# TÖL303G

# Gagnasafnsfræði

Snorri Agnarsson

# Færslur
# Transactions

# Færsla – Transaction

- Færsla í gagnagrunni er aðgerð eða runa aðgerða sem gerð er á gagnagrunninn sem breytir ástandi hans frá einu löglegu ástandi yfir í annað
  A transaction on a database is an operation or a sequence of operations on the database that changes its state from one legal state to another
- Athugið að á íslensku er orðið færsla notað bæði fyrir transaction og fyrir record – record er ekki sama og transaction

# ACID: Atomicity, Consistency, Isolation, Durability

- Við gerum yfirleitt fjórar meginkröfur til gagnagrunna
  - Four major requirements for database management systems
- **A**tomicity: **Ódeilanleiki**, sérhver færsla gerist í heilu lagi. Annaðhvort gerast heilar færslur eða engin breyting.
  - Each transaction either is performed completely or not at all
- **C**onsistency: **Samræmi**, sérhver færsla færir gagnagrunninn frá einu löglegu ástandi yfir í annað löglegt ástand.
  - Each transaction transforms the database from one legal state to another legal state
- **I**solation: **Raðbinding**, ef fleiri en einn aðili framkvæmir færslur á gagnagrunninn samtímis þá er lokaútkoman eins og ef færslurnar gerast hver á eftir annarri.
  - If more than one party executes transaction on the database concurrently then the final outcome is as if the transactions happen one after the other
- **D**urability: **Stöðugleiki**, allar færslur sem lokast (commit) halda sínum áhrifum þrátt fyrir villur svo sem rafmagnsbilanir, forritunarvillur eða kerfishrun
  - All transaction that are committed keep their effects in spite of errors such as power outages, programming errors or system crashes

# Áhersluatriði – Items Emphasized

1. Ódeilanleiki (atomicity) – vernd gegn áföllum – gagnagrunnurinn er ávallt í löglegu ástandi
   Atomicity – protection against errors – the database is always in a legal state
2. Raðbinding (isolation) – kemur í veg fyrir árekstra milli tveggja færslna. Hvernig er það útfært?

   Isolation – prevents collisions between two transactions. How is that implemented?

Sjá/See
SQLite: https://www.sqlite.org/transactional.html
Wikipedia: https://en.wikipedia.org/wiki/ACID

# Dæmi – Example

- Ef við keyrum
  - If we run

    INSERT INTO Product VALUES(‘Apple‘,101,‘Laptop‘);
    INSERT INTO Laptop VALUES(101,...);

- og það tekst aðeins að keyra fyrri INSERT skipunina þá töpum við gögnum og gætum skilið gagnagrunninn eftir í hálfkláruðu og ólöglegu ástandi því það er engin fartölva með númerið 101.
  - and only the first INSERT command is run to completion then we lose data and might leave the database in a half updated and illegal state because there is no laptop with the model number 101

# Raðbindingarkrafa (Isolation level)

- SQL gefur kost á að uppfylla nokkrar mismunandi kröfur um raðbindingu
  SQL offers options for isolation levels
- Serializable er traustasti kosturinn, en hægvirkast – Serializable is the most secure choice, but slowest
  - Sjálfgefinn kostur, notaðu það ef þú ert ekki viss – The default choice, use it if you are not sure

| Isolation Level | Dirty Reads | Nonrepeatable Reads | Phantoms |
|---|---|---|---|
| **Read Uncommitted** | Allowed | Allowed | Allowed |
| **Read Committed** | Prevented | Allowed | Allowed |
| **Repeatable Read** | Prevented | Prevented | Allowed |
| **Serializable** | Prevented | Prevented | Prevented |

# Dæmi – Example

- Gerum ráð fyrir að við viljum skrá nýja vöru í gagnagrunn, en aðeins ef varan er ekki þegar til – Assume we want to register a new product into the database, but only if the product does not already exist
- Þetta má gera í tveimur skrefum – This can be done in two steps

1. Keyra – Run
   SELECT COUNT(model) FROM Product WHERE model=?
2. Ef niðurstaðan er 0 þá keyra – If the result is 0 then run
   INSERT INTO Product VALUES(...);

- Ef tveir keyra skipun 1 á sama tíma og fá niðurstöðuna 0 þá gætu þeir báðir reynt að skrifa í gagnagrunninn – If two parties run command 1 at the same time and get the result 0 they might both try to write into the database
- Ef svo, þá ætti aðeins önnur færslan að heppnast – If so, only one of the transactions should succeed
- Pakka þarf þessum tveimur skipunum í eina færslu (transaction) og helst keyra með „Serializable“ – These two commands need to be packaged into a single transaction and preferably run with “Serializable“

# Möguleikar á afdrifum færslu – Possible results from a transaction

1. Algengast er að færsla (transaction) endi með fullnustu (commit) færslunnar – The most common result is that the transaction is committed
   - Þá er tryggt að allar breytingar sem færslan stendur fyrir gerðust í gagnagrunninum – Then it is certain that all the changes that the transaction stands for happened in the database
   - Þetta gerist í SQLite þegar SQL skipanirnar COMMIT eða END TRANSACTION eru framkvæmdar og takast – This happens in SQLite when the SQL commands COMMIT or END TRANSACTION are executed and succeed
2. Einnig er mögulegt að hætt sé við færsluna (rollback) – It is also possible that the transaction is rolled back (rollback)
   - Þá gerast engar breytingar á gagnagrunninum sem afleiðing viðkomandi tilraunar á færslu – Then no changes happen in the database as a consequence of the transaction attempt
   - Þetta gerist þegar gagnagrunnskerfið kemst að þeirri niðurstöðu að ekki sé hægt að fullnusta færsluna – This happens when the database system deduces that the transaction can not be committed
   - Getur einnig gerst sem afleiðing af beiðni forritara (ROLLBACK skipun) – Can also happen as a consequence of programmer request (ROLLBACK command)

# Ódeilanleiki – Atomicity – UNDO logging

- Þegar gögnum í gagnagrunninum er breytt í færslu og áður en COMMIT er framkvæmt (breytingarnar fullnustaðar) eru upphafleg gildi þeirra atriða sem breytast skráð í **hreyfingaskrá** (**log**) – When data in the database are changed in a transaction and before COMMIT is executed the original values of the data items that change are registered into a transaction log
  - Hreyfingaskráin inniheldur þá gömlu gildin á breyttum atriðum í gagnagrunninum – The transaction log then contains the old values of the changed items in the database
- Við verðum að tryggja að hreyfingaskráin sé uppfærð á diski **áður en** gagnagrunnurinn breytist á diski fyrir hverja einstaka breytingu – We must ensure that the transaction log is updated on disk **before** the database changes on disk for each single change
- Ef færslan fullnustast ekki eða ROLLBACK er framkvæmt þá breytum við gagnagrunninum aftur til baka með hjálp upplýsinganna í hreyfingaskránni – If the transaction is not successfully committed or a ROLLBACK is executed then we change the database to its former state using the information in the transaction log
  - Þetta kallast „UNDO logging“ – This is called “UNDO logging“
  - Einnig er til „REDO logging“ sem virkar öðruvísi – Another different logging method is called “REDO logging“

# JDBC færslur – JDBC transactions

- Í JDBC er sjálfgefið að hver skipun framkvæmir eina færslu –
  In JDBC the default is that each command executes one transaction
- Til að geta framkvæmt fleiri en eina skipun í einni færslu, notið –
  To be able to execute multiple commands in one transaction use
  ```
  conn.setAutoCommit(false);
  ```
- Til að framkvæma eina færslu – To execute one transaction:
  ```
  ... runa JDBC kalla sem er ein færsla
  … a sequence of JDBC calls that are one transaction
  conn.commit();        // fullnustar færsluna – commits the transaction
  ```
- Til að hætta við færslu – To roll back a transaction:
  ```
  ... runa JDBC kalla sem er ein færsla
  … a sequence of JDBC calls that are one transaction
  conn.rollback();      // hættir við færsluna – rolls back the transaction
  ```

# UNDO hreyfingaskrár – UNDO logs

- $\mathrm{START}(T)$
  - Byrjun færslu $T$ – Start of transaction $T$
- $\mathrm{WRITE}(T, X, old)$
  - Færsla $T$ ætlar að skrifa (eða hefur skrifað) eitthvert gildi í $X$, fyrra gildi $X$ var $old$ – Transaction intends to write (or has written) some value into $X$, the previous value of $X$ was $old$
- $\mathrm{COMMIT}(T)$
  - Færsla $T$ hefur verið fullnustuð – Transaction $T$ has been committed
- $\mathrm{ABORT}(T)$
  - Hætt hefur verið við færslu $T$ – Transaction $T$ has been aborted (rolled back)

# UNDO

- Við þurfum að skrá allar breytingar fyrir hverja færslu (transaction) í hreyfingaskrá (log)
  - We must log all changes for each transaction in the log
- Þar eð hreyfingaskráin þarf að lifa af í réttu ástandi þá þurfum við reglur um í hvaða röð má skrifa á disk

  Since the log must survive in a correct state we need rules about the order in which we write to disk

1. Ef færsla $T$ breytir $X$ úr $old$ verður að skrifa $\mathrm{WRITE}(T, X, old)$ í hreyfingaskrána áður en nýja gildið er skrifað í gagnagrunninn

   If transaction *T* changes *X* from *old* we must write $\mathrm{WRITE}(T, X, old)$ into the log before the new value is written to the database
2. Ef færsla $T$ framkvæmir COMMIT þarf að klára að skrifa allar breytingar í gagnagrunninn áður en $COMMIT(T)$ er skráð í hreyfingaskrána

   If transaction *T* executes a COMMIT then all the changes must be written into the database before $COMMIT(T)$ is written into the log
3. Ef hætt er við færslu þá þarf að klára að skrifa gömlu gildin aftur í gagnagrunninn áður en $ABORT(T)$ er skráð í hreyfingaskrána

   If transaction *T* is aborted (rolled back) then all the old values must be written back into the database before $ABORT(T)$ is written into the log

# Endurræsing gagnagrunnskerfis með UNDO

$F = \emptyset$
$U = \emptyset$
fyrir sérhverja aðgerð í hreyfingaskrá, í öfugri röð:
    ef færslan er $COMMIT(T)$ eða $ABORT(T)$ þá:
        Bætum $T$ við $F$
    ef færslan er $WRITE(T, X, v)$ og $T \notin F$ þá:
        Breytum gildi $X$ í $v$
        Bætum $T$ við $U$
fyrir sérhverja færslu $T$ í $U$:
    Bætum $ABORT(T)$ í hreyfingaskrána

