Presentation av obligatoriska uppgift 5
trafiksimulering

Ett större program med flera klasser

Hur man designar ett system

Hur man gör simuleringar

Korsningen Dag hammarsköldsväg och Vårdsätravägen/Kungsängsleden

Dag Hammarsköldsväg

Polacksbacken

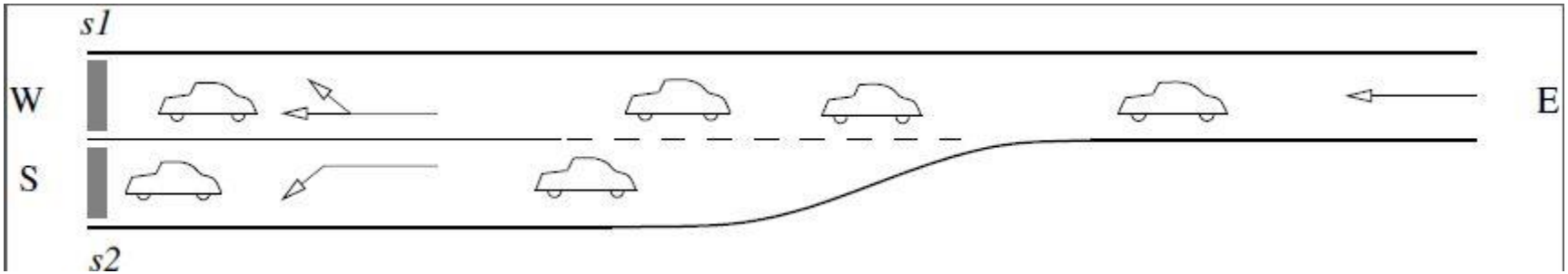
Kungsängsleden

Vårdsätravägen

Korsningen Dag Hammarsköldsväg och Vårdsätravägen/Kungsängsleden

Korsningen är kontrollerad av ljussignaler (s1,s2,s3,s4,s5,s6)

Uppgiften består i att simulera en förenklad korsning!



- Fordon (föds) inkommer från punkten E till höger.
- Tre filer: en gemensam fil, en W-fil, en S-fil
- Korsningen är kontrollerad av två ljussignaler (s1,s2)

Vi kommer att ta upp följande...

Hur gör man en simulering?

Vilka förenklingar av verkligheten gör man i simuleringen?

Vilka klasser behövs?

Hur skall (trafik)filerna kopplas ihop?
Hur kopplas signaler och (trafik)filer?

Vilka indata behövs? Vem hanterar indata?

Hur går tidsstegningen till? Vilka objekt bör känna till tiden?

Var samlas statistiken? Vad skall skrivas ut, när och av vem?

