# Tölvunarfræði 1

## Fyrirlestur 19: Sýnidæmi: Síun

Hjálmtýr Hafsteinsson

Haust 2015

HÁSKÓLI ÍSLANDS
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

# Í síðasta fyrirlestri

- Endurkvæmni í myndum
- Turnarnir í Hanoi
- Gray kóðar
- Endurkvæm grafík
  - H-tré

**Kafli 2.3**

# Í þessum fyrirlestri

- Sýnidæmi um stórt forrit
- Síun (*percolation*)
- Hermun á síun í Java
  - Lóðrétt síun

**Kafli 2.4**

# Sýnidæmi (*case study*)

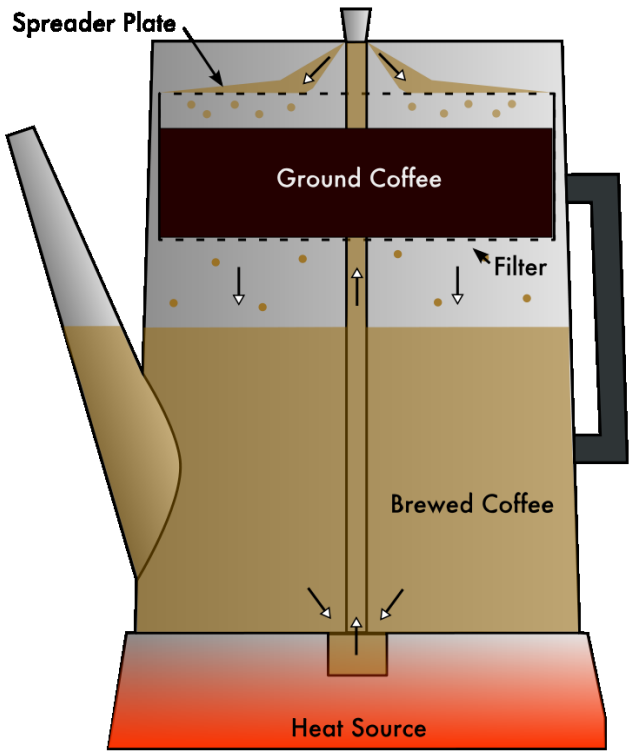
- Skoðum stærra forrit en venjulega
  - Sjá hvernig við skipuleggjum stór forrit
    - Allar ákvarðanir hafa afleiðingar
    - Reyna að sjá fyrir slæmar afleiðingar
  - Sjá endurnýtingu á forritskóða
  - Aflúsun (*debugging*) forrits

Höfum ýmsar þumalputtareglur

# Síun (*percolation*)

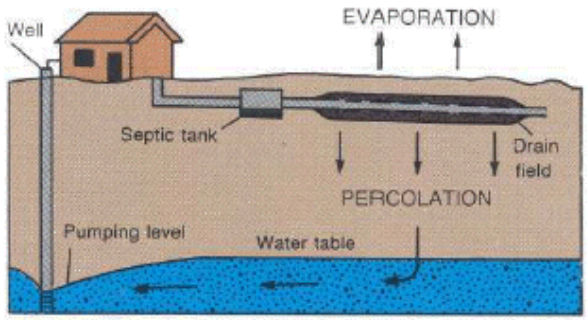
- Hella vökva yfir gljúpt efni, mun vökvinn ná botni?

- Einfalt dæmi:
  - Kaffivél:
    - Vatn látið leka niður í gegnum malaðar kaffibaunir
    - Vatnið síast í gegnum malaða kaffið og tekur í sig bragðið