Mengið $F$ mun innihalda færslur sem kláruðust
Mengið $U$ mun innihalda færslur sem hætt var við eða kláruðust ekki

# Restarting a database with UNDO

$F = \emptyset$
$U = \emptyset$
for each operation in the log, in reverse order:
    if operation is $COMMIT(T)$ or $ABORT(T)$ then:
        Add $T$ to $F$
    if operation is $WRITE(T, X, v)$ and $T \notin F$ then:
        Change value of $X$ to $v$
        Add $T$ to $U$
for each transaction $T$ in $U$:
    Add $ABORT(T)$ to log

The set $F$ will contain the transactions that succeeded (got commited)
The set $U$ will contain the the transaction that were aborted (rolled back) or that did not succeed

# Dæmi

- Gerum ráð fyrir töflu M(A,B) með eina röð (8,8) og að $T$ sé færslan

```
BEGIN TRANSACTION
        UPDATE M SET A=A*2;
        UPDATE M SET B=B*2;
COMMIT;
```

- Ef færslan klárast ekki (tölvan hrynur, til dæmis) þá þarf að skoða hreyfingaskrána, sjá hvaða breytingar hafa verið framkvæmdar og færa aftur í fyrra horf

# Example

- Assume a table M(A,B) with one row (8,8) and that $T$ is the transaction

```
BEGIN TRANSACTION
        UPDATE M SET A=A*2;
        UPDATE M SET B=B*2;
COMMIT;
```

- If the transaction does not finish (the computer crashes, for example) then the log needs to be examined to determine what changes have been executed and undo the changes

# REDO hreyfingaskrár

- Aðferðin með UNDO hreyfingaskrá skrifar óþarflega mikið á disk
  - Við getum ekki skrifað COMMIT fyrr en allar breytingar á gagnagrunninum hafa gerst
- REDO aðferðin geymir breytingar á gagnagrunninum í minni og klárar að skrifa í hreyfingaskrána áður en gagnagrunnurinn er uppfærður á diski
  - Við hrun þarf þá að fara gegnum hreyfingaskrána og klára breytingar sem ekki voru skrifaðar á disk
- Flestir gagnagrunnar nota bæði UNDO og REDO
  - SQLite notar UNDO og REDO með **`PRAGMA journal_mode=WAL`**

# REDO logs

- The method using UNDO logs writes an unnecessary amount of data to disk
  - We can not write a COMMIT until all changes to the database have happened
- The REDO method stores the changes to the database in memory and finishes writing to the log before the database is updated in external storage
  - When a crash happens we need to examine the log and finish the changes that did not make it into the database
- Most databases use both UNDO and REDO
  - SQLite uses UNDO and REDO with **`PRAGMA journal_mode=WAL`**

# Raðbinding – Isolation

- Gerum ráð fyrir töflu M(A,B) og íhugum tvær færslur, $T_1$ er
  Assume a table M(A,B) and consider two transactions, $T_1$ is

```
BEGIN TRANSACTION
        UPDATE M SET A=A+2;
        UPDATE M SET B=B+2;
COMMIT;
```

- og $T_2$ er – and $T_2$ is

```
BEGIN TRANSACTION
        UPDATE M SET A=A*2;
        UPDATE M SET B=B*2;
COMMIT;
```

- Færslurnar tryggja að $T_1$ og $T_2$ fléttast ekki en við getum ekki ákvarðað hvor keyrir fyrst – The transactions ensure that $T_1$ and $T_2$ will not be interleaved, but we can not be sure which runs first

# Lásar – Locks

- Til að ákvarða hvor færslan fær að keyra hverju sinni þarf að nota lása –
  To decide which transaction gets to run in each instance we need locks
- Færsla fær ekki að breyta gildi staks nema hún læsi stakinu fyrst með skriftarlási og aflæsi síðan þegar henni lýkur – A transaction is not able to change the value of a data item unless it locks the item first with a write lock and then unlocks when the transaction is done
- Það þarf lás til að lesa og það þarf lás (kannski annan) til að skrifa – A lock is needed to read and a lock (possibly a different one) is needed to write
- Ef einhver færsla hefur lestrarlás á staki þá getur önnur færsla fengið lestrarlás en ekki skriftarlás – If a transaction has a read lock on an item then another transaction can get a read lock, but not a write lock
- Ef færsla hefur skriftarlás á staki þá getur önnur færsla hvorki fengið lestrarlás né skriftarlás – If a transaction has a write lock on an item then no other transaction can get a write lock or a read lock
- Ef reynt er að fá lás og það reynist ekki hægt verður að hætta við í bili og reyna kannski aftur seinna eða hætta við færsluna – If an attempt is made to acquire a lock and that turns out not to be possible then the transaction must be (possibly temporarily) postponed or aborted.

# Tveggja fasa læsingar – Two-Phase Locking

- Færsla notar tveggja fasa læsingar ef beðið er um alla lása sem notaðir eru áður en nokkrum lás er sleppt
  A transaction uses two-phase locking if it asks for all necessary locks before any lock is released
- Ef allar færslur nota tveggja fasa læsingar þá er tryggt að raðbinding virkar, þ.e. við getum látið sem aðeins ein færsla sé keyrð í einu
  If all transaction use two-phase locking then it is ensured that transactions are isolated, i.e. we can pretend that only one transaction is executed at the same time
- Tveggja fasa lás tryggir að engin mistök eigi sér stað ef rétt er brugðist við þegar fullnustun er ekki möguleg
  Two-phase locking ensures that no mistakes are made if the correct response is made when committing is not possible
- Optionally see https://en.wikipedia.org/wiki/Two-phase_locking

# Sjálfhelda -- Deadlock

- Í dæminu að ofan, ef $T_1$ biður um skriftarlás á A og $T_2$ biður um skriftarlás á B þá geta báðar færslur fengið umbeðinn lás
  In the example above, if $T_1$ asks for a write lock on A and $T_2$ asks for a write lock on B then both transactions can get their locks
- Ef síðan $T_1$ biður um skriftarlás á B og $T_2$ biður um skriftarlás á A þá fær hvorug færslan umbeðinn lás og hvorug færsla getur haldið áfram hversu lengi sem þær bíða eftir að hin færslan sleppi sínum lás
  If subsequently $T_1$ asks for a write lock on B and $T_2$ asks for a write lock on A then neither transaction gets their lock and neither transaction can proceed, however long they wait for the other to release their lock
- Slíkar kringumstæður kallast sjálfheldur (deadlock) – This circumstance is called a deadlock
- Gagnagrunnurinn þarf að **bera kennsl** á sjálfheldur og **skera á hnútinn**
  The database system needs to **detect** deadlocks and **resolve it**
  - Annarri færslunni er eytt (ABORT) og sleppir þá öllum sínum lásum – One of the transactions is aborted and thereby releases all its locks
  - Hin færslan fær þá kost á að halda áfram og e.t.v. fullnustast (COMMIT) – The other transaction gets to continue and perhaps become committed

# Hvernig má koma í veg fyrir sjálfheldu?

- Ef öll forrit eru skrifuð þannig að þau **biðji um lása í sömu röð** þá er tryggt að forritin lendi ekki í sjálfheldu með læsingar
  - **Þ.e. „Deadlock prevention“**
- Það getur verið erfitt að fylgja þessari reglu
  - Því stundum er ekki hægt að vita fyrirfram hvaða lásar eru nauðsynlegir til að fullnusta færslu
- Reglan er hins vegar mjög gagnleg
- Hún er almennt gagnleg í samskeiða forritun
- Gagnagrunnskerfi, hins vegar, eru frekar byggð á aðferðum til að höggva á hnútinn í sjálfheldum
  - **Þ.e. „Deadlock resolution“**

# How can deadlocks be prevented?

- If all programs are written in such a way that they **request their lock in the same order** then it is ensured that the programs do not deadlock
  - **I.e. „Deadlock prevention“**
- This rule can be hard to follow
  - Because sometimes it is not possible beforehand to determine which locks are necessary to complete a transaction
- However, the rule is very useful
- It is generally useful in parallel programming
- Database systems, on the other hand, are commonly based on methods for **deadlock resolution** rather than deadock prevention