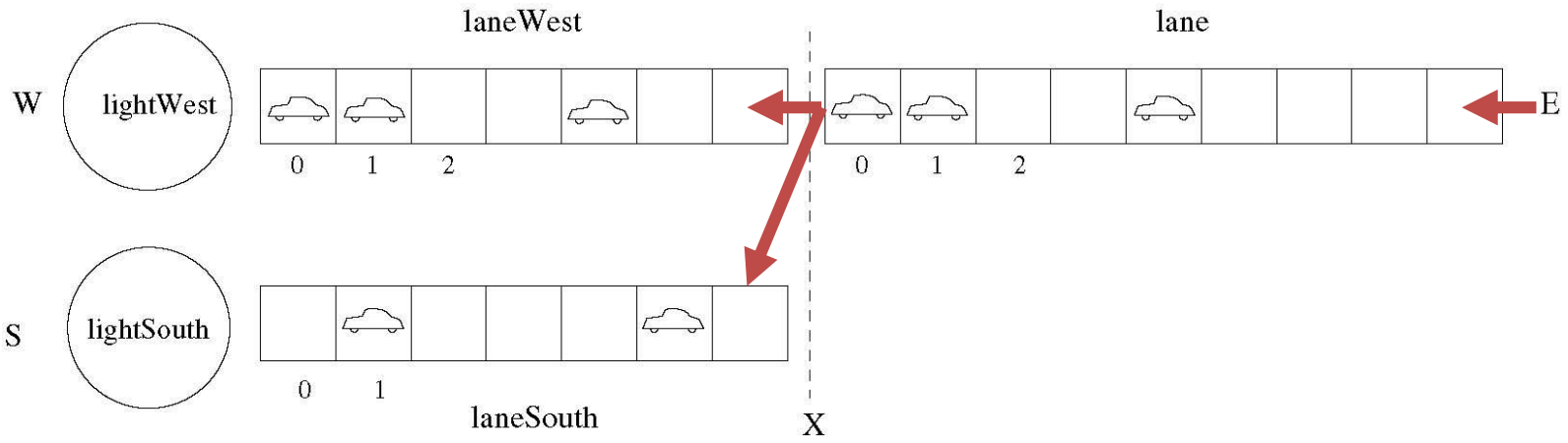
Datormodell

Ett trafiksystem ska byggas upp av två generella komponenter: trafikljus (klassen Light) och körfält (klassen Lane). Det innehåller också ett varierande antal fordonsobjekt (klassen Vehicle).

Vi kommer också behöva en kö, men en sådan representeras enkelt med en lista. Vidare används klassen Destinations för att simulera ankomster av fordon till systemet.

- Klassen Destinations får du färdig
- Klassen Vehicle får du också färdig
- Klassen Lane får du närapå färdig, en metod skall modifieras.
- Klassen Light skall du skriva.

Implementation av den förenklade korsningen (*trafiksystemet*)



Det behövs tre st filer och två st signaler för denna implementation

- Fordon kommer in i korsningen i en fil `lane` vid E.
- Fordon med destination W kör in i fil `laneWest`
- Fordon med destination S kör in i fil `laneSouth`

Notera att filerna består av diskreta positioner, i varje sådan position finns ett fordon eller är tomt.

Klasser som behövs i trafiksystemet

Vilka olika typer av objekt kan utkristalleras?

Jo, följande delar behövs i trafiksystemet:

- Vehicle: Representerar ett fordon
- Light: Representerar en trafiksignal
- Lane: Representerar en fil

Dessutom behövs en klass som utgör själva trafiksystemet som består av ovan tre delar (klasser).

- TrafficSystem:
  - Tre Lane-listor med element av typen Vehicle
  - Två st Light-objekt

    + en kö av inkommande fordon vid E
    + variabler som samlar på sig data som man kan beräkna statistik från

Hur gör man simuleringen?

Jo, allt måste styras av en global ”klocka”. Denna tickar ett tidssteg i taget. Programmet har en variabel `t` för detta.

Vid starten för simuleringen (t=0) har hela trafiksystemet initiala värden: Inga fordon finns i systemet, trafikljusen visar grönt.

I nästa steg, t=1 skall trafiksystemet ”stegas”, dvs trafikljusen skall också stegas i deras repetitiva beteende (och kanske slå om ljuset). Dessutom skall fordon i filerna röra sig framåt ett steg framåt (om möjligt) och det skall kontrolleras om ett nytt fordon kommer in (föds) vid punkten E. När fordonet föds måste vi se till att fordonet kommer ihåg tiden och dess destination.

Om det i ett givet tidssteg var grönt ljus och ett fordon lämnat systemet så skall det bokföras (dvs aktuella tidsvärdet `t`), man kan därmed beräkna hur lång tid det tog för fordonet att passera korsningen (t minus tiden när fordonet ”föddes”)

Tidstegningen kan göras hur många ggr som helst

Algoritm:

```
Skapa komponenterna som utgör trafiksystemet
t = 0
while True:
    t = t + 1
    uppdatera alla komponenter:
        (filer, trafikljus, nytt fordon?)
    presentera tillståndet (eventuellt)
```

Den givna klassen `Destinations`

Denna klass används för att simulera ankomster av nya fordon till trafiksystemet. Varje anrop till dess **`step`**-metod returnerar antingen `None`, som anger att inget fordon kom i detta steg, eller en destination (`'W'` eller `'S'`).

Koden finns i filen fDestinations.py

I denna fil finns även kod för att testa klassen.

Vi kikar på koden och testkör den.

Klassen skall ej ändras.

Den givna klassen `Vehicle`

Klassen **`Vehicle`** är ganska enkel. Dess uppgift är bara att hålla reda på när fordonet skapades och vart det ska.

Ett fordon behöver alltså inte veta var det är, inte titta på signaler och inte kunna se andra fordon. Fordonen kan betraktas som pjäser som flyttas av "någon annan".

