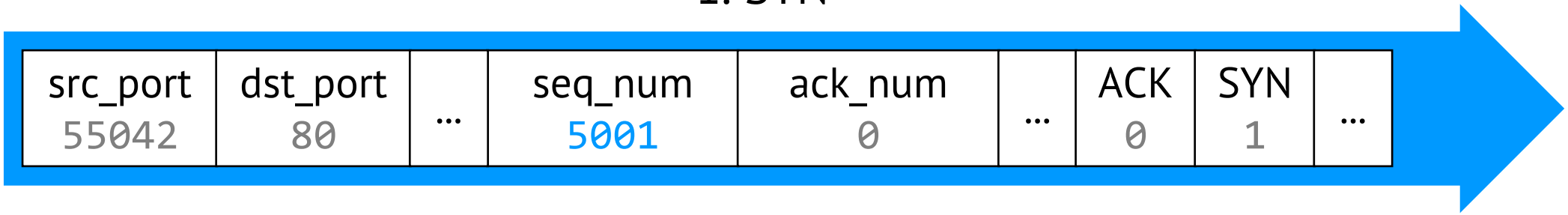
Three-way handshake



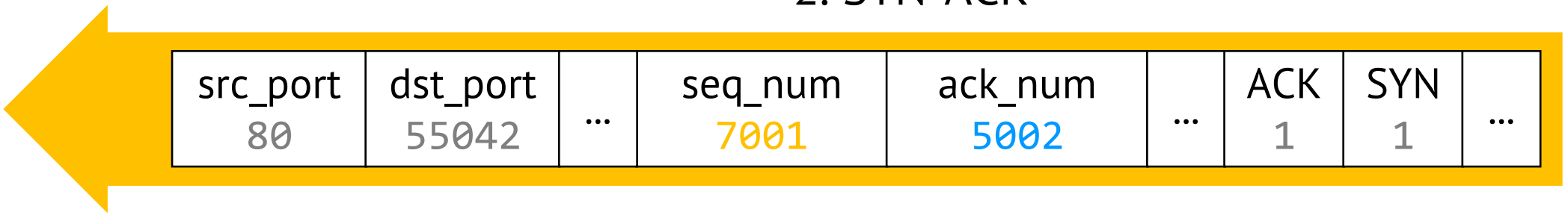
TCP kapcsolat létrehozása

Three-way handshake

1. SYN

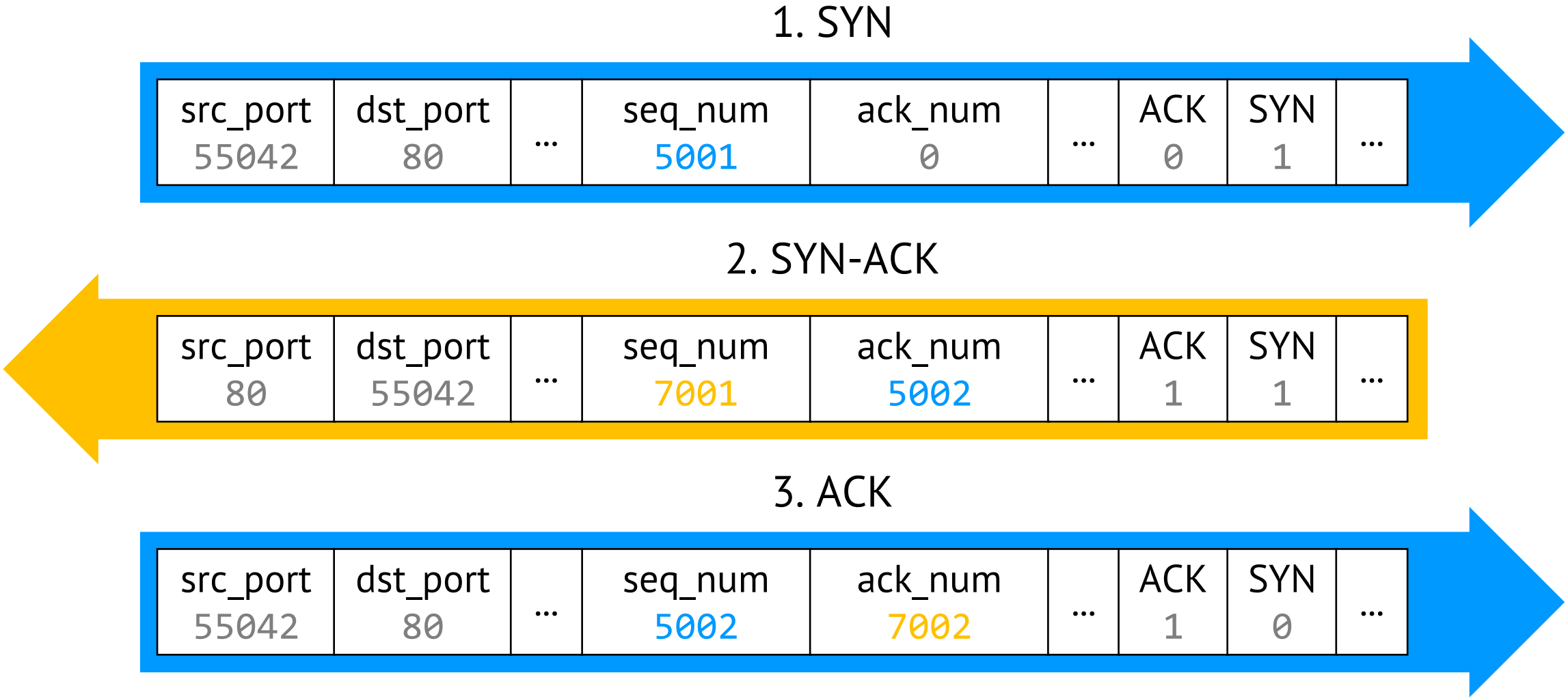


2. SYN-ACK



TCP kapcsolat létrehozása

Three-way handshake



TCP kapcsolat létrehozása

A kapcsolat felépítése a kezdeményező oldalán ún. active-open, a hívott fél pedig egy ún. passive-open-t hajt végre.

Ha a kapcsolat felépítése nem sikerült, akkor a kezdeményező egy bizonyos idő elteltével újra próbálkozik.

Ha többedszerre sem sikerül a kapcsolat felépítése, akkor a program értesíti az alkalmazást erről a hibáról.

TCP SYN flood támadás

A támadás lényege, hogy a támadott IP címre SYN csomagok tömegét küldjük.

A forrás IP címek hamisak, véletlenszerűen generáltak.