# Þriggja laga högun
# Three-Tier Architecture

Lagskipting upplýsingakerfa – Tiers of Information Systems

Valkostir -- Choices

Kostir og gallar – Advantages and Disadvantages

# Undirkerfi upplýsingakerfa – Subsystems of Information Systems

Gagnageymsla
Data Storage

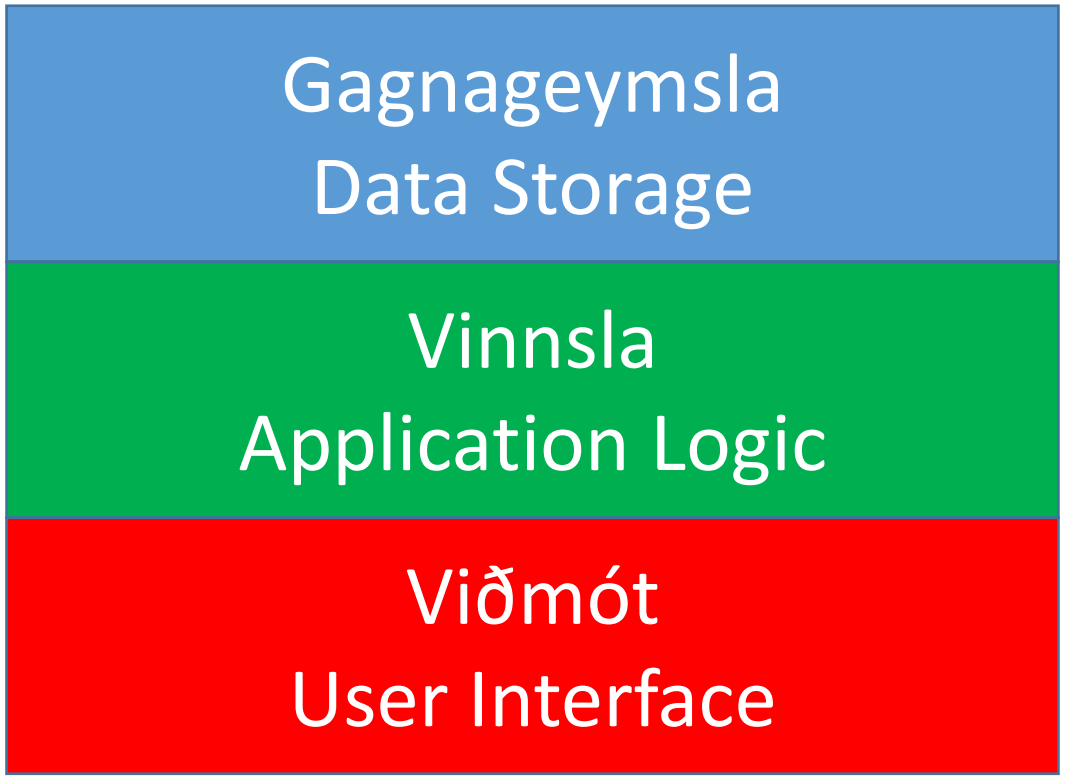
Vinnsla
Application Logic

Viðmót
User Interface

# Staðsetningar undirkerfa – Locations of Subsystems

- Gerum ráð fyrir dreifðu kerfi – Assume a distributed system
- Gerum ráð fyrir fleiri en einu viðmóti á fleiri en einum stað
  Assume more than one user interface in more than one location
- Gerum ráð fyrir að öll viðmót vinni með sömu gögn
  Assume all user interfaces work with the same data
- Viðmótin verða að vera hjá notendum
  The user interfaces need to be located with the users
- Gögnin verða að vera geymd miðlægt
  The data have to be centrally stored
- Vinnslan er einhvers staðar á milli
  The business logic is somewhere between

# Undirkerfi og lagskipting – Subsystems and Tiers



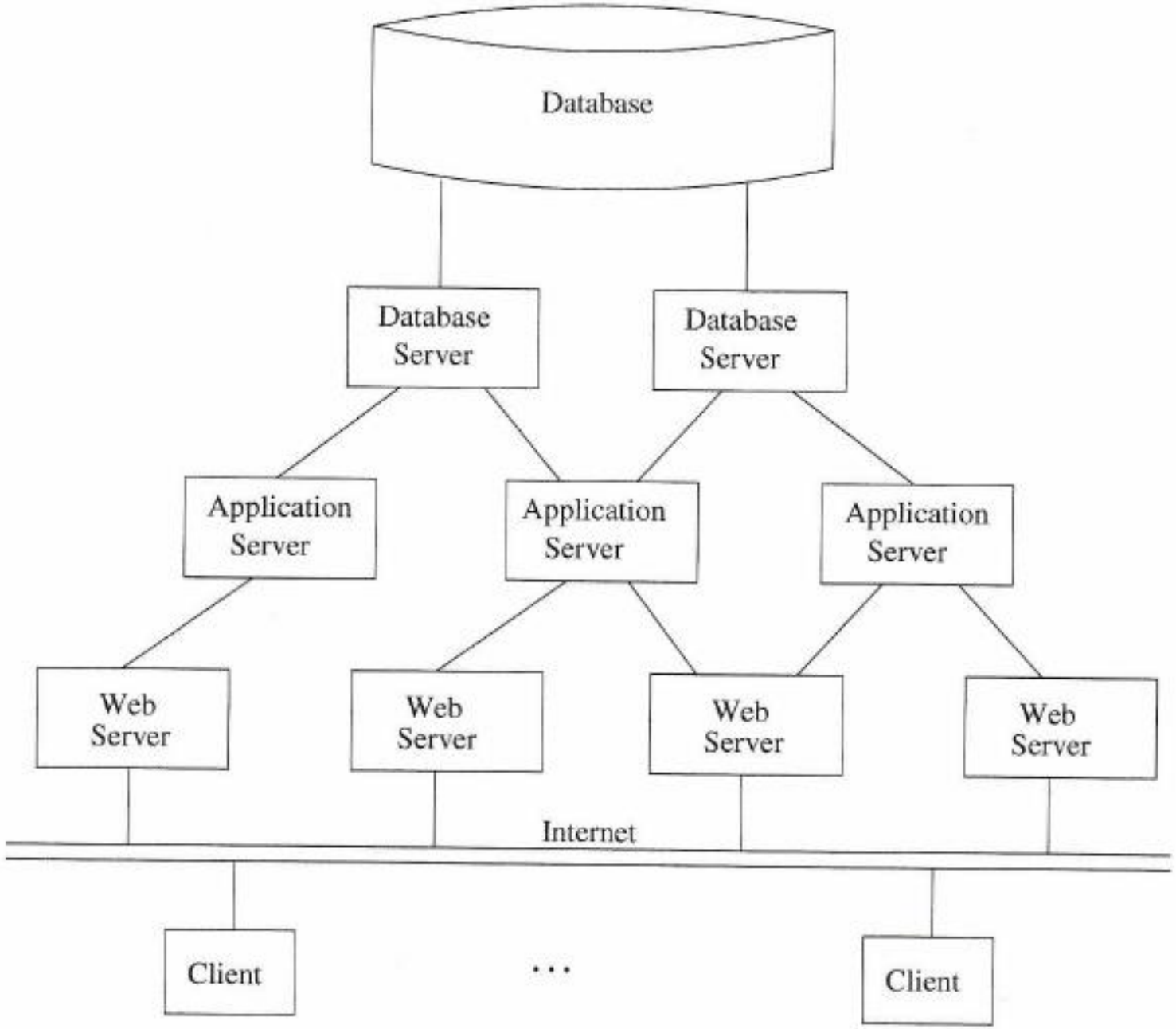
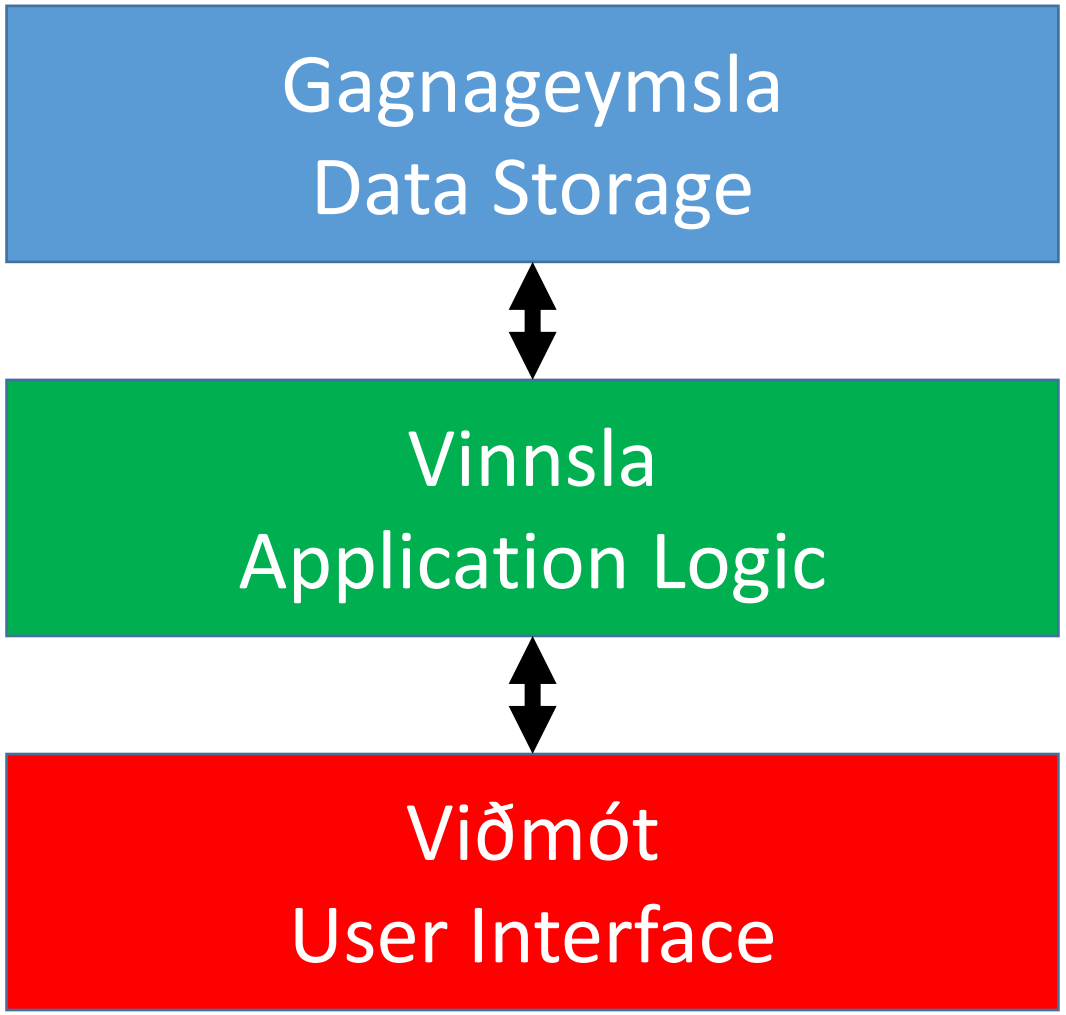
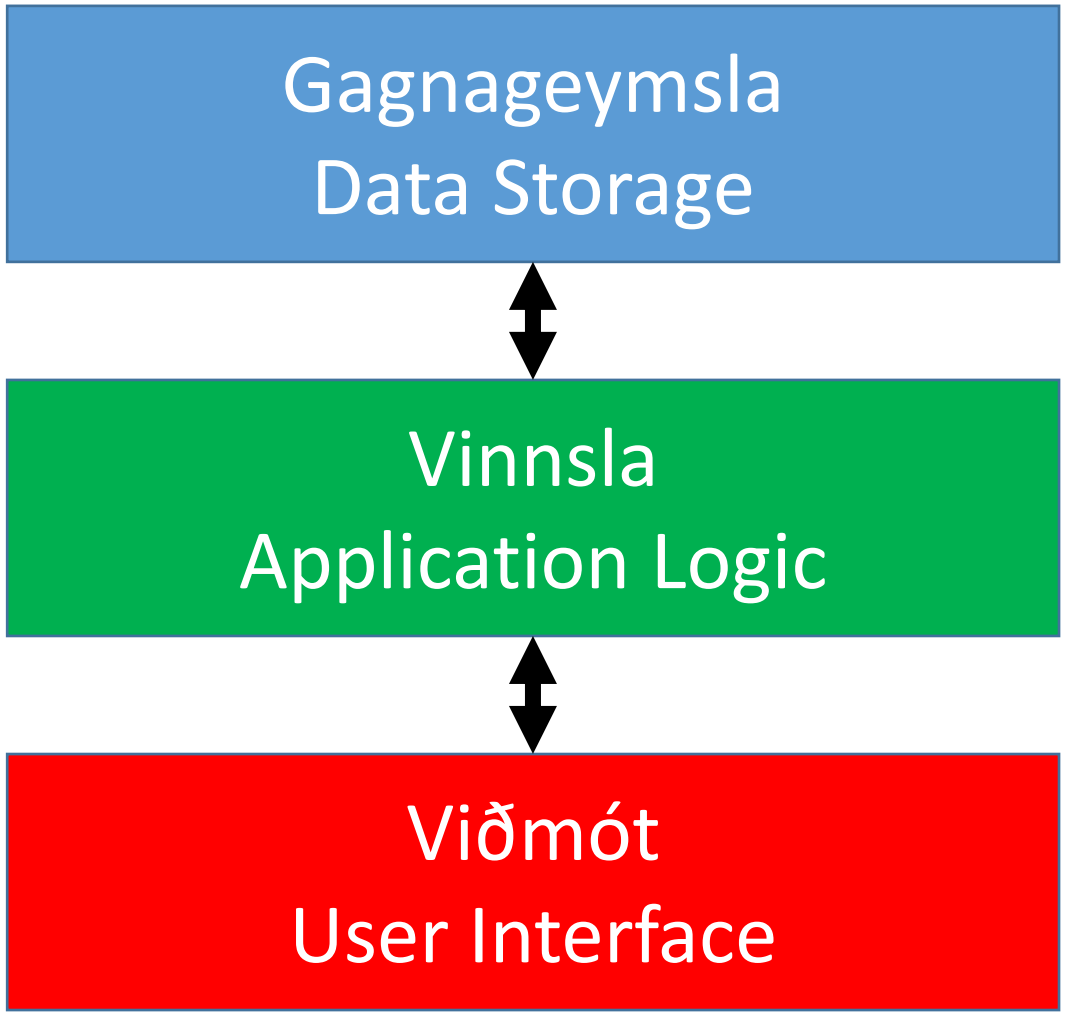