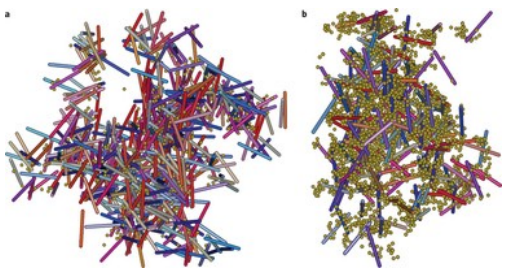


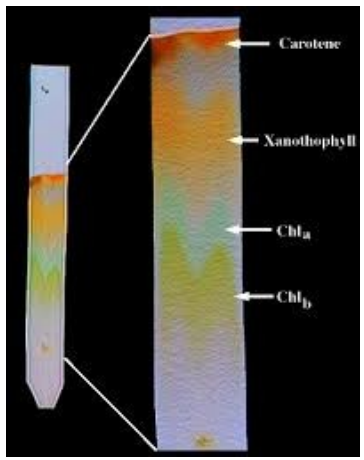
# Notkun síunar

- Raunverulegri notkunardæmi:
  - Hvernig jarðgas fer í gegnum jarðlög
  - Hvernig vatn frá rotþróm síast niður í grunnvatn




  - Litskiljun (*chromatography*) í efnafræði aðskilja efni í blöndu
  - Rafstraumur í gegnum blandað efni