A támadott állomás SYN-ACK-t válaszol, és vár a kitalált állomástól jövő ACK-ra, ami sosem fog megérkezni.

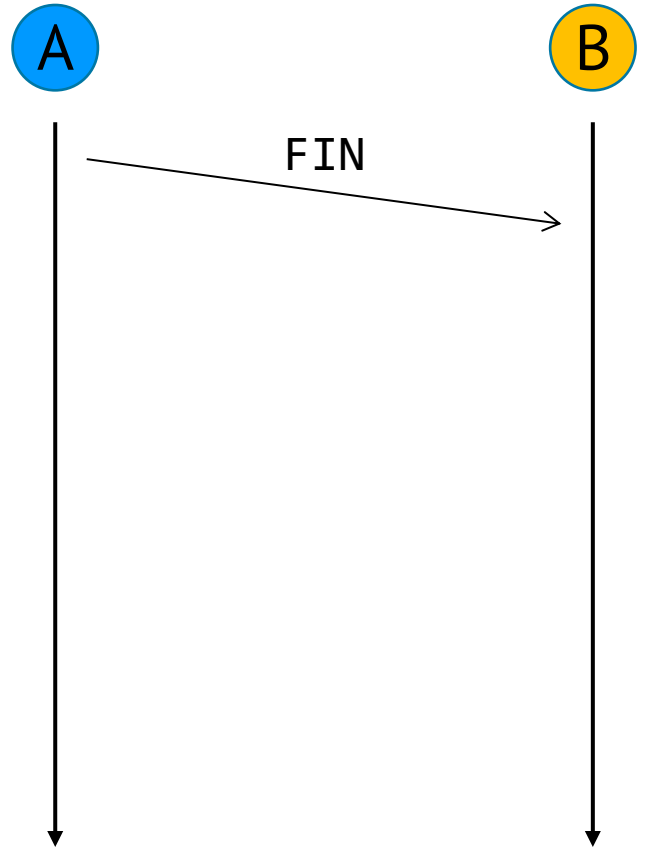
Végeredményben az állomás leterhelődik (DoS).

TCP kapcsolat bontása

Kapcsolat rendes lezárása

1. Az A eszköz FIN csomagot küld B -nek.

„Szia másik eszköz, részemről végeztünk.”



TCP kapcsolat bontása

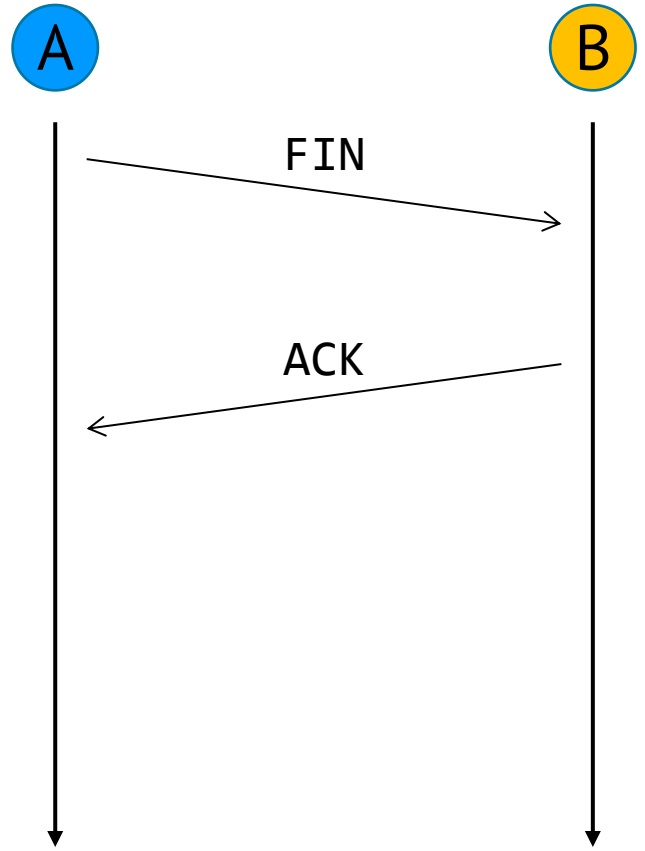
Kapcsolat rendes lezárása

1. Az A eszköz FIN csomagot küld B -nek.

   „Szia másik eszköz, részemről végeztünk.”

2. A B eszköz nyugtázza ezt egy ACK-val.

   „Szia, oké.”



TCP kapcsolat bontása

Kapcsolat rendes lezárása

1. Az A eszköz FIN csomagot küld B -nek.

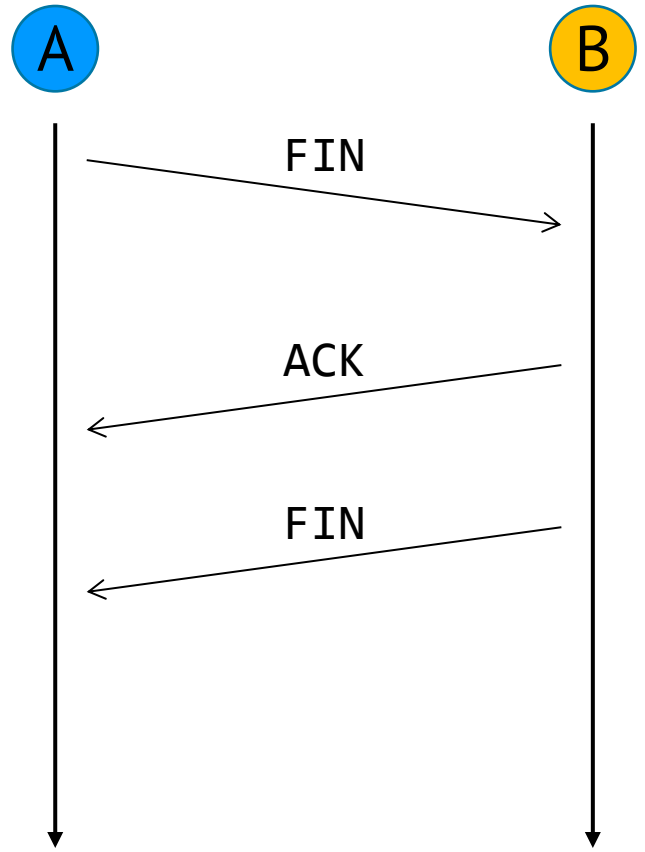
   „Szia másik eszköz, részemről végeztünk.”

2. A B eszköz nyugtázza ezt egy ACK-val.

   „Szia, oké.”

3. A B eszköz is küld egy FIN csomagot az A -nak.

   „Szia másik eszköz, részemről is végeztünk.”