Figure 1: The Three-Tier Architecture

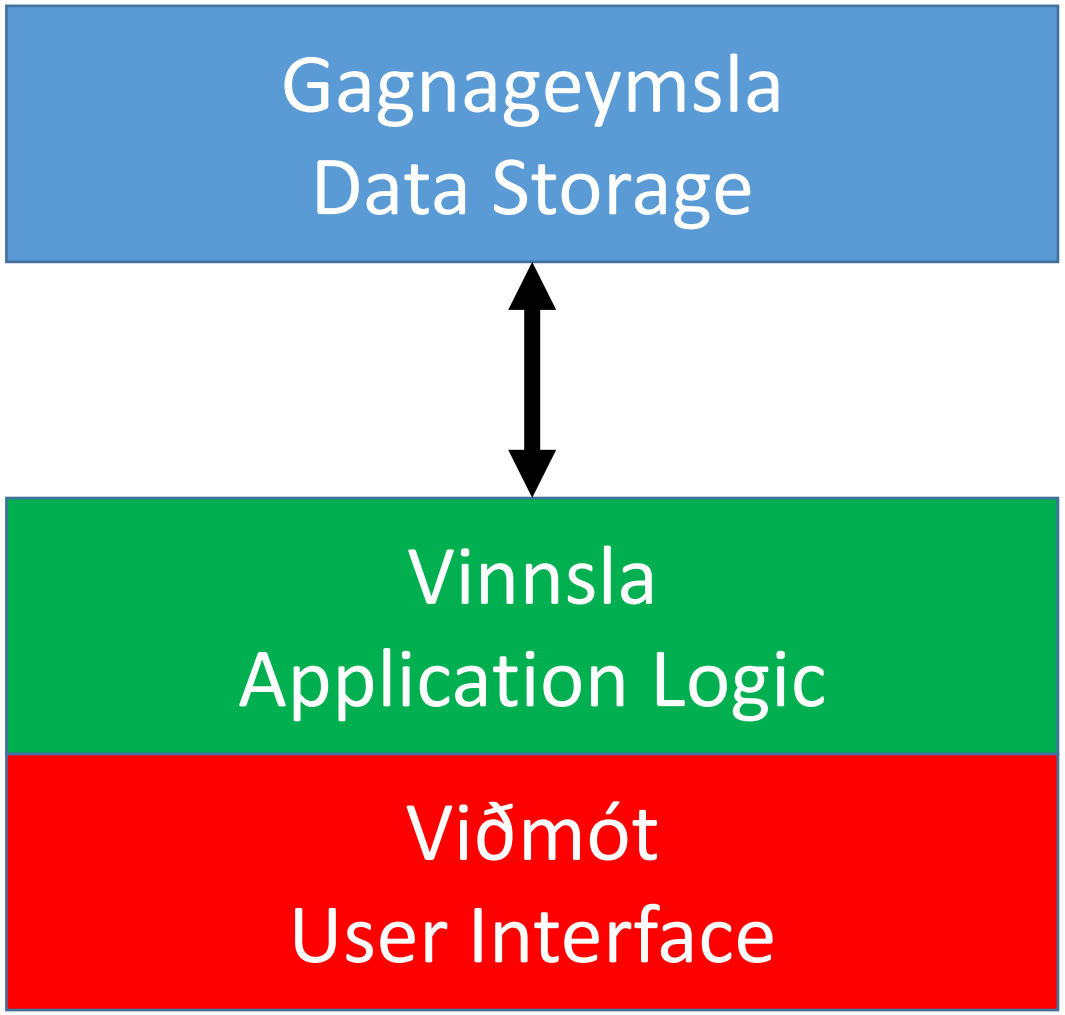
# Undirkerfi, lagskipting og samskiptaleiðir
# Subsystems, tiers and communication lines



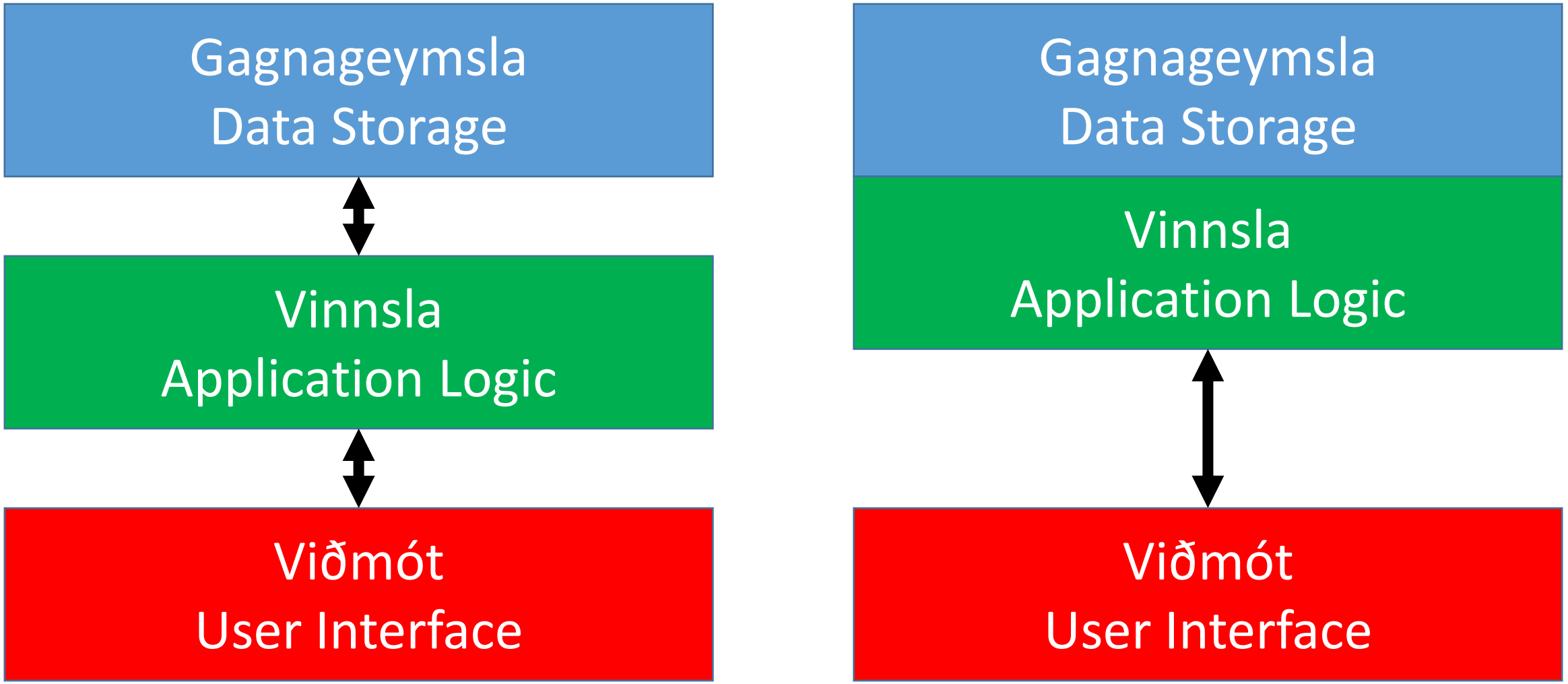
# Þriggja laga högun – Three-Tier Architecture