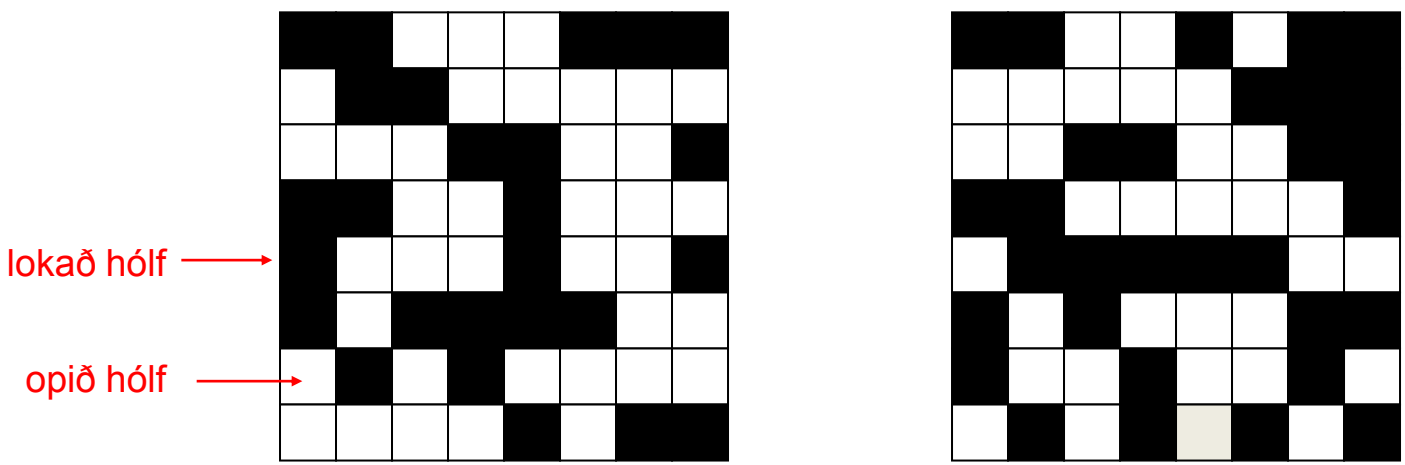




# Líkan fyrir síun

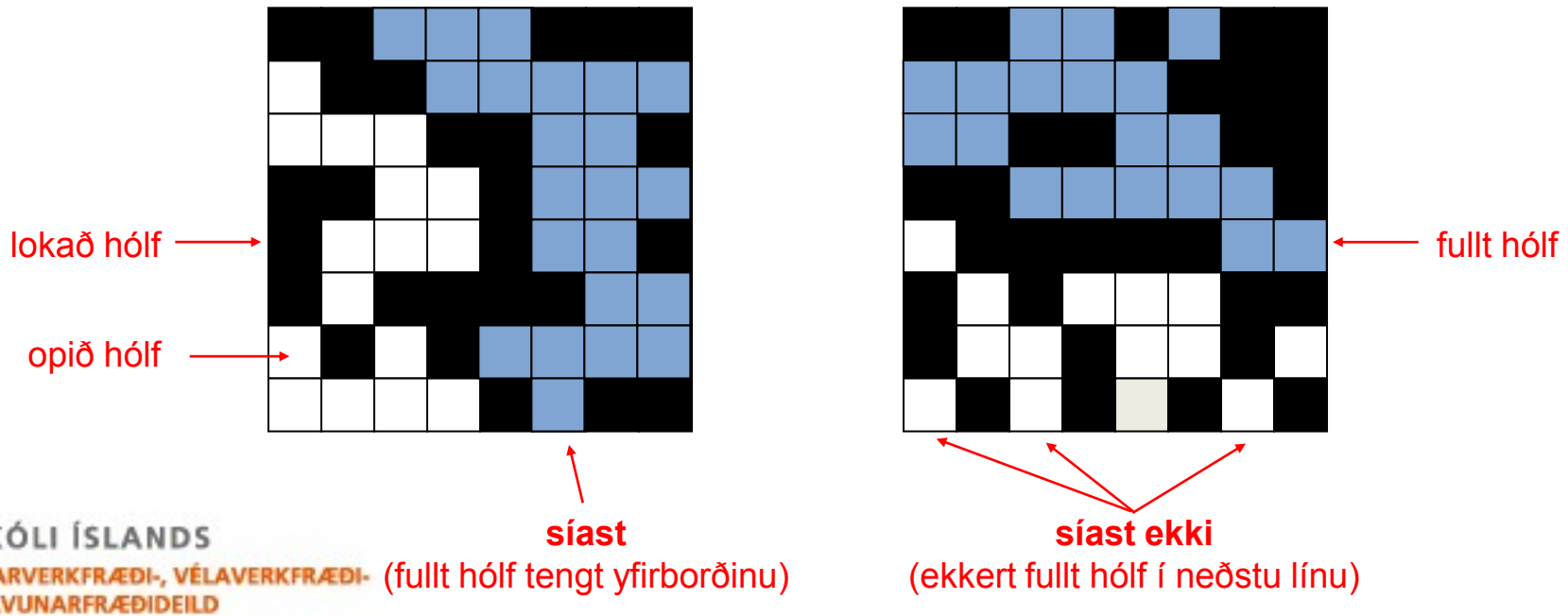
- Búum til einfalt líkan til að herma síun
  - *N*x*N* grind af hólfum
  - Hvert hólf er annaðhvort lokað (*blocked*) eða opið (*open*)

Bara tvívítt líkan, þrívítt væri raunhæfara



# Líkan fyrir síun

- Búum til einfalt líkan til að herma síun
  - *N*x*N* grind af hólfum
  - Hvert hólf er annaðhvort lokað (*blocked*) eða opið (*open*)
  - Hólf er fullt (*full*) ef það er tengt yfirborðinu í gegnum opin hólf



# Fræðileg spurning

- Fyrir *N*x*N* kerfi þar sem hvert hólf er opið með líkindum *p*, hverjar eru líkurnar á því að kerfið síist?

p = 0.3 (síast ekki) | p = 0.4 (síast ekki) | p = 0.5 (síast ekki) | p = 0.6 (síast) | p = 0.7 (síast)

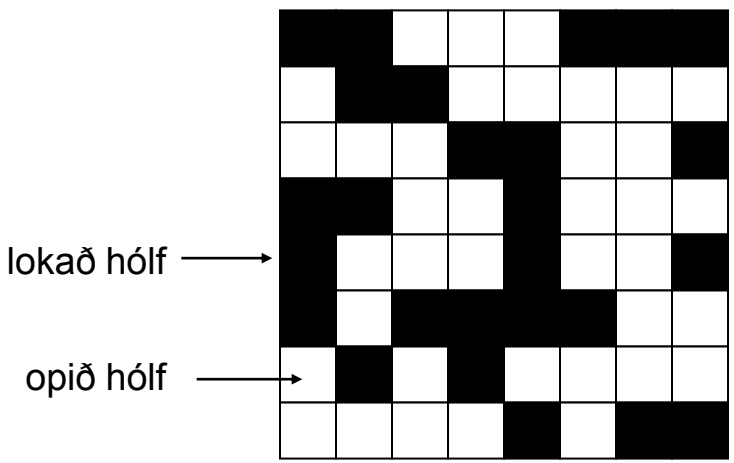
  - Frægt opið verkefni í tölfræðilegri eðlisfræði