TCP kapcsolat bontása

Kapcsolat rendes lezárása

1. Az A eszköz FIN csomagot küld B -nek.

   „Szia másik eszköz, részemről végeztünk.”

2. A B eszköz nyugtázza ezt egy ACK-val.

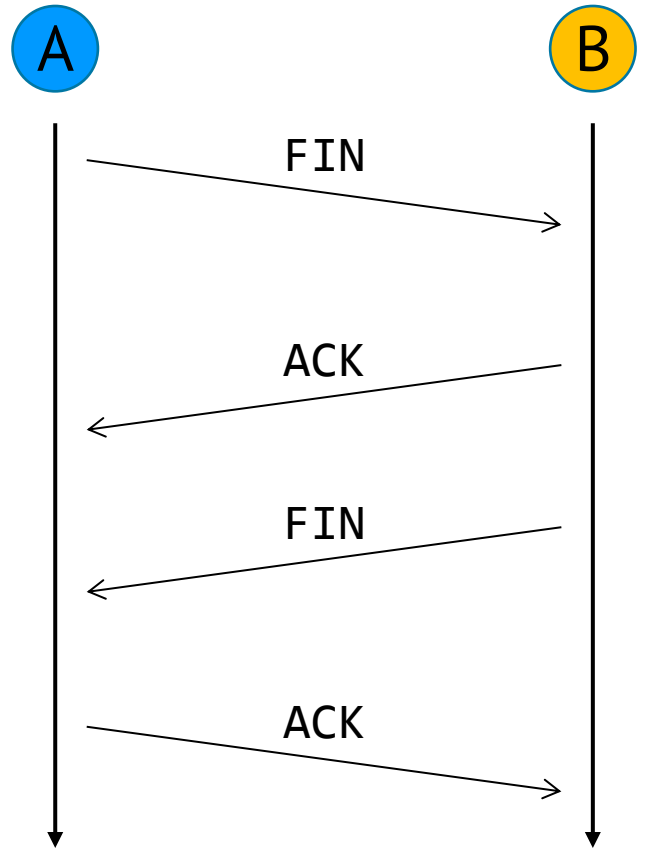
   „Szia, oké.”

3. A B eszköz is küld egy FIN csomagot az A -nak.

   „Szia másik eszköz, részemről is végeztünk.”

4. Az A eszköz is válaszol egy ACK-t.

   „Szia, oké.”



TCP kapcsolat bontása

A kapcsolat bontása a kezdeményező oldalán ún. active-close, a másik fél pedig egy ún. passive-close-t hajt végre.

Ha a kapcsolatot csak az egyik fél bontja, akkor elvben létezhet ún. half-open állapot is, de ezt a gyakorlatban nagyon ritkán használjuk.

#06/4 – Összefoglalás

Elvek

Three-way handshake

Mit szinkronizál a SYN üzenet?

Kapcsolat rendes bontása

#06/5 – TCP állapotok

TCP állapotok

A TCP kapcsolat lehetséges állapotai a következők:

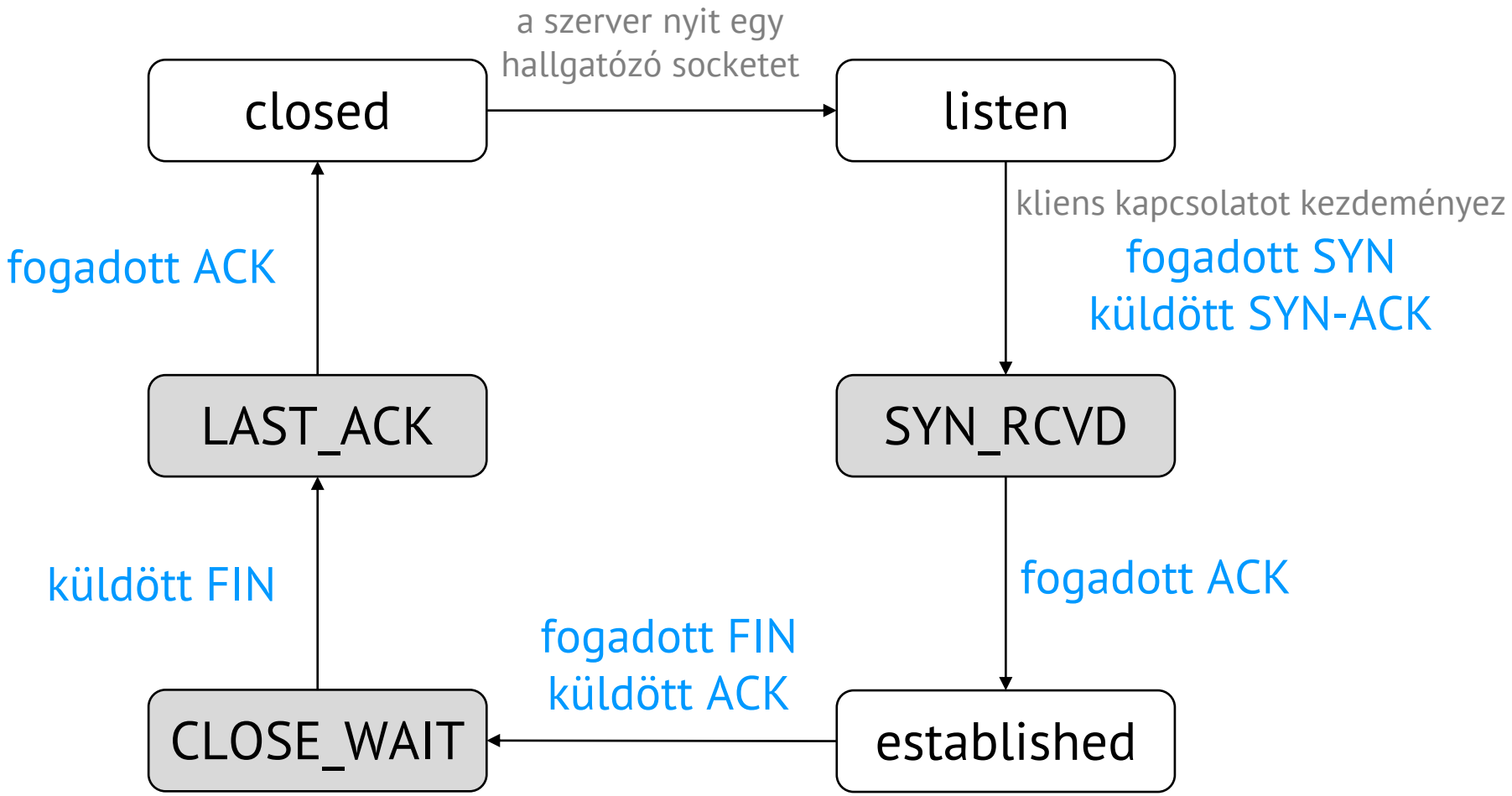
- closed — zárt állapot, nincs kapcsolat
- listen — passzív socket open utáni állapot, figyel, vár a kapcsolatra.
- established — felépült kapcsolat, lehetséges a kommunikáció