# Tveggja laga högun



# Þriggja laga högun – Three-Tier Architecture



# Hraði samskiptaleiða – Communication speed

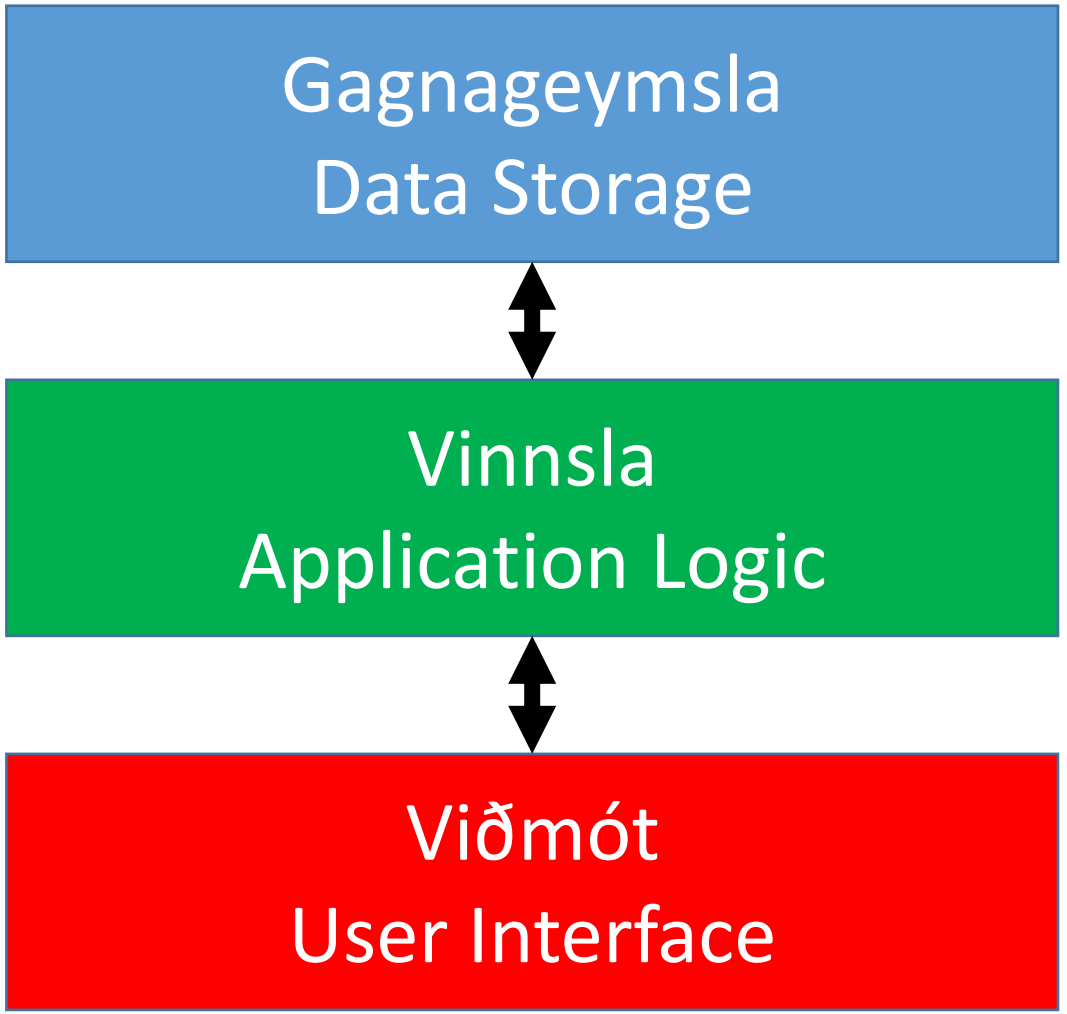
- Það er ekki hægt að skilgreina hraðann með einni tölu
- Bandbreidd (bandwidth) samskiptaleiðar segir til um magn gagna sem sendandi getur sett inn í samskiptaleiðina eða móttakandi tekið út úr samskiptaleiðinni á hverri tímaeiningu
  - Dæmigerðar einingar fyrir bandbreidd eru bæti á sekúndu og gígabitar á sekúndu
- Biðtími (latency, round-trip time) samskiptaleiðar segir til um tímann sem það tekur fyrir hverja litla upplýsingaeiningu (bæti, biti, pakki) að komast frá sendanda til móttakanda **og til baka**
  - Dæmigerðar einingar fyrir biðtíma eru millisekúndur eða nanósekúndur

# Hraði samskiptaleiða – Communication speed

- The speed can not be defined by a single number
- The bandwidth of a communication channel denotes the amount of data that a sender can pump into the channel or a receiver extract from the channel per time unit
  - Typical units for bandwidth are bytes per second or gigabits per second
- The latency (round-trip time) of a channel denotes the time it takes for each tiny information unit (byte, bit, packet) to travel from sender to receiver **and back**
  - Typical units for latency are milliseconds or nanoseconds

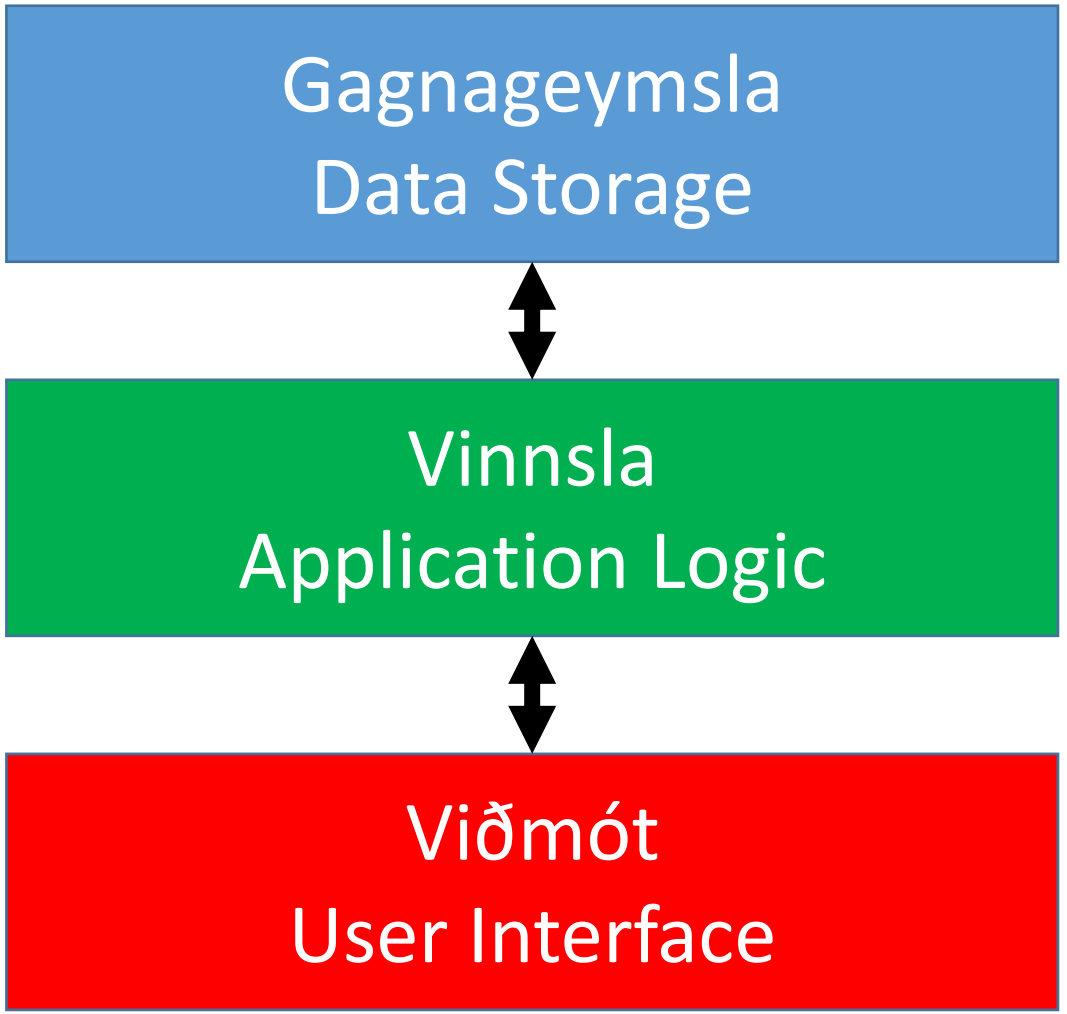
# Svartími – Response Time

- Svartími er sá tími sem það tekur fyrir tölvukerfi að sýna notanda niðurstöður aðgerða sem notandinn biður kerfið að framkvæma
- Þegar samskipti þarf milli kerfishluta til að framkvæma aðgerð þá hefur það áhrif á svartímann
- Hver er svartíminn $\boldsymbol{r}$ ef bandbreiddin er $\boldsymbol{b}$, biðtíminn er $\boldsymbol{t}$, magn gagna sem senda þarf er $\boldsymbol{d}$, fjöldi samskiptahringrása er $\boldsymbol{n}$ og reiknitími er $\boldsymbol{c}$?

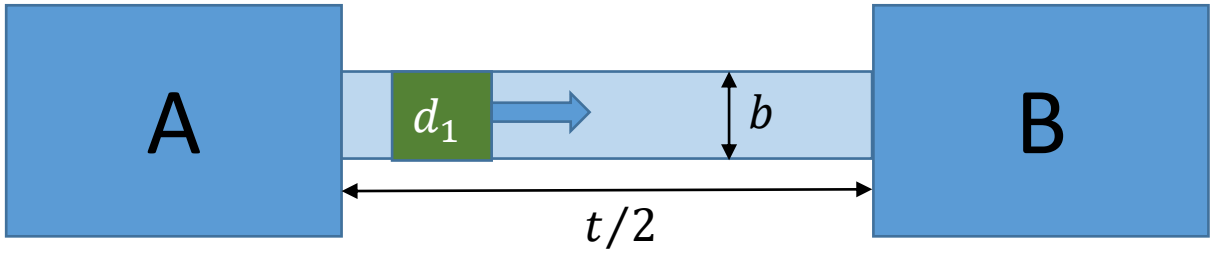


# Svartími – Response Time

- The response time is the time it takes for a computer system to show the users the results of operations that the user asks the system to perform
- When communication is needed between parts of the system to perform an operation then this affects the response time
- What is the response time $\boldsymbol{r}$ if the bandwidth is $\boldsymbol{b}$, the latency is $\boldsymbol{t}$, the amount of data sent is $\boldsymbol{d}$, the number of communication round trips is $\boldsymbol{n}$ and the computation time is $\boldsymbol{c}$?



