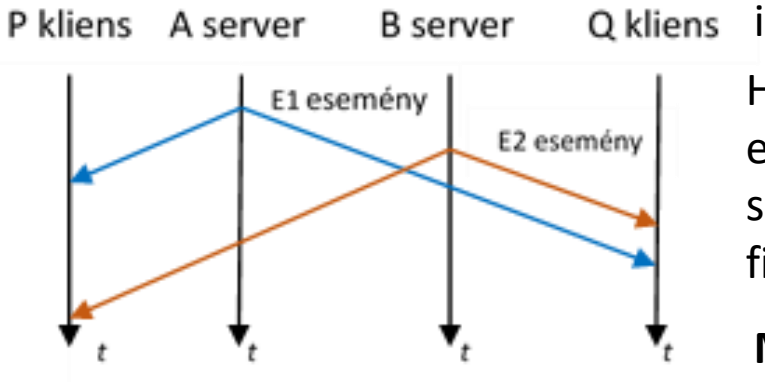
Beágyazott információs rendszerek

1. Bevezetés (folyt.)
2. Ütemezés

2020. szeptember 16.

Példák az időviszonyok sajátosságaira beágyazott rendszerekben:

- **relativisztikus hatás**: a kommunikáció időviszonyai események tényleges sorrendjét a vétel helyén megváltoztathatják. Az ábrán az látható, hogy a **Q** kliens esetében az **E2** eseményről szóló híradás megelőzi az időben korábbi **E1** eseményről érkező híradást.



Ha az **E1** és az **E2** események nem függetlenek egymástól, akkor jogos felvetés, hogy az **E2** eseményről szóló híradás megérkezését követően a híradásokat figyelembe vevő döntéseinkkel várakozzunk.

Meddig?

Addig, amíg minden olyan eseményről szóló híradás, amely az **E2** eseménnyel egyidejűleg vagy azt megelőzően történt – a legkedvezőtlenebb esetben is – beérkezik a **Q** klienshez.

Ezt a várakozási időt **akció késleltetési időnek** (*action delay*) nevezzük.

A szükséges akció késleltetési időt akkor tudjuk meghatározni, ha van minimális (alsó) és maximális (felső) korlátunk az üzenettovábbítási időre, azaz a d üzenettovábbítási időre fennáll:

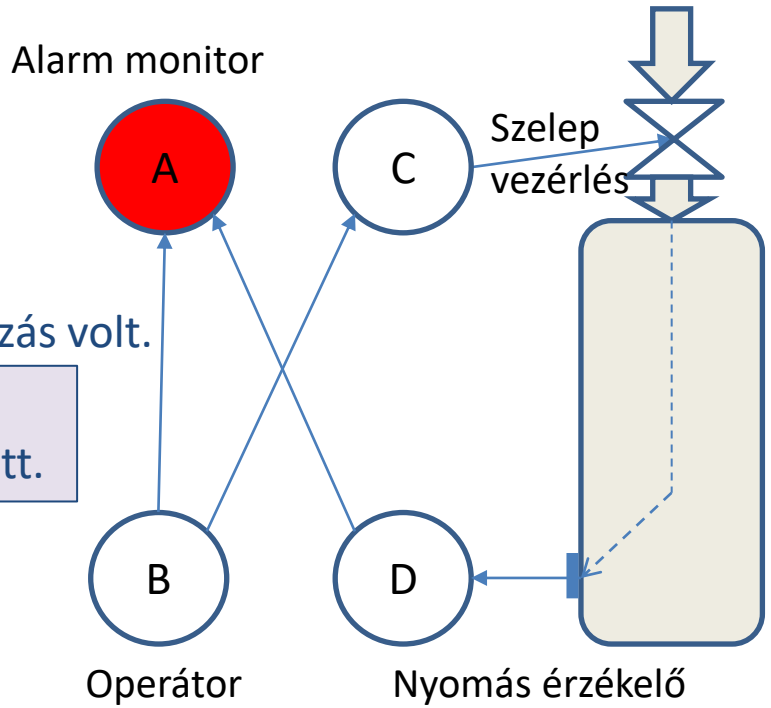
$$d_{min} \leq d \leq d_{max}$$

Mennyiségek, változók valós idejű rendszerekben

Példa: Egy tartályban lévő nyomást monitorozunk egy elosztott rendszerrel.