TCP állapotok

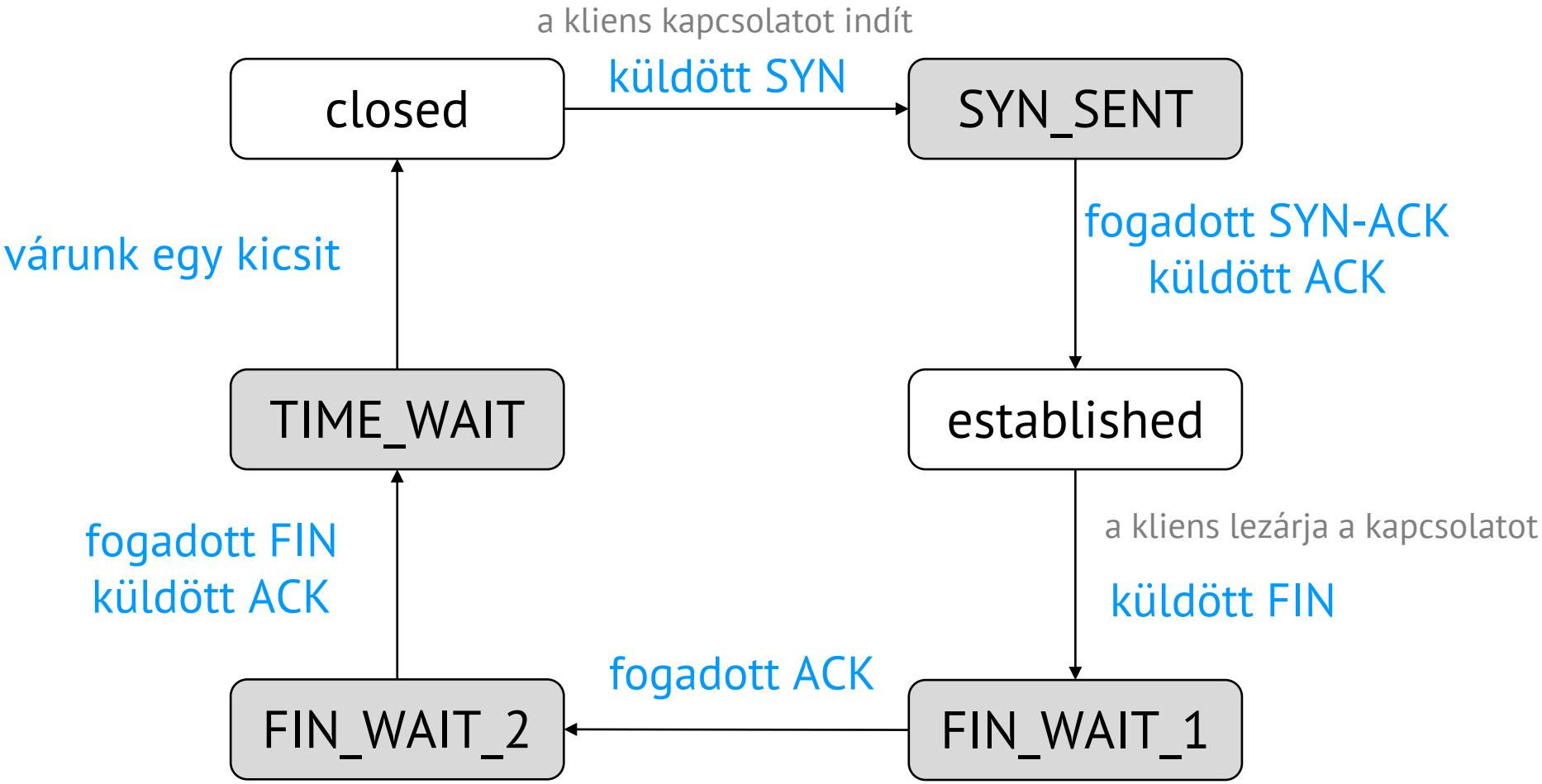
Lehetséges köztes állapotok:

- SYN_SENT elküldött SYN üzenet (aktív kapcsolatépítés)
- SYN_RCVD fogadott SYN üzenet (passzív kapcsolatépítés)
- FIN_WAIT_1 elküldött FIN üzenet (aktív kapcsolatbontás)
- FIN_WAIT_2 fogadott ACK az előbb küldött FIN-re
- TIME_WAIT várunk még egy kicsit, miután elbontottuk a kapcsolatot
- CLOSE_WAIT fogadott FIN üzenet (passzív kapcsolatbontás)
- LAST_ACK az általunk küldött FIN-re várjuk az ACK-t

TCP állapotábra– szerver (passive open & close)



TCP állapotábra – kliens (active open & close)



A TIME_WAIT állapot

Miért kell várni egy kicsit?

A TIME_WAIT az aktív lezárást végző oldal utolsó állapota. Emlékezzünk, hogy az utolsó mozzanat, hogy az aktív oldal elküldi az ACK-t.

Lehet, hogy ez az ACK elveszik. Ekkor a másik fél újra fogja küldeni a FIN-t. Készen kell álnunk arra, hogy „a semmiből” egyszercsak jön egy ilyen FIN, ami még ehhez a kapcsolathoz tartozik.

Megoldás: várunk 2×MSL időt.

MSL: Maximum Segment Lifetime, becsült érték, ennél tovább nem lehet a hálózaton egy TCP szegmens (ennyi idő alatt tuti célba ér).

RESET

Lehetséges, hogy olyan csomag érkezik, amit nem tartunk szabályosnak.

Pl. olyan portra érkezik TCP kérés, amit nem figyelünk.

Ilyenkor RESET-et küldünk vissza: „valami gáz van, kezd az elejéről a működést”

Használható a kapcsolat erőszakos bontására is; ekkor nincs garancia arra, hogy minden elküldött szegmenst megkapott a fogadó oldal.

TCP RESET támadás

A két eszköz közötti kommunikációba egy gonosz harmadik eszköz egy RESET csomagot csempészhet.