# Svartímadæmi Example




- Þegar kerfishluti A sendir gagnamagn $d_1$ til hlutar B yfir samskiptaleið (svo sem net) með bandbreidd $b$ og biðtíma $t$ og kerfishluti B reiknar svo á tímabili af lengd $c$ og sendir svo gagnamagn $d_2$ til baka þá tekur það tíma (liðir í tímaröð)

$$r = \frac{d_1}{b} + \frac{t}{2} + c + \frac{d_2}{b} + \frac{t}{2}$$

- sem einfaldast í

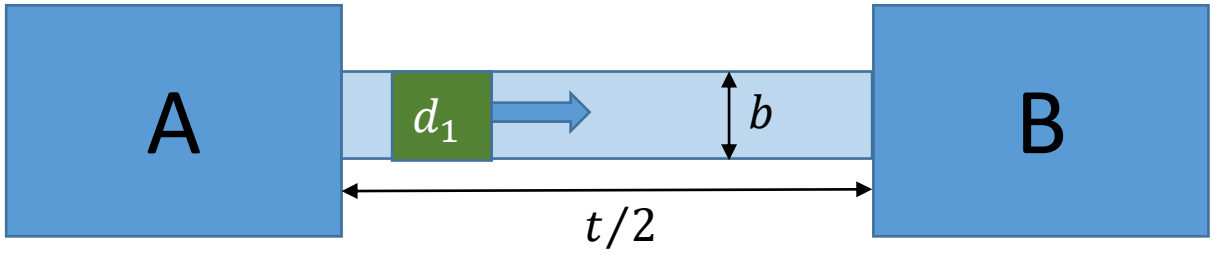
$$r = \frac{d_1 + d_2}{b} + t + c$$

- Athugið að þetta er **ein hringrás** gagna almenna formúlan fyrir $n$ umferðir er

$$r = \frac{d_{total}}{b} + n \cdot t + c_{total}$$

$d_{total}$ er **heildar**magn gagna sent
$c_{total}$ er **heildar**reiknitími
$n$ er fjöldi umferða
$t$ er biðtíminn
$b$ er bandbreiddin

# Svartímadæmi
# Example




- When a subsystem A sends data of size $d_1$ to subsystem B through a channel (such as a network) with bandwidth $b$ and latency $t$ and subsystem B computes for an interval of length $c$ and then sends data of size $d_2$ back to A then that takes time (the terms are in time order)

$$r = \frac{d_1}{b} + \frac{t}{2} + c + \frac{d_2}{b} + \frac{t}{2}$$

- Which simplifies to

$$r = \frac{d_1 + d_2}{b} + t + c$$

- Note that this is **one round trip** of data the general formula for $n$ round trips is

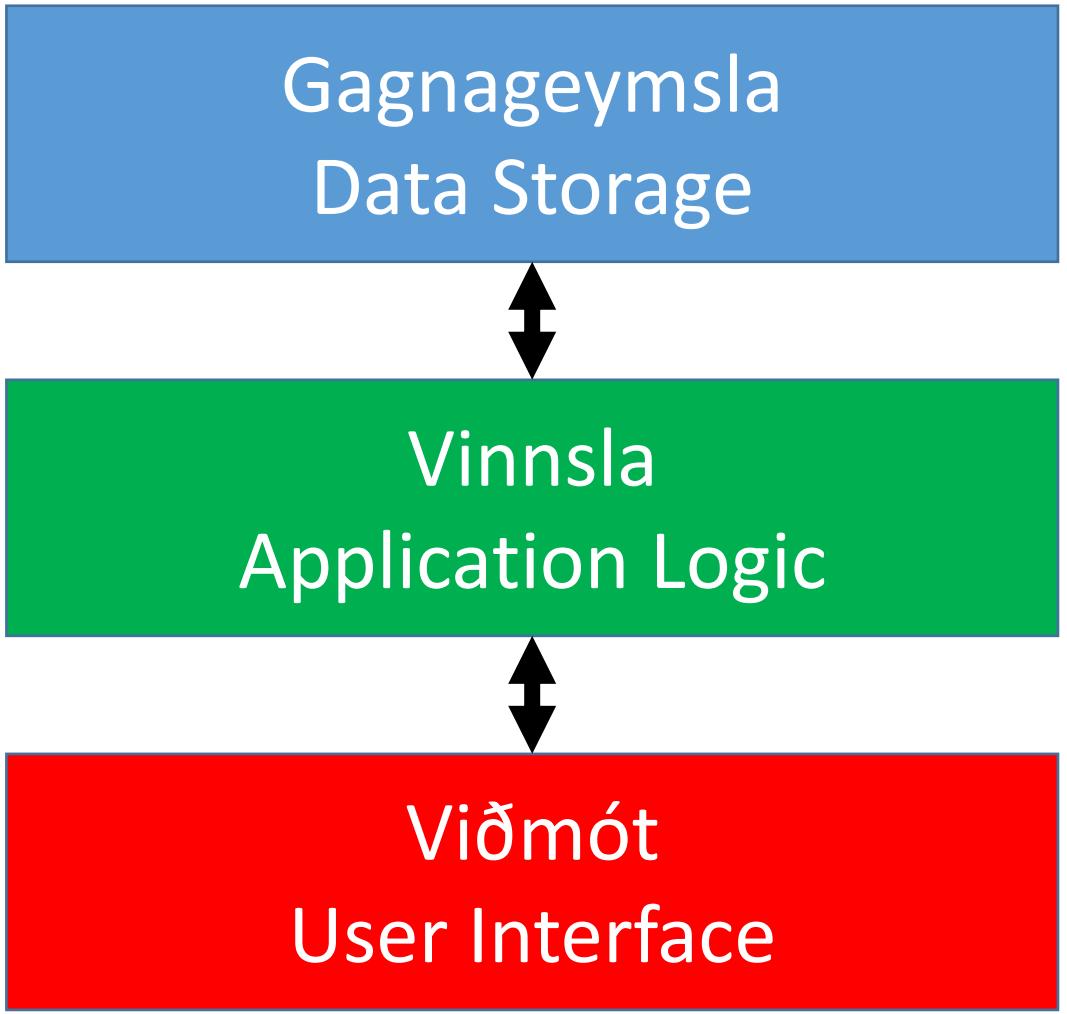
$$r = \frac{d_{total}}{b} + n \cdot t + c_{total}$$

$d_{total}$ is the **total** amount of data sent
$c_{total}$ is the **total** computation time
$n$ is the number of round trips
$t$ is the latency
$b$ is the bandwidth

# Svartími – Response Time

- Svartími er sá tími sem það tekur fyrir tölvukerfi að sýna notanda niðurstöður aðgerða sem notandinn biður kerfið að framkvæma
- Þegar samskipti þarf milli kerfishluta til að framkvæma aðgerð þá hefur það áhrif á svartímann
- Hver er svartíminn $\boldsymbol{r}$ ef bandbreiddin er $\boldsymbol{b}$, biðtíminn er $\boldsymbol{t}$, magn gagna sem senda þarf er $\boldsymbol{d}$, fjöldi samskiptahringrása er $\boldsymbol{n}$ og reiknitími er $\boldsymbol{c}$?

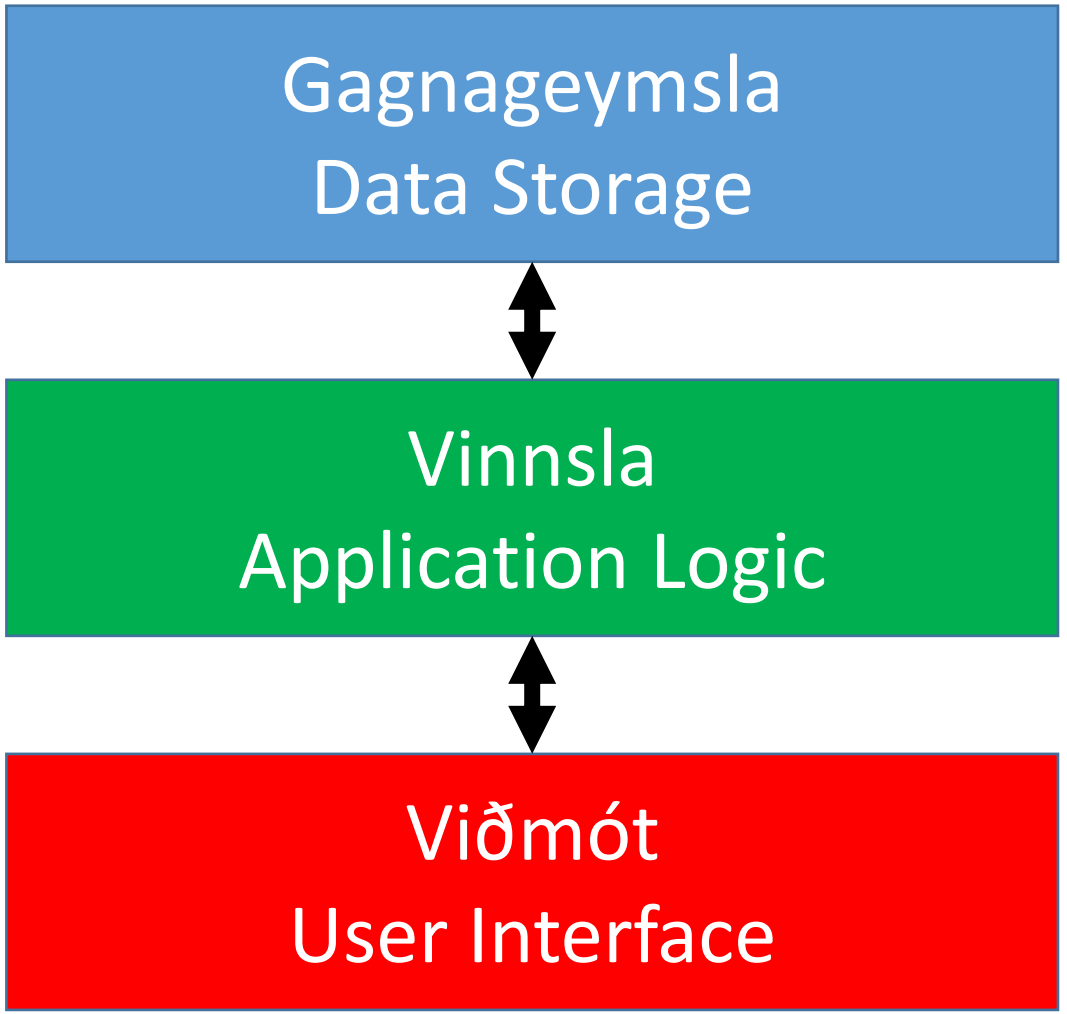
$$\boldsymbol{r = \frac{d}{b} + n \cdot t + c}$$