A csomópont: alarm monitor,
B csomópont: operátor,
C csomópont: szelep vezérlés,
D csomópont: nyomás érzékelő.

Lehetséges üzenetek:
$\mathbf{M}_{DA}$: jelzi, hogy a nyomás hirtelen megváltozott,
$\mathbf{M}_{BC}$: operátori parancs a változtatásra,
$\mathbf{M}_{BA}$: nincs alarm helyzet, mert operátori beavatkozás volt.



Van egy eltakart, a fizikai rendszer működéséből adódó csatorna a szelep és a nyomásérzékelő között.

Téves riasztás jöhet létre, ha
a $B \rightarrow C \rightarrow D \rightarrow A$ láncon gyorsabban fut végig az információ, mint a $B \rightarrow A$ láncon.

Ennek elkerülése érdekében az alarm monitor minden akcióját késleltetni kell.
(Bizonyos akciók visszavonhatatlanok: pilóta katapultál, lőfegyver elsül, stb.)

Vegyük észre, hogy maga a technológia Is kommunikációs csatornát valósít meg!

Megjegyzés: Párhuzam az 1986-os csernobili katasztrófával!

Mennyiségek, változók valós idejű rendszerekben

- ***Példa:*** az alábbi táblázatban néhány gépjármű motor jellemző szerepel együtt a megkívánt amplitúdó pontossággal és az ennek megfeleltethető időintervallumokkal.

RT kép a számítógépben	max. változás	pontosság	időbeni pontosság
Dugattyú pozíció	6000 ford/perc	0.1°	3μsec
Gázpedál pozíció	100%/sec	1%	10 msec
Motor terhelés	50%/sec	1%	20 msec
Olaj és hűtővíz hőmérséklet	10%/perc	1%	6 sec

A RT képek pontossági intervallumai között több, mint 6 nagyságrend eltérés van!
A dugattyú pozíció esetében ez a pontosság praktikusan csak állapotbecsléssel (a programon belüli jóslással) lehetséges.

A megfigyelés és a felhasználás között eltelt idő egy *v* változó esetén a következő hibát okozza:

$$hiba(t) \cong \frac{dv(t)}{dt}\left[C(t_{felhasználás}) - C(t_{megfigyelés})\right]$$

Ha egy időben pontos RT képet használunk, akkor a *worst-case* hiba:

$$hiba = \underbrace{\max}_{\forall t}\left|\frac{dv(t)}{dt}\right| d_{pontosság}$$

Mennyiségek, változók valós idejű rendszerekben

Kiegyensúlyozott tervezés esetén ez utóbbi az amplitúdó mérési hiba nagyságrendjébe kell essen. Ahhoz, hogy az RT képre alapozott számításaink pontosak legyenek, be kell tartanunk következő feltételt: $\left[C(t_{felhasználás}) - C(t_{megfigyelés})\right] \leq d_{pontosság}$

Példa az időbeni érvényességre:

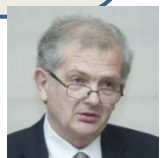
1993. szeptember 14, varsói repülőtér: egy Lufthansa A320-as Airbus túlszaladt a kifutópályán: 2 halott, 54 sebesült.
A balesetet az okozta, hogy a gép kilenc másodpercig csak az egyik oldali kerekén támaszkodott, ezért a fékező mechanizmusok bekapcsolása nem történt meg, mivel annak feltételeként a vezérlő logikában mindkét (fő)kerék földet érését írták elő.

Valójában az a következtetés, hogy "*a repülőgép még a levegőben van, ezért a fékező mechanizmusok nem aktiválhatók*" időben érvénytelenné vált abban a pillanatban, amikor az egyik kerék földet ért.

Egy periodikusan frissített RT képet ***parametrikusnak***, vagy ***fázis-érzéketlennek*** hívunk, ha

$$d_{pontosság} > \left(d_{frissítés} + WCET_{\text{üzenet továbbítás}}\right).$$

A parametrikus RT kép a vevő oldalon bármikor felhasználható anélkül, hogy a beérkezés és a felhasználás fázisviszonyait mérlegelni kellene: még a pontossági időn belül megjön a frissítés.