Ehhez a gonosz eszköznek az alábbiakat kell tudnia:

- forrás és cél IP cím
- forrás és cél port
- sequence number

A sequence number esetén nem kell tudni a pontos értéket, elég csak egy olyan szám, ami pont a window-ban van (nagy window méret esetén ez könnyű).

Ezzel erővel elbontható a kapcsolat a két fél közt, meggátolható a kettejük kommunikációja (DoS).

Half-open kapcsolatok

Előfordulhat, hogy a TCP kapcsolatban lévő alkalmazások közül az egyik újraindul. A másik fél erről nem tud, tehát élőnek hiszi a kapcsolatot.

Amikor a másik fél üzenetet küld, akkor az az újraindult alkalmazás oldalán ez meglepetés lesz, hiszen az újraindulás miatt minden az alapállapotba került vissza.

A meglepett alkalmazás ekkor RESET választ küld.

Szimultán open&close

Néha van olyan, hogy két alkalmazás egyszerre küld egymásnak SYN-t (egyszerre akarják mindketten az active-open-t csinálni).

Ilyenkor mindkettő sima ACK-val válaszol, és csak egy kapcsolat épül fel.

Az is lehetséges, hogy a zárás esik egybe; ekkor a FIN-re nem ACK hanem FIN jön.

A FIN-t minden esetben meg kell válaszolni egy ACK-val, tehát küldünk egy ACK-t, de várjuk a saját FIN-ünk ACK-ját is.

Mindkét résztvevő TIME_WAIT állapotba kerül.

#06/5 – Összefoglalás

Eljárások lehetséges állapotok

passive és active open

#06/6 – Flow control

Flow control

Hogyan kell adni, ha ki akarjuk használni a sávszélességet?

- mikor küldjünk el egy szegmenst?
- mikor nyugtázzuk le a kapott szegmenst?
- mennyi szegmenst küldjünk egyszerre?
- mit csináljunk, ha észrevesszük, hogy elvesznek a szegmensek?
- milyen stratégiát válasszunk, ha gyorsan kell kevés infót oda-vissza küldeni?
- milyen stratégiát válasszunk, ha sok infót kell egyik irányba küldeni?
- milyen stratégiát válasszunk, ha minden hálózaton átvitt byte után fizetni kell?

Fontos fogalmak

Értenünk kell, hogy mit jelent...

- buffer (puffer)
- szegmens
- sliding window
- sávszélesség
- RTT
- tripla ACK
- torlódás

Fontos fogalmak

Buffer (puffer)

A hálózati eszközök azon memória jellegű része, ahol

- a beérkezett, de még fel nem dolgozott, vagy
- a már előállított, de még ki nem küldött

üzeneteket, csomagokat, frame-eket átmenetileg tároljuk.

A buffer véges méretű, gondot jelent, ha gyorsabban telik mint ahogy ürül.

Ha egy elem nem fér bele a bufferbe, akkor többféle taktikát követhetünk:

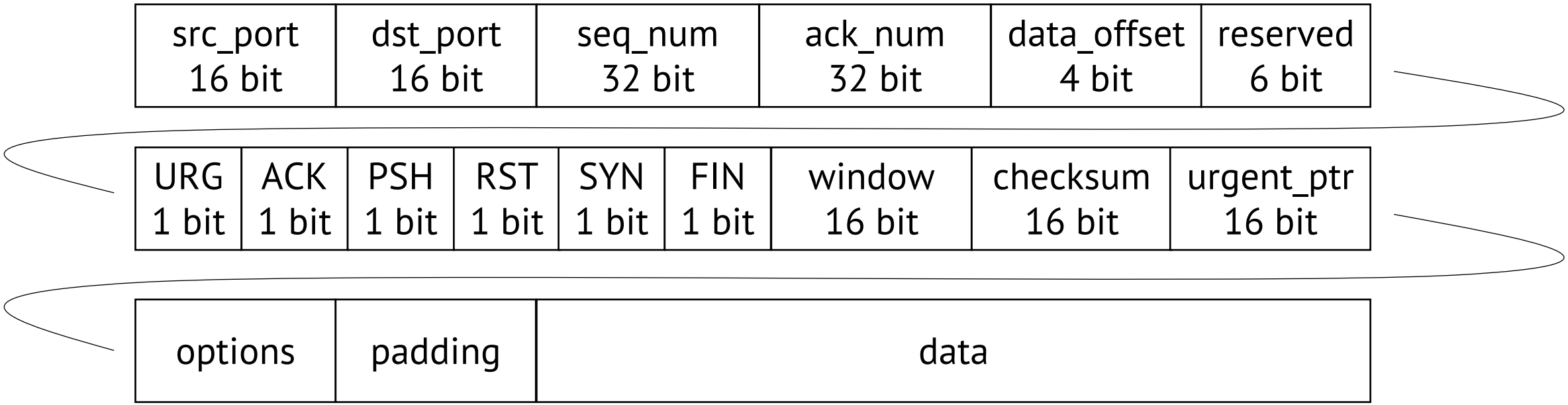
- eldobjuk a bele nem férő elemet,
- eldobunk egy másik elemet,
- teljes pánik…

Fontos fogalmak

Szegmens

A TCP réteg által darabokra bontott információ (byte-folyam) egy része, amit fejléccel látunk el, és átküldünk a hálózaton. A fejléc minimum 20 byte méretű.

A TCP minden szegmensre nyugtát vár.



Fontos fogalmak

Sliding window protocol

Az ablakot három paraméter határozza meg: az ablak mérete, a legutolsó nyugtázott üzenet sorszáma (ez lesz az ablak bal oldali vége), és az, hogy hány üzenetet küldtünk el az ablakon belül. Az ablakba eső üzeneteket el szabad küldeni, és ha megjön az ACK, akkor az ablak bal végét jobbra szabad mozdítani.

Fontos fogalmak

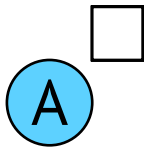
Sávszélesség (bandwidth)

Az egyik irányba adott időegység alatt átvihető adatmennyiség mértéke.

Mértékegysége adatmennyiség/idő jellegű (pl. kbps, Mbps, Gbps)

Mbps = megabit per másodperc ≠ MBps = megabyte per másodperc

PLANG.jar
42 MB



Fontos fogalmak

Sávszélesség (bandwidth)

Az egyik irányba adott időegység alatt átvihető adatmennyiség mértéke.

Mértékegysége adatmennyiség/idő jellegű (pl. kbps, Mbps, Gbps)

Mbps = megabit per másodperc ≠ MBps = megabyte per másodperc

PLANG.jar
42 MB

A

$$\frac{42\ \text{MB}}{3\ \text{s}} = 14\ \text{MBps} = 8 \cdot 14\ \text{Mbps} = 112\ \text{Mbps}$$

B

Fontos fogalmak

Sávszélesség (bandwidth)

Az egyik irányba adott időegység alatt átvihető adatmennyiség mértéke.