# Response Time

- The response time is the time it takes for a computer system to show the users the results of operations that the user asks the system to perform
- When communication is needed between parts of the system to perform an operation then this affects the response time
- What is the response time $\boldsymbol{r}$ if the bandwidth is $\boldsymbol{b}$, the latency is $\boldsymbol{t}$, the amount of data sent is $\boldsymbol{d}$, the number of communication round trips is $\boldsymbol{n}$ and the computation time is $\boldsymbol{c}$?

$$\boldsymbol{r = \frac{d}{b} + n \cdot t + c}$$

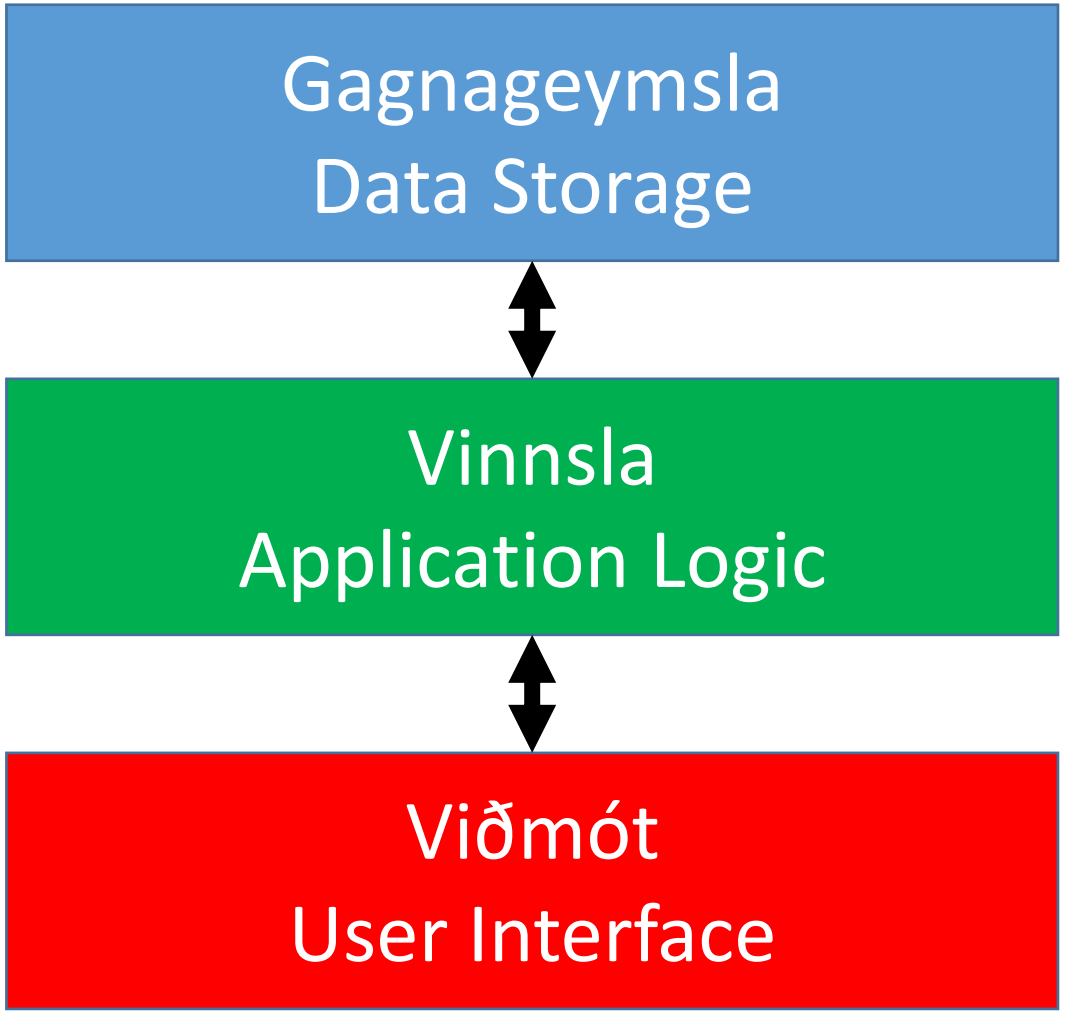


# Svartími og tíðni samskipta

- Svartími $\boldsymbol{r}$ ef
  - bandbreiddin er $\boldsymbol{b}$
  - biðtíminn er $\boldsymbol{t}$
  - magn gagna sem senda þarf er $\boldsymbol{d}$
  - fjöldi samskiptahringrása er $\boldsymbol{n}$ og
  - reiknitími er $\boldsymbol{c}$

$$\boldsymbol{r = \frac{d}{b} + n \cdot t + c}$$

- Hvar eru tíðustu samskiptin milli undirkerfa?
- Hvaða áhrif hefur það á hagstæðar staðsetningar undirkerfa?

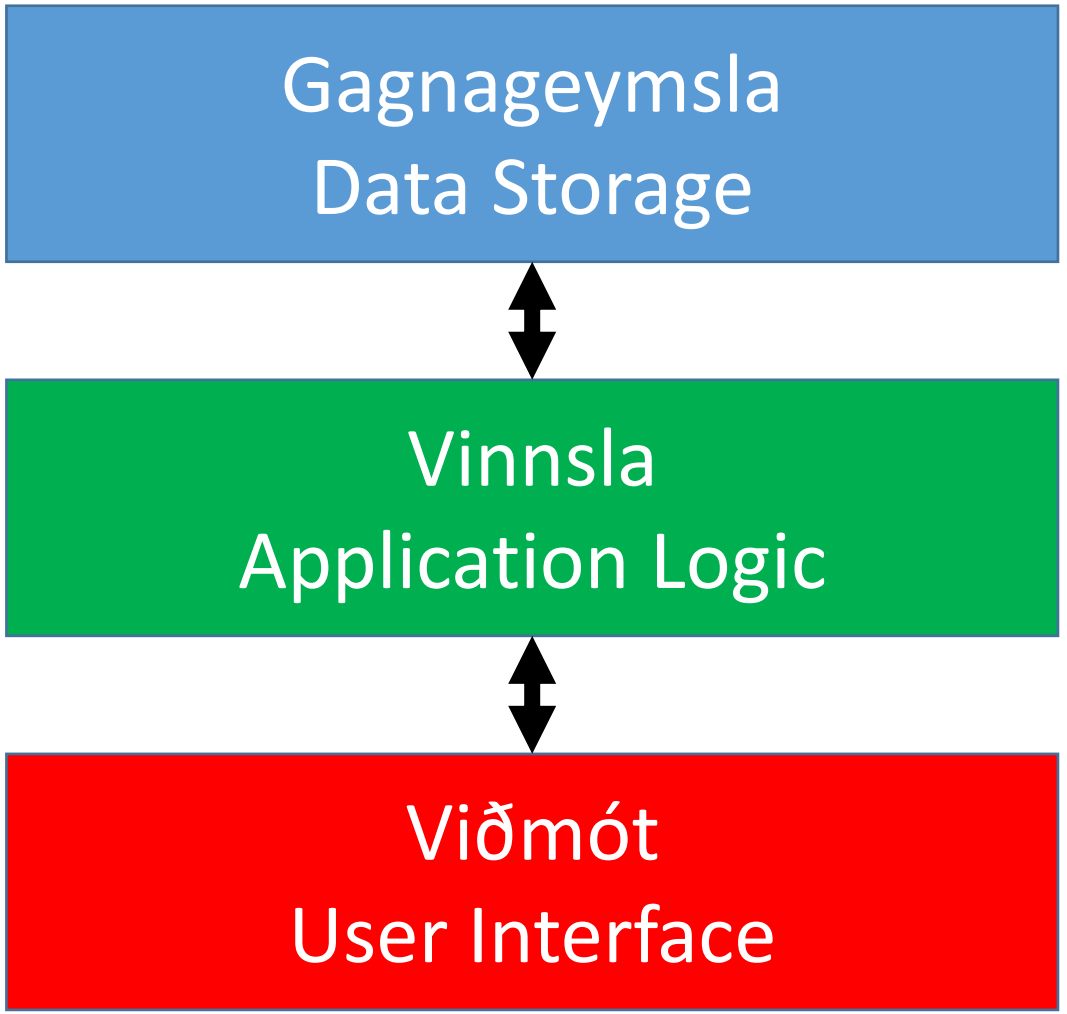


# Response time and communication frequency

- The response time $\boldsymbol{r}$ if
  - the bandwidth is $\boldsymbol{b}$
  - the latency is $\boldsymbol{t}$
  - the amount of data sent is $\boldsymbol{d}$
  - the number of connunication round trips is $\boldsymbol{n}$, and
  - the computation time is $\boldsymbol{c}$

$$\boldsymbol{r = \frac{d}{b} + n \cdot t + c}$$

- Where are the most frequent communications between subsystems?
- What effects does that have on the most optimal locations for the subsystems?