Ekki til nein lokuð formúla

  - Við notum Monte Carlo hermun til að fá tölfræðilega lausn

# Framsetning gagnanna

- Notum rökfylki (*boolean array*) til að tákna þau hólf sem eru opin (`true` = opin, `false` = lokuð)
  - Notum svo annað rökfylki til að geyma hvaða hólf eru full
  - Notum `stdArrayIo` til að lesa og skrifa út fylkin



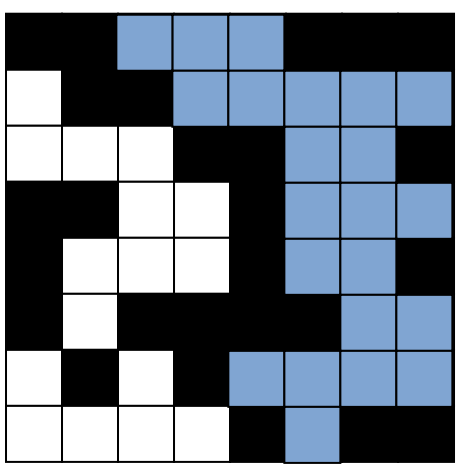

```
8 8
0 0 1 1 1 0 0 0
1 0 0 1 1 1 1 1
1 1 1 0 0 1 1 0
0 0 1 1 0 1 1 1
0 1 1 1 0 1 1 0
0 1 0 0 0 0 1 1
1 0 1 0 1 1 1 1
1 1 1 1 0 1 0 0
```

`open[][]`

Í úttakinu notum við 1 fyrir opið hólf og 0 fyrir lokað

# Framsetning gagnanna

- Gætum líka hafa notað heiltölufylki fyrir *N*x*N* grindina
  - Með kóða fyrir stöðu hólfanna (1: opið, 2: lokað, 3: fullt)
  - Ekki eins almenn lausn



fullt hólf

```
8 8
0 0 1 1 1 0 0 0
0 0 0 1 1 1 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 0 1 1
0 0 0 0 1 1 1 1
0 0 0 0 0 1 0 0
```

`full[][]`

Fylkið fyrir full hólf

# Upprifjun: `StdArrayIO.java`

```
public class StdArrayIO {
   ...
   // read M-by-N boolean matrix from standard input
   public static boolean[][] readBoolean2D() {
      int M = StdIn.readInt(); int N = StdIn.readInt();
      boolean[][] a = new boolean[M][N];
      for (int i = 0; i < M; i++)
         for (int j = 0; j < N; j++)
            if (StdIn.readInt() != 0) a[i][j] = true;
      return a;
   }

   // print boolean matrix to standard output
   public static void print(boolean[][] a) {
      int M = a.length;
      int N = a[0].length;
      StdOut.println(M + " " + N);
      for (int i = 0; i < a.length; i++) {
         for (int j = 0; j < a[i].length; j++) {
            if (a[i][j]) StdOut.print("1 ");
            else         StdOut.print("0 ");
         }
         StdOut.println();
      }
   }
}
```

Notum fallið `readBoolean2D` til að lesa líkan af staðalinntaki

Lesa fyrst víddirnar

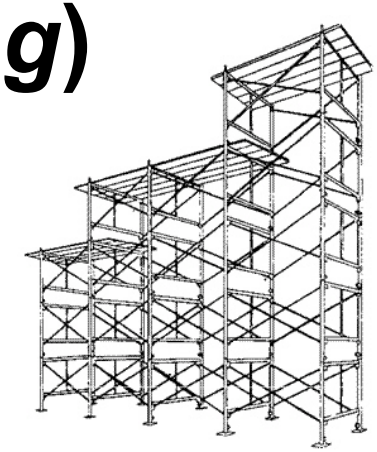
Ef gildið er ekki 0 þá er stak fylkisins `true`