Koden (definitionen) finns i samma fil som klassen Lane.

Klassen skall ej ändras.

Den givna klassen `Lane`

En fil är en struktur som innehåller ett antal fordon och ett antal tomrum. Fordonen kan bara komma in i början och lämna i slutet av filen d.v.s. filen är som ett "rör".

Vi kikar på en diskussion och specifikation av klassen från lektion 10.

Klassen **`Lane`** är given, bortsett från en metod `number_in_lane` som skall skrivas klart. I övrigt skall den inte ändras.

Koden kan hämtas här: fVehicleAndLane_Stud.py

I denna fil finns även klassen **`Vehicle`** samt kod för att testa dessa båda klasser.
Vi kikar på koden och testkör den.

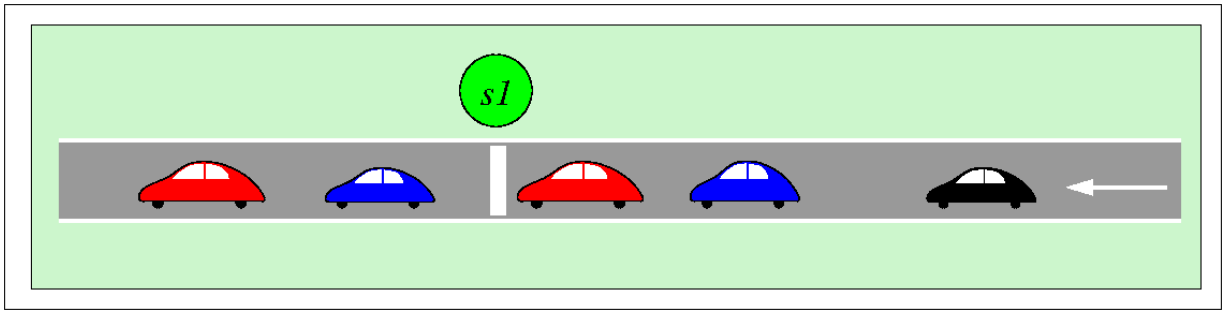
Klassen `Light`

Klassen används för att representera trafiksignaler. Vi kikar på en diskussion och specifikation av klassen från lektion 10.

Filen fLight_Stud.py finns att hämta som innehåller ett minimalt skal till klassen **`Light`** och kod som testar klassen om/när den är färdig.

Klassen skall du skriva färdigt så att den passar specifikationen och den kod som testar den.

Det första trafiksystemet, `TrafficSystem1`



Det finns ett givet program **`fTrafficSystem1_Stud`** som sköter om (driver) simuleringen och tidsstegningen av detta trafiksystem. Finns att ladda ned från kursens hemsida: fTrafficSystem1_Stud.py

Programmet innehåller:

- En klass **`TrafficSystem1`**.
- En funktion (`main`) som sköter simuleringen
- En ofullständig metod **`number_in_system`**

Vi kikar på programmet och kör det.

Utskrifter för `fTrafficSystem1.py`

Metoden `snapshot` i klassen **`TrafficSystem1`** skall skriva ut alla de värden som gör att vi kan se vad som händer i systemet. Metoden skall skriva ut trafikljusets värden och filernas värden i varje tidssteg

Exempel på en programkörning med utskrifterna:

```
  0: ( 0)[.....](G)[.....]   []
  1: ( 1) [.....](G)[....S]   []
  2: ( 2) [.....](G)[...SS]   []
  3: ( 3) [.....](G)[..SSS]   []
  4: ( 4) [.....](G)[.SSSS]   []
  5: ( 5) [.....](G)[SSSSS]   []
  6: ( 6) [....S](G)[SSSSW]   []
  7: ( 7) [...SS](G)[SSSWS]   []
  8: ( 8) [..SSS](R)[SSWSS]   []
  9: ( 9) [.SSS.](R)[SSWSS]   ['S']
 10: (10) [SSS..](G)[SSWSS]   ['S', 'W']
 11: (10) [SS..S](G)[SWSSS]   ['W', 'W']
 12: (10) [S..SS](G)[WSSSW]   ['W', 'S']
 tid       West   L  East     Queue
```

Trafikljuset har här perioden 10 och gröntid 8

Testa trafiksystemet genom att ändra trafikljusets rytm.