Mennyiségek, változók valós idejű rendszerekben

Egy periodikusan frissített RT képet *fázis-érzékenynek* hívunk, ha

$$WCET_{\text{üzenet továbbítás}} < d_{pontosság} < \left(d_{frissítés} + WCET_{\text{üzenet továbbítás}}\right).$$

Ilyenkor nem biztos, hogy a pontossági időn belül megjön a frissítés, ezért a frissítés és a felhasználás idejére oda kell figyelni.

Példa: A fenti táblázatban szereplő gázpedál pozíció továbbítási ideje 4 *msec*. Ha ekkor a periodikus lekérdezés üteme kisebb, mint 6 *msec*, akkor az RT kép *parametrikus*, ha pedig pl. 8 *msec*, akkor pedig *fázis-érzékeny*.

A fázis érzékenységet megfelelő mintavételi frekvenciával, vagy állapotbecslés alkalmazásával kerülhetjük el.

Idempotencia:

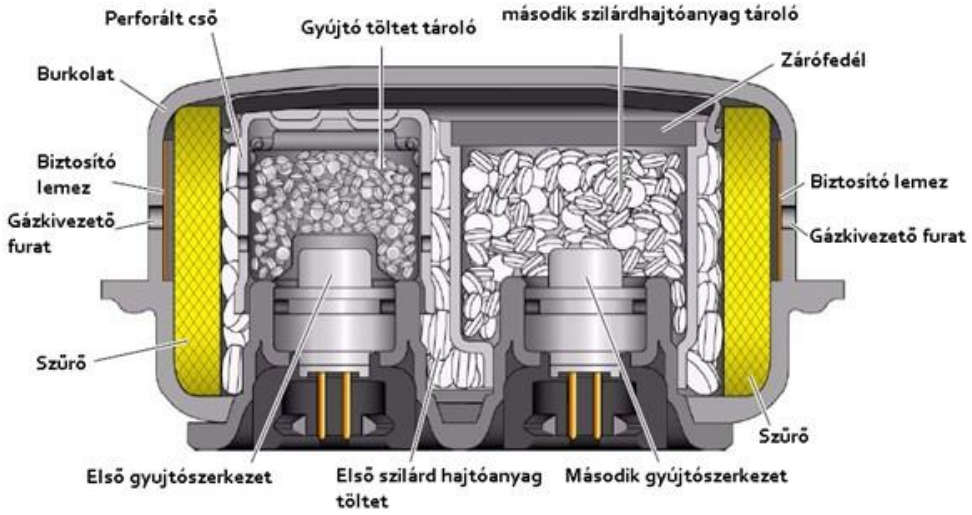
Ha ugyanaz az üzenet – tipikusan hibatűrési céllal – többször is megérkezik ugyanarra a csomópontra, akkor ezt az üzenethalmazt idempotensnek nevezzük, ha a a többszöri azonos üzenet hatása ugyanaz, mint az egyszerié.

Ez a fogalom azért fontos, mert ha az üzenet úgy konstruáljuk meg, hogy az megváltozást hordozzon, akkor a többszöri üzenetküldés többszöri "korrekciót" eredményez, miközben csak egyszerit szerettünk volna.

Példa: szelep-állás 45° (állapot üzenet) ↔ szelep-állás változás 5° (esemény üzenet).

Kemény és puha valós idejű rendszerek:

- **Kemény valós idejű rendszer** (hard real-time system (HRT)):
  katasztrofális következményekkel jár, ha nem tartjuk az időkorlátot (pl. légzsák vezérlése).
- puha valós idejű rendszer (soft real-time system (SRT), on-line system):
  az eredmény értékes az időkorláton túl is, de az idővel degradálódik (pl. banki rendszerek).



(Kormány)légzsák kioldó szerkezet

Bankautomata, ATM

HRT és SRT jellemzése különböző szempontok szerint:

- *válaszidő* (response time):

HRT esetében **msec,** vagy annál kevesebb (pl. légzsák), az emberi beavatkozás lehetősége kizárt, a rendszer autonóm működésű és biztonságos kell, hogy legyen.
SRT esetén a válaszidő **másodperc** nagyságrendű, az időkorlát túllépése nem okoz katasztrófát.