Yfirhlaðin útgáfa af `print` fallinu til að skrifa út `boolean` fylki

Skrifa út víddirnar

Ef gildið er ekki 0 þá er stak fylkisins `true`

# Reisum vinnupalla (*scaffolding*)



- Taka ákvörðun um framsetningu gagna
- Skipuleggja beinagrind forritsins:

Aðalforrit (`main`):

- Lesa inn líkan — Gerum þetta með `stdArrayIO`
- Reikna flæðið — Erfiðasti hlutinn, geymum!
- Athuga hvort líkanið síist — Ath. hvort fullt hólf í neðstu línu
- Skrifa út niðurstöðu

- Byrjum á forriti sem vinnur með líkön úr skrá
  - Auðveldara að finna og laga villur
- Búum síðan til slembilíkön
  - Líkur á opnu hólfi eru *p*

# Kóði fyrir vinnupalla

```
public class Percolation {

   // return boolean matrix representing full sites
   public static boolean[][] flow(boolean[][] open)

   // does the system percolate?
   public static boolean percolates(boolean[][] open) {
      int N = open.length;
      boolean[][] full = flow(open);
      for (int j = 0; j < N; j++)
         if (full[N-1][j]) return true;
      return false;
   }

   // test client
   public static void main(String[] args) {
      boolean[][] open = StdArrayIO.readBoolean2D();
      StdArrayIO.print(flow(open));
      StdOut.println(percolates(open));
   }
}
```

Eigum eftir að skrifa þetta fall!

Ef eitthvert hólf í neðstu línu er fullt þá síast líkanið

**Aðalforritið:**
Les inn fylki
Reikna flæði
Ath. hvort síast

# Fyrirlestraræfing

1. Hvernig breytist fylkið `b = {1, 2, 3, 4}`, ef það er sent niður í eftirfarandi fall:

```
public static void leyndo(int[] a) {
   for (int i=1; i<a.length; i++)
      a[i] = a[i] + a[i-1];
}
```

2. Stingið uppá öðrum notkunardæmum fyrir síun
   Vísbending: Vírus, skógareldur, fatnaður, völundarhús

3. Síast líkanið líkanið hér til hliðar?
   Ath.: Getum ekki farið á ská, bara niður og til hliðar

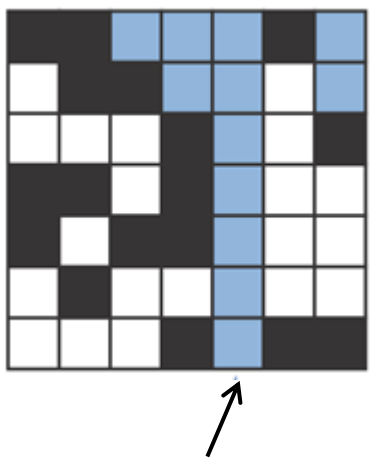
```
8 8
0 0 1 1 1 0 0 0
1 0 0 1 1 1 1 1
1 1 1 0 0 1 1 0
0 0 1 1 0 1 1 1
0 1 1 1 0 1 1 0
0 1 0 0 1 0 0 1
1 0 1 0 1 1 1 1
1 1 1 1 0 1 0 0
```

HÁSKÓLI ÍSLANDS
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI- OG TÖLVUNARFRÆÐIDEILD

# Lóðrétt síun (*vertical percolation*)

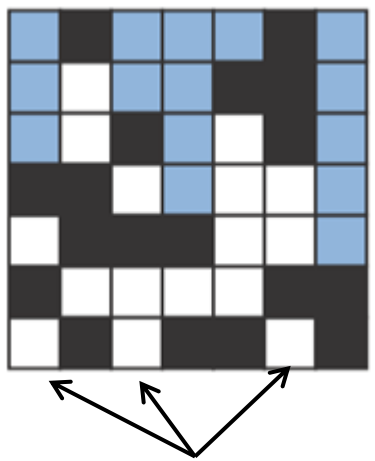
- Byrjum á að leysa auðveldari útgáfu af verkefninu:
  - Er til leið frá yfirborðinu niður í neðstu línu sem fer beint niður (þ.e. lóðrétt)?