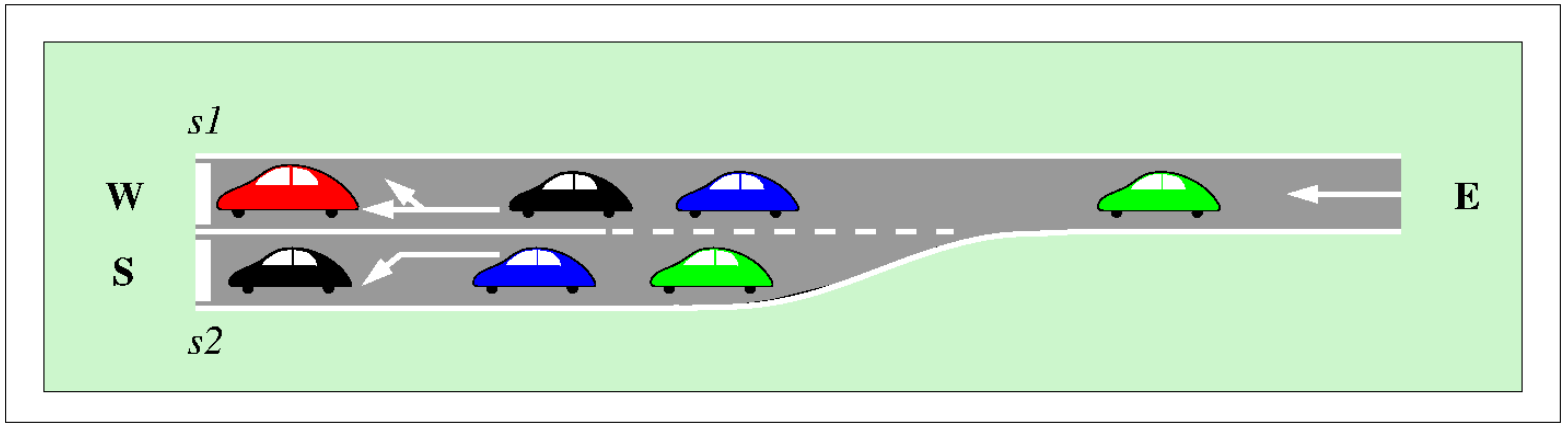
Om det hela tiden är grönt ljus bör fordonen löpa helt fritt genom systemet.

Om det oftare är rött än grönt bör kön av fordon växa kontinuerligt.

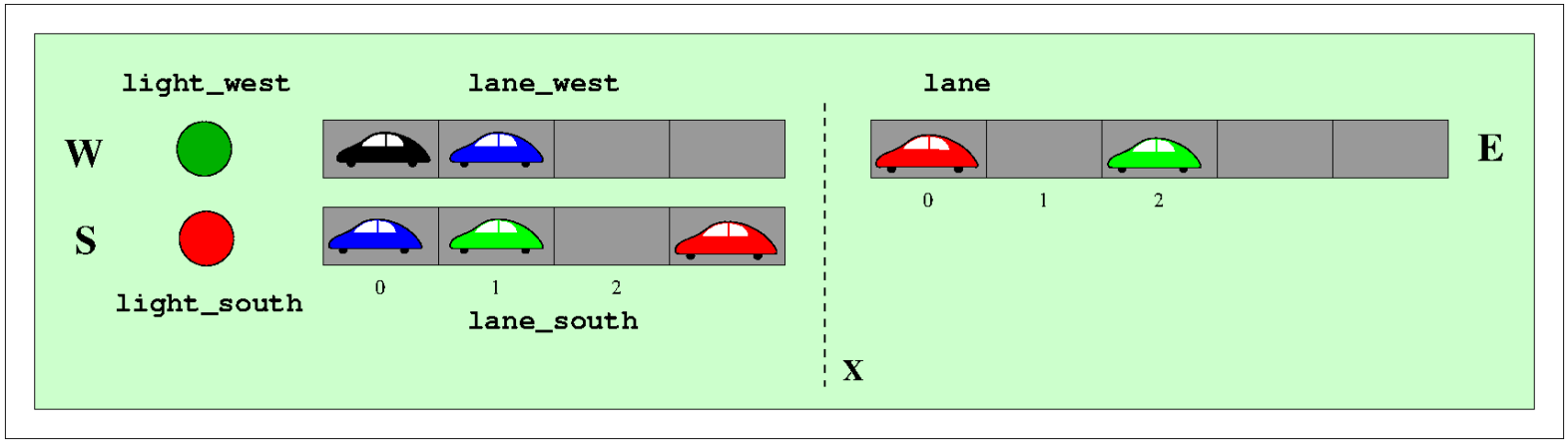
Testa genom att ändra trafikljusets rytm.