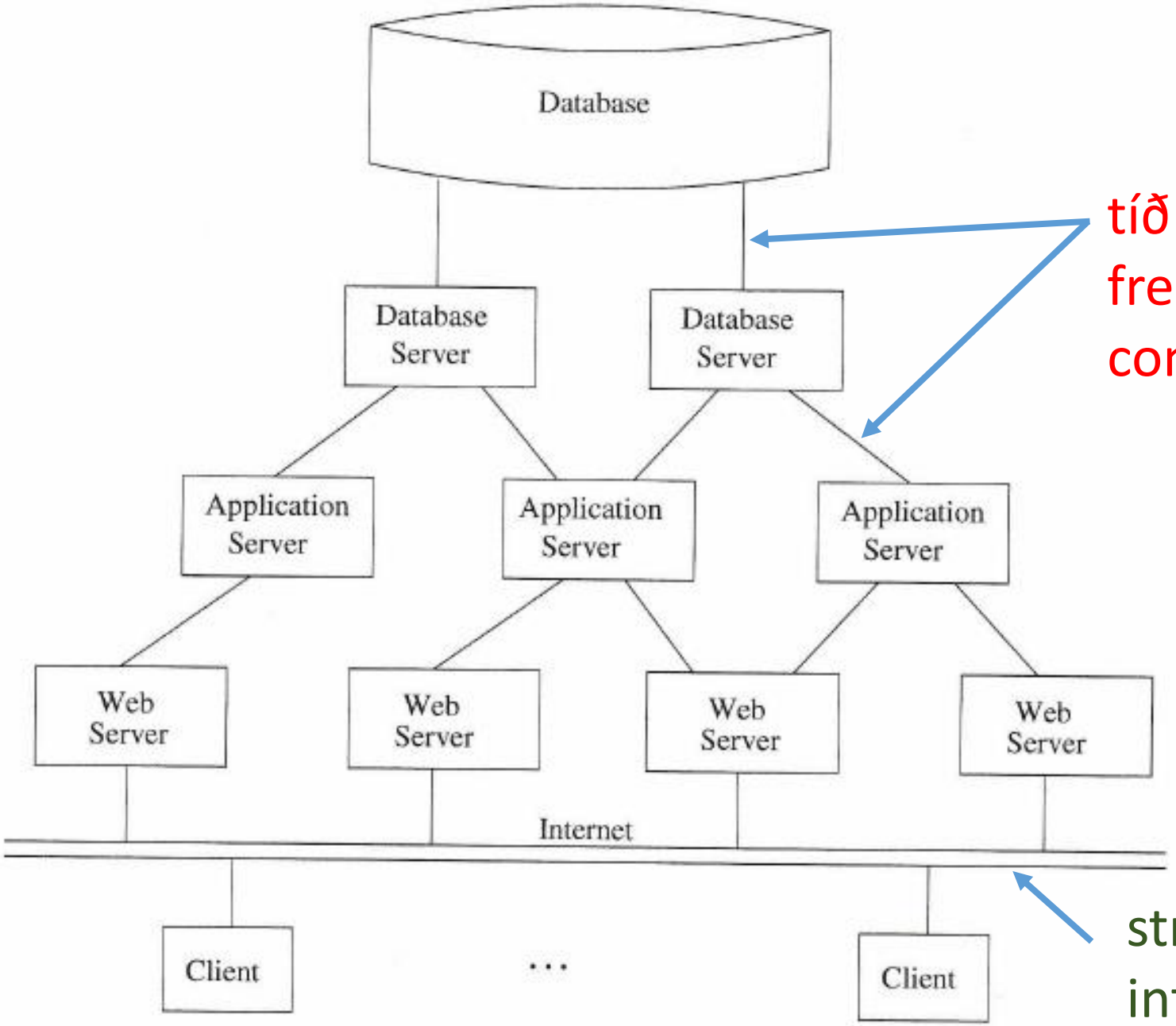



Figure 1: The Three-Tier Architecture

tíð samskipti
frequent
communication

strjál samskipti
infrequent
communication

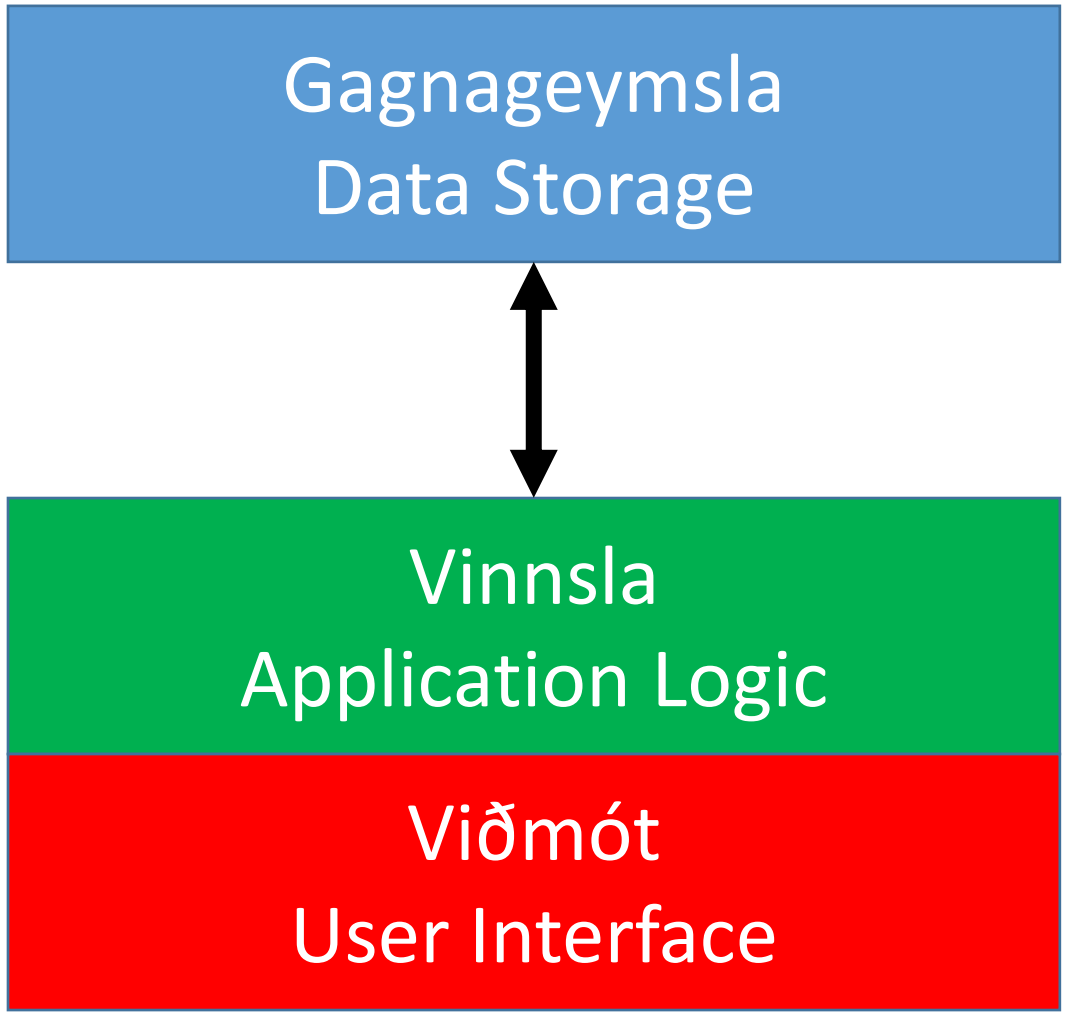
# Nokkrar tegundir samskiptaleiða

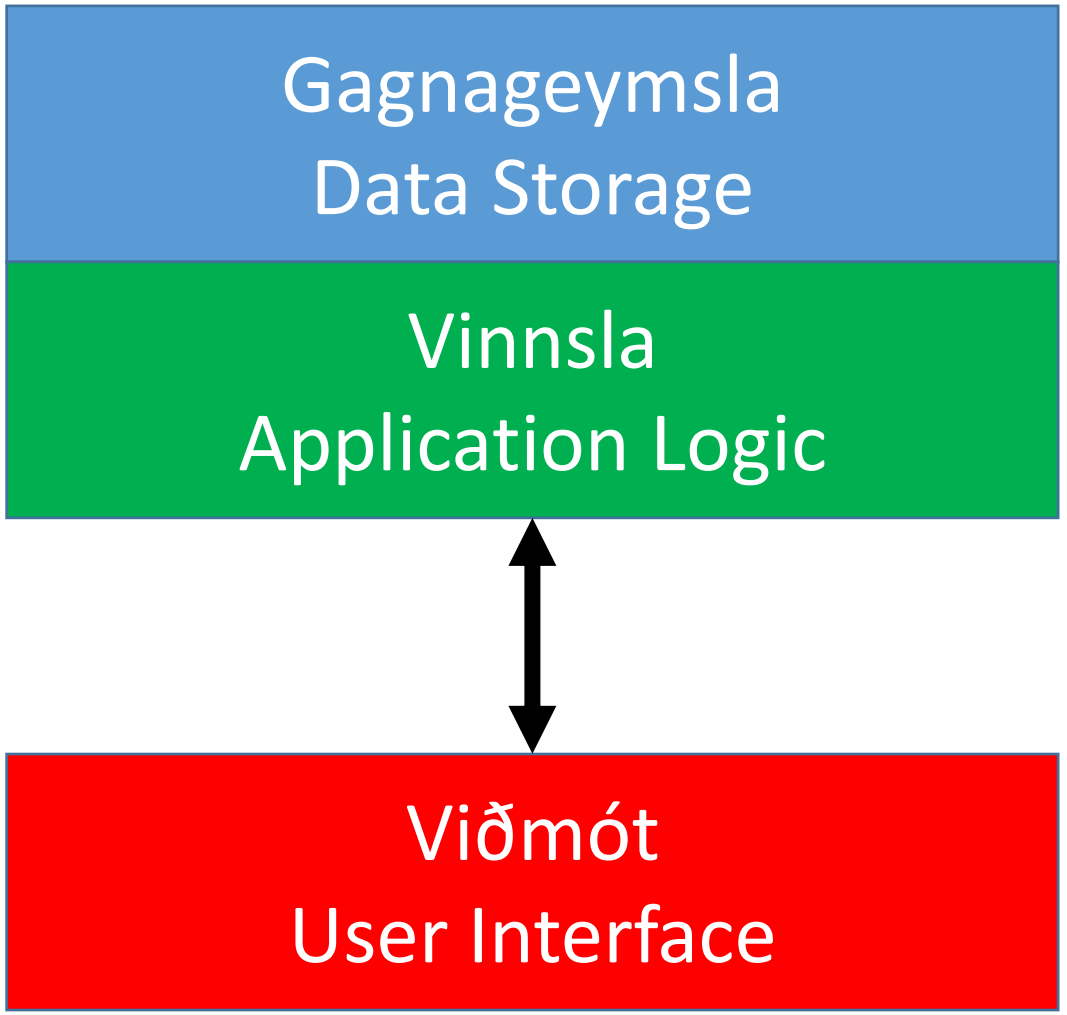
- Samskipti yfir stór net (WAN – wide area network)
- Samskipti yfir lítil net (LAN – local area network)
- Samskipti við jaðartæki (peripherals)
- Samskipti við minni (RAM)
- Samskipti við skyndiminni (cache)
- Samskipti við gisti (register)
- Sjá (aftur) https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

# Some types of channels

- Over WAN – wide area network
- Over LAN – local area network
- With peripherals
- With RAM memory
- With cache memory
- With a register
- Sjá (aftur) – See (again)
  https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

# Samanburður tveggja og þriggja laga
# Comparing Two-Tier and Three-Tier



# Samanburður tveggja og þriggja laga
# Comparing Two-Tier and Three-Tier