Síast lóðrétt



Hólf tengt yfirborði með lóðréttri slóð

Síast ekki lóðrétt



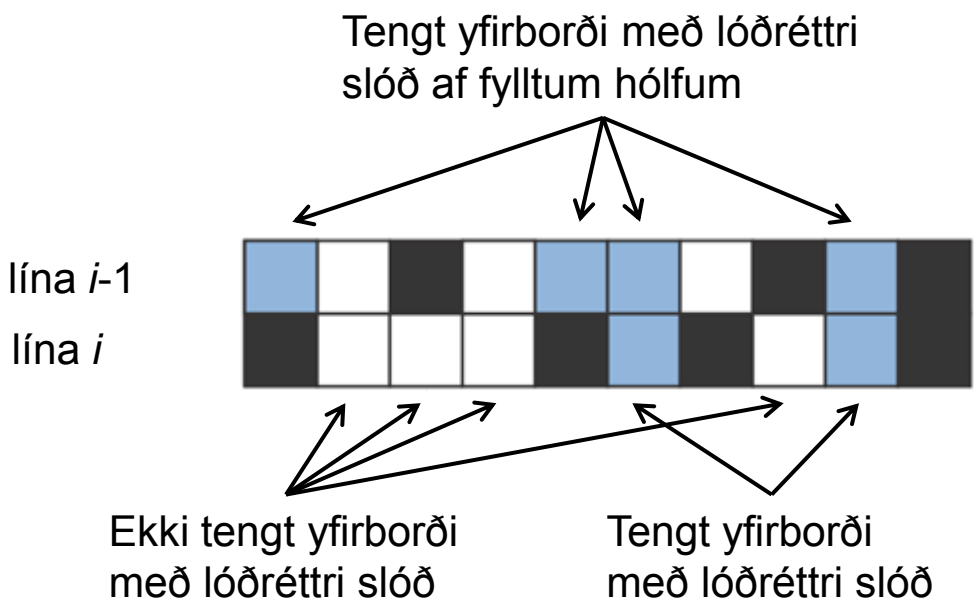
Ekkert opið hólf tengt yfirborði með lóðréttri slóð

HÁSKÓLI ÍSLANDS
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI- OG TÖLVUNARFRÆÐIDEILD

# Lóðrétt síun

- Er hólf (*i*, *j*) fullt?
- Það er fullt ef (*i*, *j*) er opið og hólf (*i*-1, *j*) er fullt

**Reiknirit:** Skanna línur, frá efstu til neðstu



# Lóðrétt síun - Java kóði

Fallið sem reiknar flæðið

```
public static boolean[][] flow(boolean[][] open) {

    int N = open.length;
    boolean[][] full = new boolean[N][N];
    for (int j = 0; j < N; j++)
       full[0][j] = open[0][j];

    for (int i = 1; i < N; i++)
       for (int j = 0; j < N; j++)
          full[i][j] = open[i][j] && full[i-1][j];

    return full;
}
```

Búa til `full` fylkið og upphafsstilla fyrstu línuna

Fyrir hinar línurnar:
Ef hólf opið og hólfið fyrir ofan er fullt þá þetta hólf fullt

lína *i*-1

lína *i*

# Lóðrétt síun - prófanir

- Setjum fallið inní "vinnupallakóðann" og prófum fyrir nokkur fylki:

```
% more testT.txt
5 5
0 1 1 0 1
0 0 1 1 1
1 1 0 1 1
1 0 0 0 1
0 1 1 1 1
```

```
% java VerticalPercolation <testT.txt
5 5
0 1 1 0 1
0 0 1 0 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
true
```