Det fullständiga programmet, `fTrafficSystem2.py`

Tips: Börja med göra en kopia av din python-fil `fTrafficSystem1.py` (första systemet). Döp kopian till `fTrafficSystem2.py` eftersom detta program skall simulera det andra systemet. Programmet skall simulera följande scenario.



Trafiksystemet ska således bestå av tre filer (`lane`, `laneWest` och `laneSouth`), trafikljusen `lightWest` och `lightSouth` samt en kö (`Queue`) vid `E`.

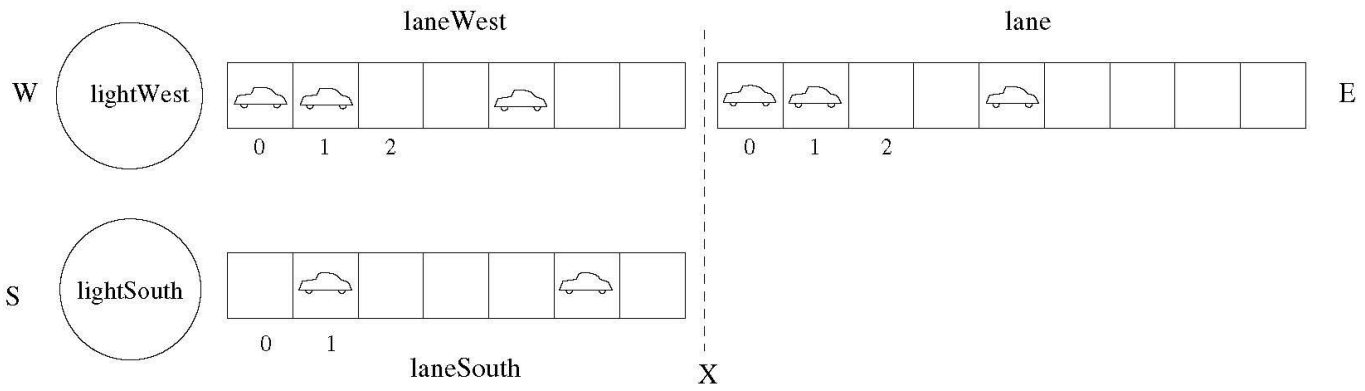


Alla fordon skall först in i filen `lane`, men om den är full skall fordonen placeras i en kö vid E (ej utritad ovan).

Från filen `lane`, skall fordonen placeras i filen `laneSouth` eller `laneWest` beroende på fordonets destinationsvärde S eller W.

Om exvis trafikljuset `lightSouth` visar rött och filen `laneSouth` är full innebär detta att filen `lane` blockeras om ett fordon där i första positionen som har destination S.

Vad som händer vid varje tidssteg, dvs vad metoden `step` i klassen `TrafficSystem2` gör…