Mértékegysége adatmennyiség/idő jellegű (pl. kbps, Mbps, Gbps)

Mbps = megabit per másodperc ≠ MBps = megabyte per másodperc

Práter

Mixáth

$$\frac{4\ \text{TB}}{10\ \text{Gbps}} = 53\ \text{perc}$$

Práter

Mixáth

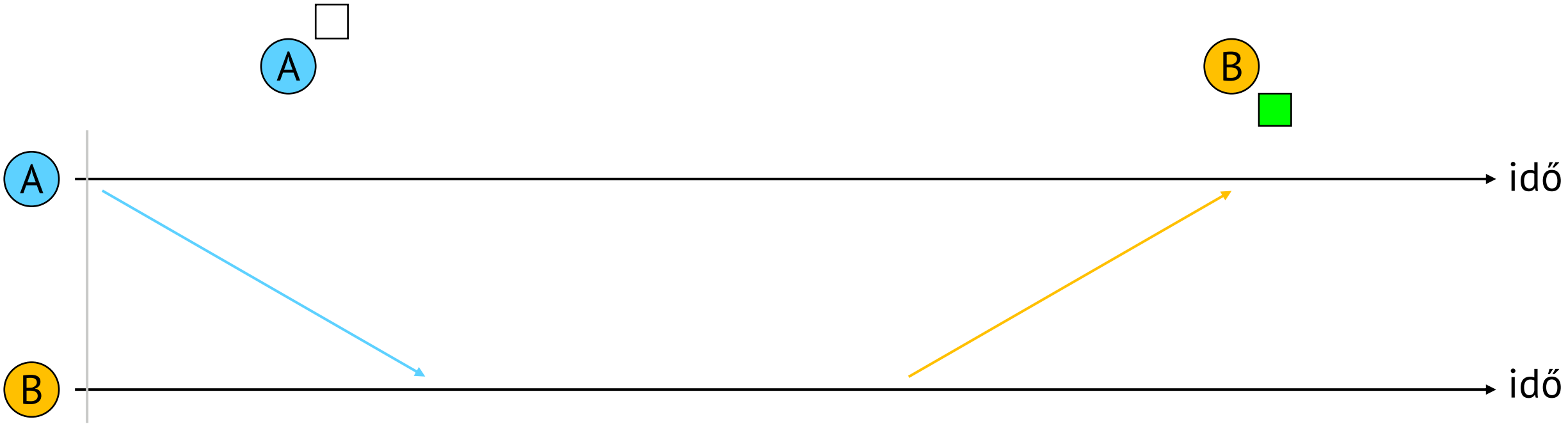
$$\frac{4\ \text{TB}}{10\ \text{perc}} = 53\ \text{Gbps}$$

Fontos fogalmak

RTT (round trip time – csomagfordulási idő)

A szegmens elküldése és az őt visszaigazoló ACK beérkezése közt eltelt idő.

Mértékegysége idő jellegű (pl. ms)



Fontos fogalmak

Tripla ACK

Ha a fogadó oldal kimaradó szegmenst észlel, akkor ezt az előzőleg visszaigazolt sequence numberre adott többszöri ACK-val tudatja. A kimaradás oka lehet a szegmensek sorrendjének felcserélődése, de adatvesztés is.

A küldő oldal három darab azonos sequence numberre kiadott ACK után kezdhet gyanakodni, hogy valami hiba történt az átvitel során.

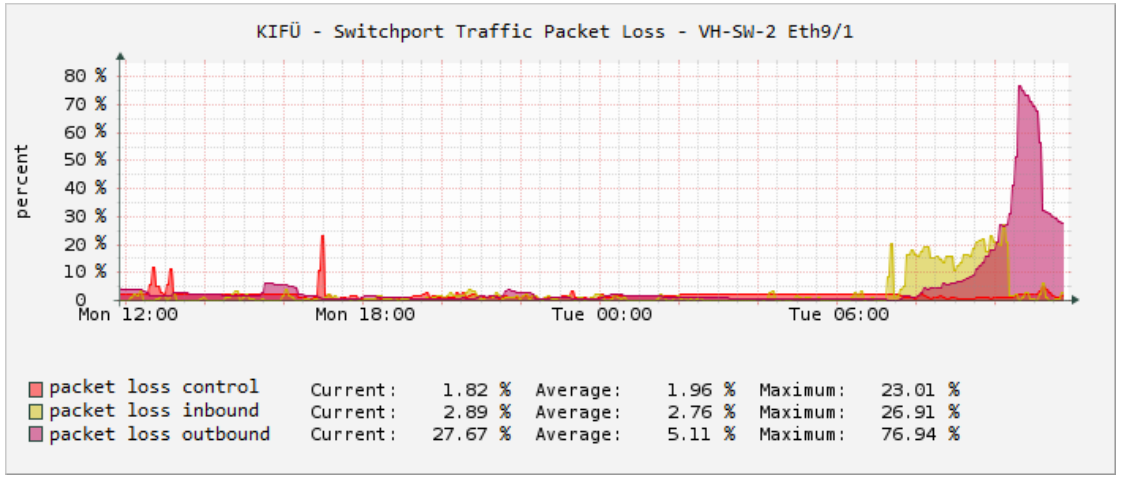
Fontos fogalmak

Torlódás (congestion)

Ha túl sok csomag van jelen a hálózatban, akkor torlódásról beszélünk.
A túl sok csomag általában a teljesítőképesség visszaesését, a csomagküldések késleltetését okozza.

A torlódás kialakulásának számos oka lehet, pl. egy hiba miatt hirtelen kieső eszköz, agresszív magatartás, rosszindulatú támadás, túl sok felhasználó stb.

A torlódás akár összeomlást is okozhat: ha minden csomag késve ér célba, akkor addigra már elavultak lesznek, újra fogják kérni őket az eszközök. Ettől mégtöbb csomag lesz a hálózatban, ami mégkésőbb ér célba, és így tovább...



#06/6 – Összefoglalás

Fogalom

- buffer (puffer)
- szegmens
- sliding window
- sávszélesség
- RTT
- tripla ACK
- torlódás

#06/7 – Sliding window trükkök

A window viselkedése

A sliding window egy nagyon ügyes megoldás (tud lenni, ha okosan csináljuk).
Az ablak az alábbi négy műveletet tudja:

nyílik
csak a jobb széle jobbra mozdul

A fogadó oldal nagyobb ablakot hirdet.

zsugorodik
csak a jobb széle balra mozdul

Lehetséges, de ellenjavalt!

csúszik
a jobb és bal széle is jobbra mozdul

A fogadó oldal nyugtázza a bal elem(ek)et.

becsukódik
az ablakméret nulla lesz

Elfogy az adat, vagy a fogadó 0-t hirdet.