- *viselkedés csúcsterhelés esetén* (peak-load performance):

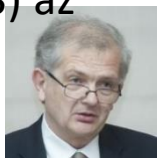
HRT esetén jól definiált kell, hogy legyen.

„Ha esik, ha fúj!”

Az **SRT** rendszereket **átlagos** teljesítmény-jellemzőkre tervezzük, a ritkán előforduló csúcsterhelések következményeit - gazdaságossági megfontolásból - elviseljük.

- *az ütem vezérlése* (control of pace):
  A **HRT** rendszernek minden körülmények között szinkronban kell lennie környezetének (irányított objektum (pl. autó), ill. az emberi operátor (pl. sofőr)) állapotával.
  Az **SRT** rendszerek befolyásolják környezetüket, ha nem képesek eleget tenni feladatuknak (egy tranzakciós rendszer például megnöveli a válaszidejét).
- *biztonság* (safety):
  A biztonság kritikusságának mértékétől függően sokféle feladat merül fel tervezési időben. Autonóm hibadetektálási mechanizmusok "talpra állítási" (recovery) akciók az adott alkalmazás által diktált időviszonyok mellett.
- *az adatfájlok mérete, adatintegritás* (size of data files, data integrity):
  HRT rendszerek: kisméretű adatfájlok, valós idejű adatbázisok, amelyekben az adatintegritás rövid idejű.
  SRT rendszerekben a hosszú idejű adatintegritás fontos. (Pl. bankszámla adatok.)
- *a redundancia típusa* (redundancy type):
  SRT rendszerekben (pl. tranzakciós rendszerek) hiba esetén a számításokat "visszagörgetik" a legutolsó ellenőrzési ponthoz, amikor még biztosan helyes volt a működés és onnan kezdik a "talpra állítást" (időredundancia).
  HRT rendszerek esetén ez a stratégia alig használható mert: (1) az időkorlát tartása nehéz, jósolható, (2) a környezetet befolyásoló "utasítás" nem tehető meg nem történtté, (3) az ellenőrzési pontnál érvényes adatok az idő múlásával érvényüket veszítik.

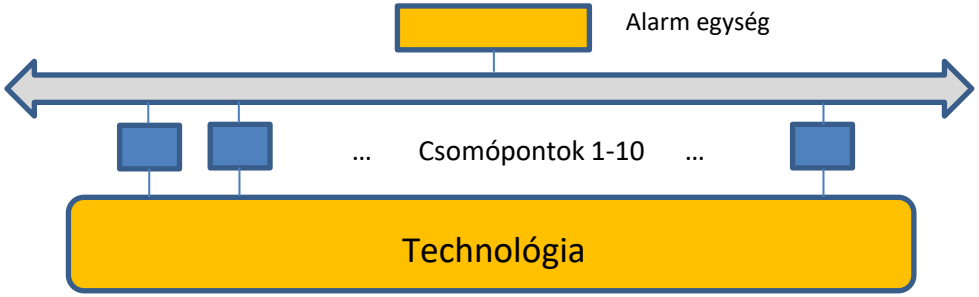
Hardver redundancia kell!



- *hibadetektálás* (error detection): **HRT**: autonóm, **SRT**: felhasználó által segített.

Eseményvezérelt (ET) és idővezérelt (TT) rendszerek:

Példa: Egy technológiai folyamatot 10 csomópont felügyel.



Mindegyik csomópont 40 bináris jelet (vészjelzés, például határérték átlépés információ) figyel.

A 10 csomópont egymással buszon kommunikál.

Ugyanide csatlakozik egy vészjelző (alarm) egység. A buszon a jelátviteli sebesség **100 kbit/s.**

A vészjelzésnek **100 ms**-on belül el kell jutnia az alarm egységhez.

Eseményvezérelt eset: ET/CAN protokoll szerint. A legkisebb átvihető üzenethossz a bájt.