Prentum út `full` fylkið

```
% more testF.txt
5 5
1 0 1 0 0
1 0 1 1 1
1 1 1 0 1
1 0 0 0 1
0 0 0 1 1
```

```
% java VerticalPercolation <testF.txt
5 5
1 0 1 0 0
1 0 1 0 0
1 0 1 0 0
1 0 0 0 0
0 0 0 0 0
false
```

# Lóðrétt síun - prófanir

- Bætum við hjálparföllum til að búa til slembilíkön og teikna líkön upp með **`StdDraw`**

```
public class Percolation {

   ...

   public static boolean[][] random(int N, double p) {
      boolean[][] a = new boolean[N][N];
      for (int i = 0; i < N; i++)
         for (int j = 0; j < N; j++)
            a[i][j] = StdRandom.bernoulli(p);
      return a;
   }

   public static void show(boolean[][] a, boolean foreground)

}
```

Skilar rökfylki, hvert hólf `true` með líkum *p*

Teiknar rökfylki á staðalteikningu

# Gagnabirting (*data visualization*)

- Mikilvægt að geta séð gögnin á grafískan hátt
  - Búum til annað forrit **`Visualize.java`**, sem býr til slembifylki, framkvæmir hermun og teiknar líkanið upp grafískt

```
public class Visualize {
   public static void main(String[] args) {
      int    N = Integer.parseInt(args[0]);
      double p = Double.parseDouble(args[1]);
      boolean[][] open = Percolation.random(N, p);
      boolean[][] full = VerticalPercolation.flow(open);
      StdDraw.setPenColor(StdDraw.BLACK);
      Percolation.show(open, false);
      StdDraw.setPenColor(StdDraw.CYAN);
      Percolation.show(full, true);
   }
}
```

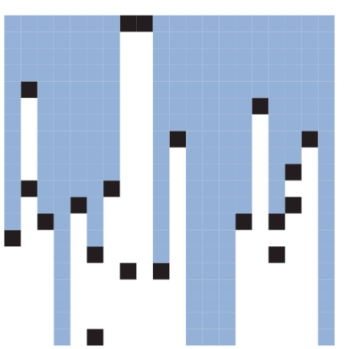
Búa til slembifylki

Reikna flæði

Teikna lokuð hólf sem svört

Teikna full hólf sem blágræn

```
% java Visualize 20 .95 1
```



Fyrsta gildi: Vídd fylkis
Annað gildi: *p* (Líkur á opnu)
Þriðja gildi: Fjöldi fylkja

# Lóðrétt síun - líkindamat

- Getum nú gefið svar við fræðilegu spurningunni:
  - Hverjar eru líkurnar á því að kerfi síist lóðrétt ef líkur á opnu hólfi eru *p*

Þetta er einfaldara en við byrjuðum með

```
public class VerticalEstimate {

   public static double eval(int N, double p, int T) {
      int cnt = 0;
      for (int t = 0; t < T; t++) {
         boolean[][] open = Percolation.random(N, p);
         if (VerticalPercolation.percolates(open)) cnt++;
      }
      return (double) cnt / T;
   }

   public static void main(String[] args) {
      int    N = Integer.parseInt(args[0]);
      double p = Double.parseDouble(args[1]);
      int    T = Integer.parseInt(args[2]);
      StdOut.println(eval(N, p, T));
   }
}
```

Fyrir gefið *N* og *p*, keyra hermun *T* sinnum og skila meðalfjölda síana

Lesa gildi af skipanalínu og kalla á hermifallið

# Niðurstaða

Til lokuð formúla fyrir lóðrétta síun:

$$1-(1-p^N)^N$$

$1 - (1 - 0.7^{20})^{20} \approx 0.015838061...$

$1 - (1 - 0.8^{20})^{20} \approx 0.206993479...$

$1 - (1 - 0.9^{20})^{20} \approx 0.925169717...$

Hermun gefur nokkuð réttar niðurstöður!