Az egyik legrosszabb stratégia

Miért nem küldjük el az egész merevlemezt az interneten keresztül? (Végtelen nagy ablak.)

- A sávszélesség korlátoz minket felülről...
- A fogadó oldal nem biztos, hogy tud ilyen ütemben fogadni...
- Mások is használják a hálózatot (vagy legalábbis szeretnék)...
- Ha elküldtük a fél gépet, és kiderül, hogy a második szegmens elveszett, akkor megint csinálhatunk mindent előről...

A másik legrosszabb stratégia

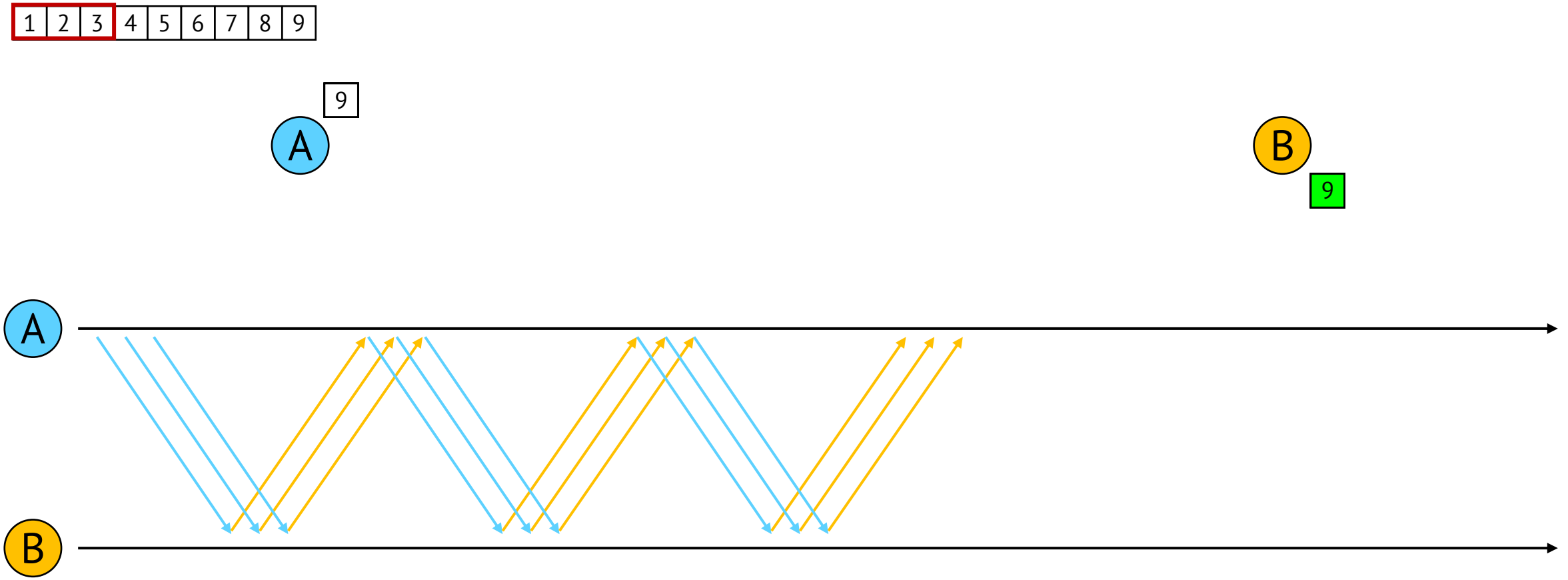
Miért nem várunk meg minden nyugtát, mielőtt elküldenénk az adatot?
(Végtelen kicsi ablak.)

- Na de ne már, hogy egyesével küldözgessünk...
- A fogadó oldal lehet, hogy többet is föl bírna dolgozni...
- Ezért kár fenntartani egy gigabites hálózatot...

Kísérlet

Az a cél, hogy folyamatosan tudjunk küldeni.

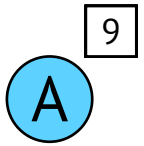
Ablakméret = 3

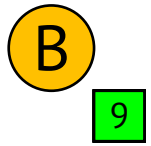


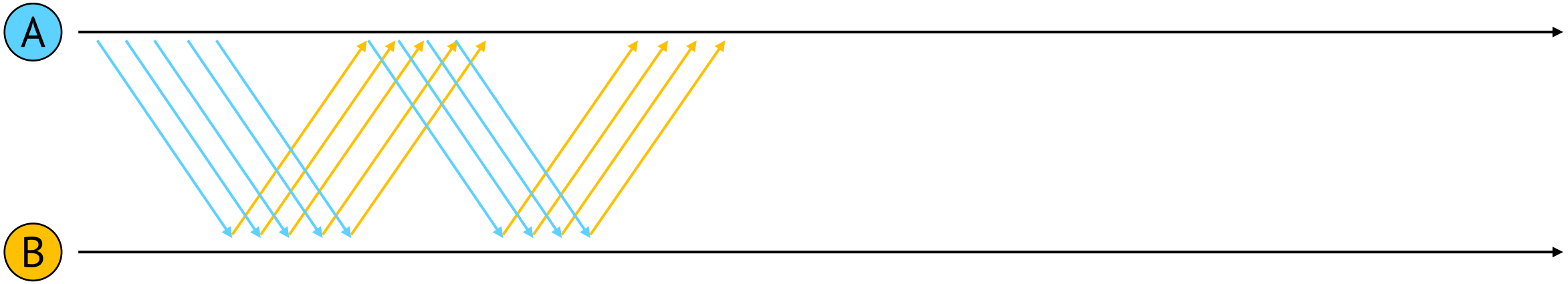
Kísérlet

Az a cél, hogy folyamatosan tudjunk küldeni.