A protokoll szabályai szerint felépülő üzenet teljes hossza: **44 bit** overhead, **1 bájt** üzenet, amit **4 bit** ún. intermessage gap követ. Ez összesen **56 bit.**

A **100 kbit/s** azt jelenti, hogy az előírt **100 msec**-en belül **10 000 bit** jut át.

56 bites üzenetekben gondolkodva **10 000/56 ~ 180** juthat át a specifikált határidőn belül.

Mivel **180 < 400**, ezért egyidejűleg valamennyi jelzés átküldésére nincsen lehetőség.

Idővezérelt eset: TT/CAN protokoll szerint.

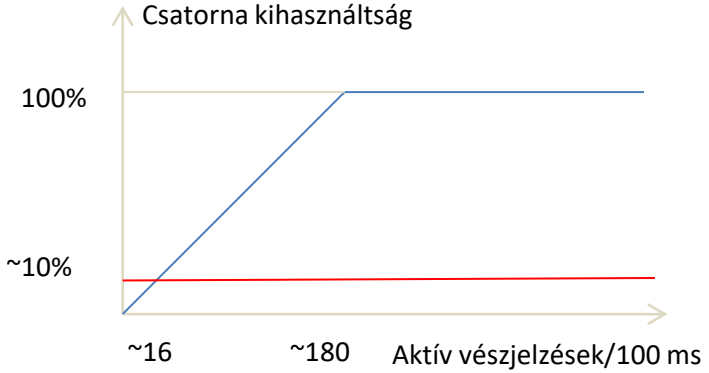
A csomópontok rendszeresen beküldik az állapotjelző biteket az alarm egységnek.

Ez **40** bitenként egy-egy üzenet.

A protokoll szabályai szerint felépülő üzenet teljes hossza: **44 bit** overhead, **40** bit (**5 bájt**) üzenet, amit **4 bit** ún. intermessage gap követ. Ez összesen **88 bit**.

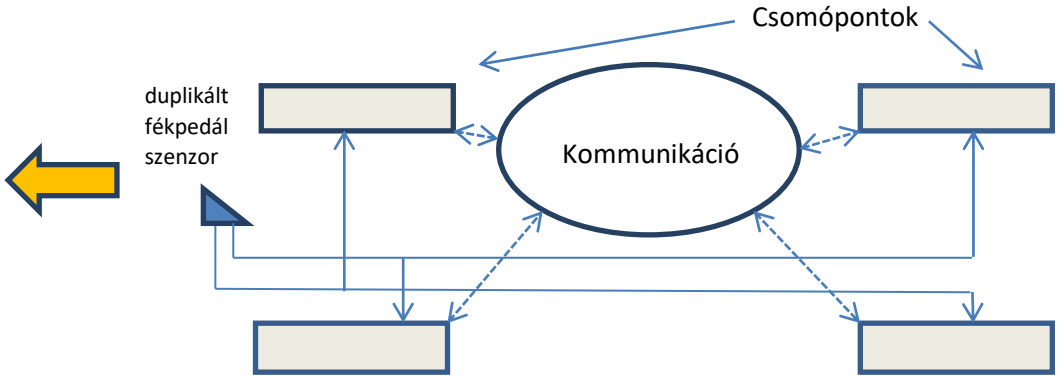
88 bites üzenetekben gondolkodva **10 000/88 ~ 110** juthat át a specifikált határidőn belül.

Mivel **110 > 10,** ezért valamennyi állapotjelző bit átjut az alarm egységhez, ráadásul állandó, **~ 10%-os** csatorna kihasználás mellett.



Megegyezés protokollok jelentősége

Példa: elektronikus fékvezérlés (brake-by-wire):

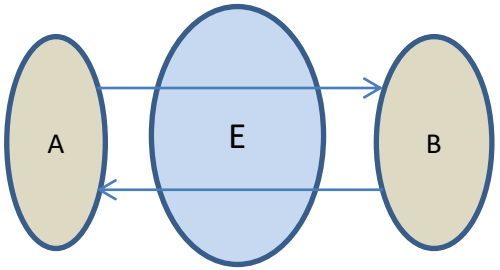


A példa szerint a biztonság érdekében **duplikált fékpedál szenzort** alkalmazunk.
Az egyes kerekek fékjeihez önálló vezérlő csomópontok tartoznak.
A csomópontok egymást tájékoztatják arról, hogy mi az ő véleményük a szenzor értékéről, és kiszámítják a fékerőt.