`Destinations`

Step → W, S, None

`queue`

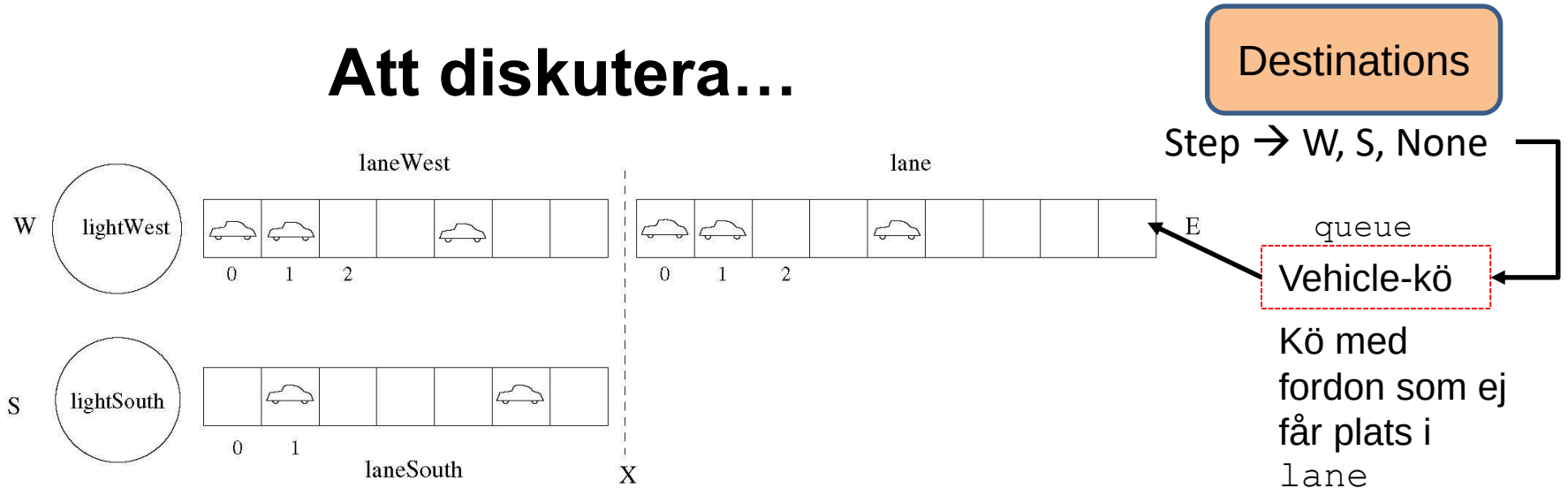
Vehicle-kö

Kö med fordon som ej får plats i `lane`

Tänk på att göra sakerna i en konsekvent ordning:

- Om trafikljus är grönt: Om det finns fordon i första positionen i fil, tag bort detta.
- Stega trafikljusen till nästa läge, metoden `step` för objekten `lightWest, lightSouth`
- Flytta alla fordon framåt i resp fil, metoden `step` för objekten `laneWest, laneSouth`
- Flytta fordon från första pos i `lane` till sista pos i `laneWest` el `laneSouth` (om ledigt)
- Flytta alla fordon framåt i `lane`.
- Om nytt fordon ”föds” av `step`-metoden i Destinations, dvs W el S: Placera fordon sist i kön.
- Flytta fordon först i kön till sist i `lane` (om ledigt)

Att diskutera…



Vet ett fordon om var den befi*n*ner sig?

Nej! Ett Lane-objekt har koll på sina fordon.

TrafikSystem-objektet har koll på sina Lane-objekt.

Kan fordon titta framför sig och ”se” vad som är framför i filen?

Nej! Ett Lane-objekt har koll på sina fordon.

TrafikSystem-objektet har koll på sina Lane-objekt.

Kan ett fordon ”se” trafikljuset?

Nej. TrafikSystem-objektet har koll

Tips: Börja med en förenkling där båda ljusen är gröna hela tiden, så att du ser att det blir ett flöde av fordon genom hela systemet i filerna och utan några som helst hinder.

Statistik som skall beräknas i det andra testsystemet
Data för statistiken fås genom att varje gång ett fordon lämnar korsningen beräknas hur länge fordonet varit i korsning. Bl.a. genomsnittlig tid för fordon och maximal tid för ett fordon att passera korsningen.

Samla tiderna för fordon som lämnar systemet i två lämpliga variabler ett för vardera utgång, för att ta fram statistiken Det innebär att trafiksystemet skall innehålla en instansvariabel som samlar på sig lämpliga data.

Statistik skall beräknas och skrivas ut av metoden `print_statistics` i klassen TrafficSystem2.

- genomsnittliga och maximala tider (antal tidssteg) för fordon att passera ljussignalen `lightW` respektive `lightS`
- andel tidssteg som kön framför vardera signalen varit längre än längden på `laneW` och `laneS` dvs den tid som fildelningen vid X varit blockerad av kö

  (för att beräkna detta behöver man räkna antal tidssteg som det ej gick att lägga ett fordon i `laneW` resp `laneS` därför att sista positionen var upptagen)
- andel tidssteg som det funnits fordon i kön vid entrypunkten (E).

Notera – uppgiften finns beskriven till fullo på

https://www.it.uu.se/edu/course/homepage/prog1/python/vt22/lessons/le10/

Redovisas senast vid labtillfälle L10:5