Ablakméret = 5



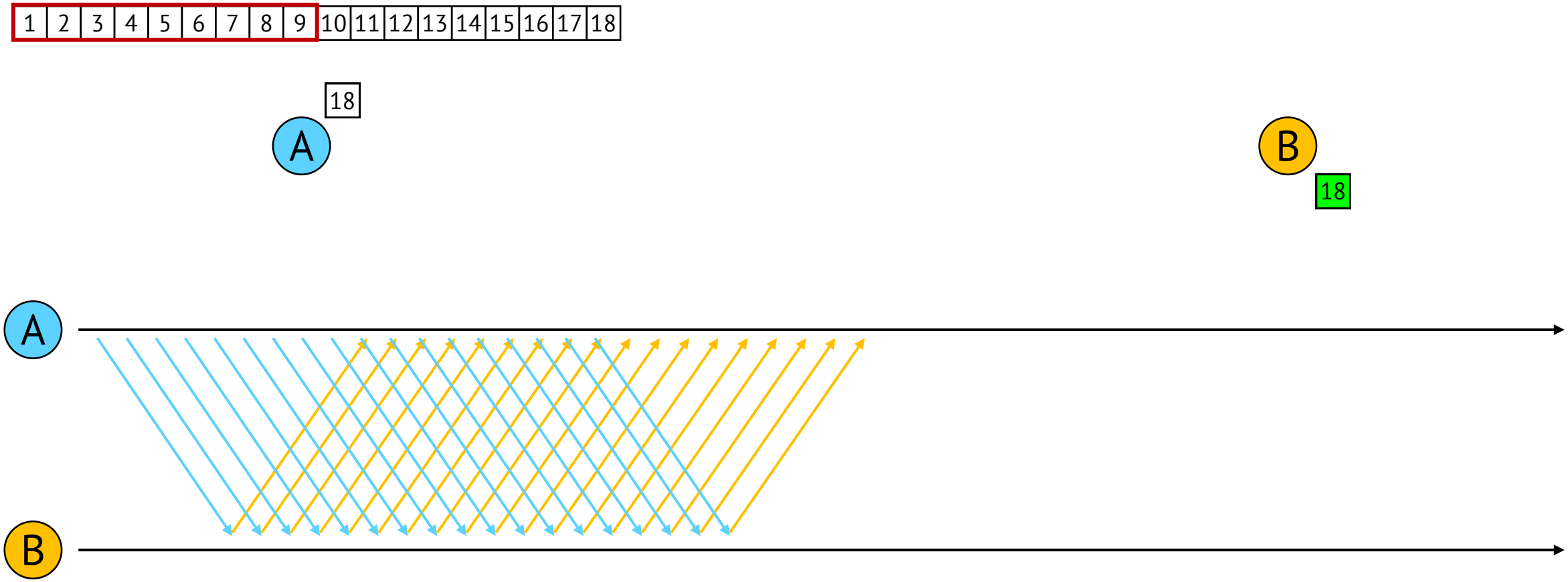




Kísérlet

Az a cél, hogy folyamatosan tudjunk küldeni.

Ablakméret = 9



Egy jó stratégia

Akkor küldhetünk folyamatosan, ha az első csomag nyugtájának megérkezéséig folyamatosan adhatunk (vagyis „nem fogy ki a tár” az ACK megérkezése előtt).

Mekkora legyen a window size?
Akkora, hogy a csomagfordulási idő alatt pont ne fogyjon ki belőle az adat.

Milyen sebességgel fogy ki belőle az adat?
Amekkora az adatátviteli sebesség.

Mindezekből következik, hogy a folyamatos adást biztosítani képes ablakméret legalább sávszélességszer RTT méretű kell legyen:

$$N\ [\text{bit}] \geq BW\ \left[\frac{\text{bit}}{\text{s}}\right] \cdot RTT\ [\text{s}]$$

Szemléltetés

A gyakorlatban sokszor látjuk, hogy hiába van gigabites internetünk, a kapcsolat felhasználói élmény szempontjából mégis „vacak”.

Ennek oka gyakran a magas RTT értékben keresendő. Ezt szoktuk laikusan ‚ping’-nek hívni, mert az kvázi az ICMP csomagfordulás során lemért RTT idő.

Ugyanazon kapcsolati sebesség mellett az eltérő RTT értékek az ablakméretet jelentősen befolyásolják:

100 Mbps kapcsolat és 10 ms RTT esetén az ablakméret 1 Mb = 125 kB

100 Mbps kapcsolat és 2000 ms RTT esetén az ablakméret 200 Mb = 25 MB

Window scale

SYN 1 bit	FIN 1 bit	window 16 bit	checksum 16 bit

A TCP fejrészben a window size mező 16 bites.

A fejrész alapvetően byte-okban adja meg a window méretét.

A 16 biten kifejezhető legnagyobb szám 65 535, azaz a max. ablakméret ≈ 65 kB.

De hát nekünk ennél nagyobb ablakok kellenek!?

Window scale

SYN 1 bit	FIN 1 bit	window 16 bit	checksum 16 bit

A TCP fejrészben a window size mező 16 bites.

A fejrész alapvetően byte-okban adja meg a window méretét.

A 16 biten kifejezhető legnagyobb szám 65 535, azaz a max. ablakméret ≈ 65 kB.

De hát nekünk ennél nagyobb ablakok kellenek!?

Vezessünk be egy opciót (a TCP fejrész options részét használva):

kind (3) 1 byte	length (3) 1 byte	shift count 1 byte

Window scale

kind (3) 1 byte	length (3) 1 byte	shift count 1 byte

kind: a TCP opció típusát határozza meg.
Értéke mindig 3 (ez jelenti, hogy ez egy window scale opció).

length: a TCP opció hozzát adja meg.
Értéke mindig 3, merthogy 3 byte hosszú az opció.

shift count: megadja, hogy 2-nek hányadik hatványa lesz a window size mező mértékegysége; mennyivel skálázzuk át az értéket.

Pl. shift count = 0 esetén 2^0 = 1 a skálázás; a 10-es window size 10 byte-os ablakot jelent.

Pl. shift count = 5 esetén 2^5 = 32 a skálázás; a 10-es window size 320 byte-os ablakot jelent.

Window scale

A window scale opcióban a shift count mező értéke maximum 14 lehet.

Ezzel a skálázási érték $2^{14} = 16\ 384$, és így a legnagyobb lehetséges ablakméret

$$65\ 535 \times 16\ 384 \text{ byte} = 1\ 073\ 725\ 440 \text{ byte} \approx 1 \text{ GB}$$

A window scale opció a SYN flaggel együtt küldhető a three-way handshake során.

Az opciót akkor tekintjük a felek által elfogadottnak, ha a partner azt a SYN-ACK üzenetben megismétli.

#06/7 – Összefoglalás

Fogalom	A window viselkedése
	Window scale
Stratégiák	Két rossz stratégia
	Egy jó stratégia

VÉGE

PÁZMÁNY

Pázmány Péter Katolikus Egyetem

Információs Technológiai és Bionikai Kar