Stærð fylkis (*N*), líkur (*p*) og fjöldi hermana (*T*)

```
% java VerticalEstimate 20 0.7 1000000
0.015867

% java VerticalEstimate 20 0.8 1000000
0.206646

% java VerticalEstimate 20 0.9 1000000
0.925489

% java VerticalEstimate 40 0.9 1000000
0.448927
```

Hver keyrsla: ~30 sek.

Hver keyrsla: ~2 mín.

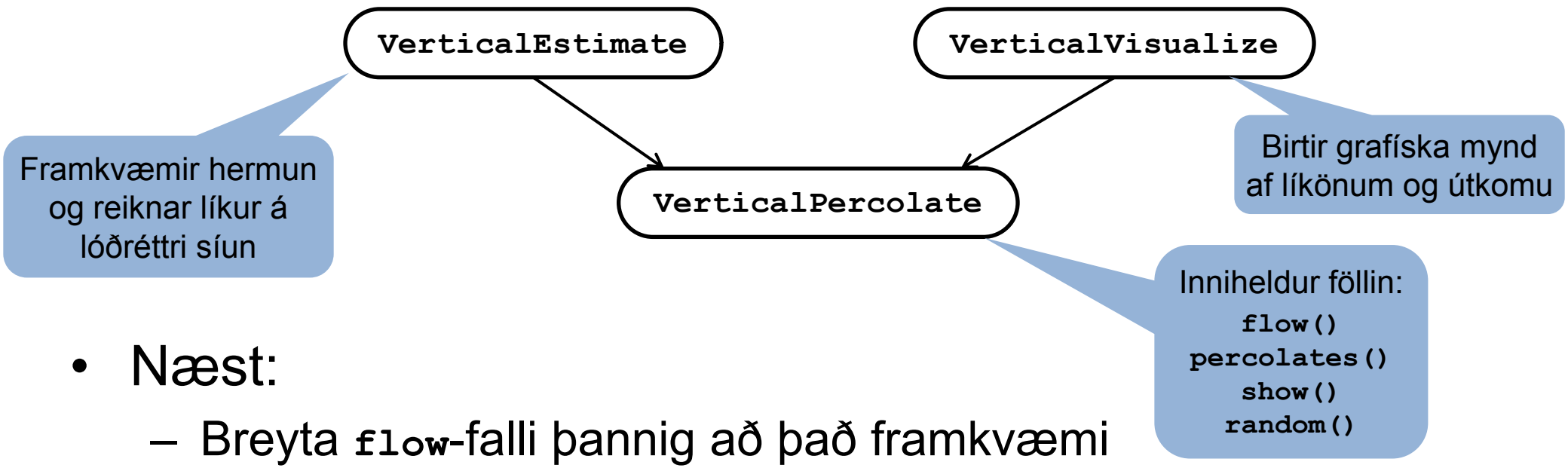
**Keyrslutími:** Vex í hlutfalli við $TN^2$

**Minnisnotkun:** Vex í hlutfalli við $N^2$

HÁSKÓLI ÍSLANDS
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI- OG TÖLVUNARFRÆÐIDEILD

# Samantekt

- Höfum smíðað forrit til að herma einfalda útgáfu af síun




- Næst:
  - Breyta `flow`-falli þannig að það framkvæmi almenna síun

# Fyrirlestraræfing

4. Skrifið fallið **`isEven(int[] a)`**, sem skilar **`true`** ef öll stökin í **`a`** eru jafnar tölur, en **`false`** ef einhver þeirra er oddatala

5. Hvers vegna upphafsstillum við fyrstu línuna í **`full`** svona í **`flow`**-fallinu?

```
for (int j = 0; j < N; j++)
  full[0][j] = open[0][j];
```

6. Stingið uppá notkunardæmum fyrir lóðrétta síun (*vertical percolation*)

# Samantekt

- Í þessum tíma:
  - Sýnidæmi: Síun (*percolation*) **Kafli 2.4**
  - Stórt forrit fyrir alvöru verkefni
- Í næsta tíma:
  - Almenn síun
  - Önnur áhugaverð forrit **Kafli 2.4**