Ha megsérül egy csomópont, akkor automatikusan szabadonfutó lesz, ilyenkor nincsen fékhatás. A többi három, amikor észleli, hogy egy kiesett, automatikusan **újraszámítja a fékerőt**, és biztonságosan fékez.

Elosztott rendszerekben sokféle kérdésben szükséges futási idejű megállapodás:
idő szinkronizáció, elosztott állapotok konzisztenciája, elosztott kölcsönös kizárás, elosztott befejezés, elosztott választás, stb.
Hibák fellépése esetén is megállapodásra kellene jutni! **Ez nem mindig sikerül!**

Példa: Két hadsereg problémája:



A szövetséges **A** és **B** hadseregnek együttesen több katonája van, mint az **E** ellenségnek, de egyenként kevesebb. Megállapodásra kell jutni a támadás időpontjáról.

Ehhez kommunikálni kell, például hírnököt (**H**) küldeni, akit azonban elfoghat az **E** ellenség, tehát a kommunikáció **nem hibamentes**.

Ha **A** parancsnoka **H** hírnököt küld **B** parancsnokának, hogy holnap **délután 4-kor** támadjunk, akkor a nem hibamentes csatorna miatt **kell visszaigazolás**.

(Ettől függetlenül az is lehetséges, hogy B parancsnoka is küld hírnököt más időpont javaslattal.)

A probléma nyilvánvaló:

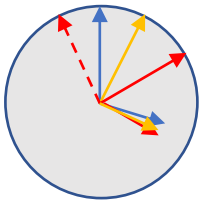
- Ha **H** nem tér vissza **A**-hoz, mi a konklúzió?
- Ha **H** a visszaúton esik fogságba, akkor **B** elindul adott valószínűséggel, de **A** nem fog, mert nem kapott visszaigazolást.
- Ha **H** az odaúton esik fogságba, akkor **A** van veszélyben, ha egyedül cselekszik.
- Ha **H** vissza is tér **A**-hoz, van valószínűsége, hogy **B** nem támad, mert nem tudja visszaért-e a hírnök. Ezt elkerülendő **B** elküldheti a saját hírnökét **A**-hoz, annak ellenőrzésére, hogy a visszaigazolás odaért-e.
- Ha újabb hírnököket küldünk, akkor nő annak valószínűsége, hogy a visszaigazolás átjut, de ez nem oldja meg alapvetően a problémát, mert mindig van véges valószínűsége, hogy a hírnököt elfogják.

Amit tehetünk: **a megegyezés valószínűségét növeljük.**

Lehetetlenségi tétel (Impossibility Result):

Formálisan bizonyítható, hogy nem garantálható, hogy két vagy több elosztott egység megegyezésre/megállapodásra jut véges idő alatt egy aszinkron kommunikációs közegen keresztül, ha a közeg veszteséges vagy valamelyik egység kiesik.

Megegyezés **bizánci típusú hibák** esetén: **Példa:** Órák szinkronizálása:



Az A óra 4.00-t mutat,
a B óra 4.05-t mutat,
a C óra az A-nak 3.55-t, a B-nek 4.10-et.

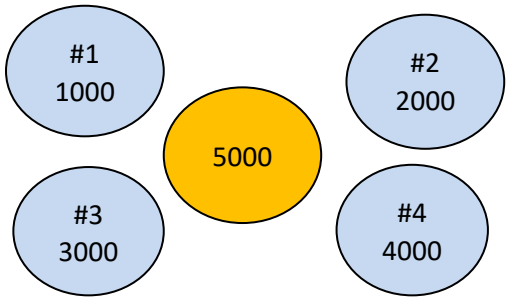
Ezt a hibafajtát nevezzük **bizánci típusú hibának**.

Ilyenkor nem jön létre a megállapodás, mert az A óra és a B óra is arra a megállapításra jut, hogy az általa mutatott érték a másik két óra által mutatott érték számtani közepe, tehát nincs indok megváltoztatni.

A bizánci típusú hibás csomópont kiszűrése akkor lehetséges, ha legalább **3*k*+1** csomópont vesz részt a szinkronizációban, ahol a ***k*** a bizánci típusú hibás csomópontok számát jelöli.

Esetünkben egy hibátlanul működő további óra-csomópont (D) szükséges a hibás csomópont kiszűréséhez.

Példa: A bizánci generálisok problémája:



Menetközben kiderül, hogy az egyik generális hazudós („szoftver hiba”). Az ellenségnek **5000** katonája van.

Az ábrán látható elrendezésben 4 hadtest generálisa megegyezésre törekszik az egyszerre harcba küldhető katonák számát illetően.
A (formális) szövetségesek egymással hibamentesen kommunikálnak: mindenki megküldi a katonái számát.
A **#3** számú generális/csomópont hazudós (szoftver hibás):
x, y, z a ténylegestől különböző, egymástól potenciálisan eltérő értéket küld a helyes (**3000**) helyett.

Az egyes csomópontokban az alábbi adatok állnak rendelkezésre:
#1: (1K, 2K, xK, 4K),
#2: (1K, 2K, yK, 4K),
#3: (1K ,2K ,3K, 4K),
#4: (1K, 2K, zK, 4K).

Annak érdekében, hogy az értékek helyes voltát ellenőrizni tudják, az információs vektoraikat körbeküldik a kommunikációs csatornáikon keresztül.

A csatornák az előzőek szerint fognak viselkedni, tehát a hazudós csomópont a körbeküldött vektor-elemeket illetően is hazudós.

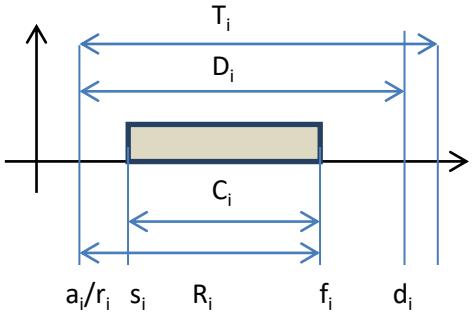
A körbeküldést követően az egyes csomópontokban a következő információ áll rendelkezésre (ezer katonában):

#1: $\begin{bmatrix} 1 & 2 & y & 4 \\ a & b & c & d \\ 1 & 2 & z & 4 \end{bmatrix}$ **#2**: $\begin{bmatrix} 1 & 2 & x & 4 \\ e & f & g & h \\ 1 & 2 & z & 4 \end{bmatrix}$ **#4**: $\begin{bmatrix} 1 & 2 & x & 4 \\ 1 & 2 & z & 4 \\ i & j & k & l \end{bmatrix}$

Mindhárom – nem hazudós – generális a három információs vektor esetében két helyről ugyanazt az információt kapja. A **#3**-as generálistól esetleges értékek érkeznek.

A nem hazudós generálisok következtetése, hogy $[\mathbf{1K} \quad \mathbf{2K} \quad \mathbf{ismeretlen} \quad \mathbf{4K}]$, azaz lesz legalább **7 ezer** katona, akire számítani lehet a támadásnál.

2. Ütemezés



Probléma: a processzor(ok)nak többféle időzítés mellett többféle feladatot (taszk) kell ellátniuk. Egy *i*-edik feladathoz (taszk-hoz) köthető időviszonyok az alábbiak szerint értelmezhetőek:

- $\mathbf{a_i}$ vagy $\mathbf{r_i}$ az érkezési idő (arrival/release/request time),
- $\mathbf{s_i}$ a végrehajtás kezdésének ideje (start time),
- $\mathbf{f_i}$ a végrehajtás befejezésének ideje (finishing time),
- $\mathbf{d_i}$ a végrehajtás határideje (deadline),
- $\boldsymbol{T_i}$ a periódusidő (period time),
- $\boldsymbol{D_i}=\mathbf{d_i}-\mathbf{a_i}$ a kérés időpontjához képesti határidő (deadline),
- $\boldsymbol{C_i}$ a számítási idő (computation time),
- $\boldsymbol{R_i}=f_i-a_i$ a válaszidő (response time).