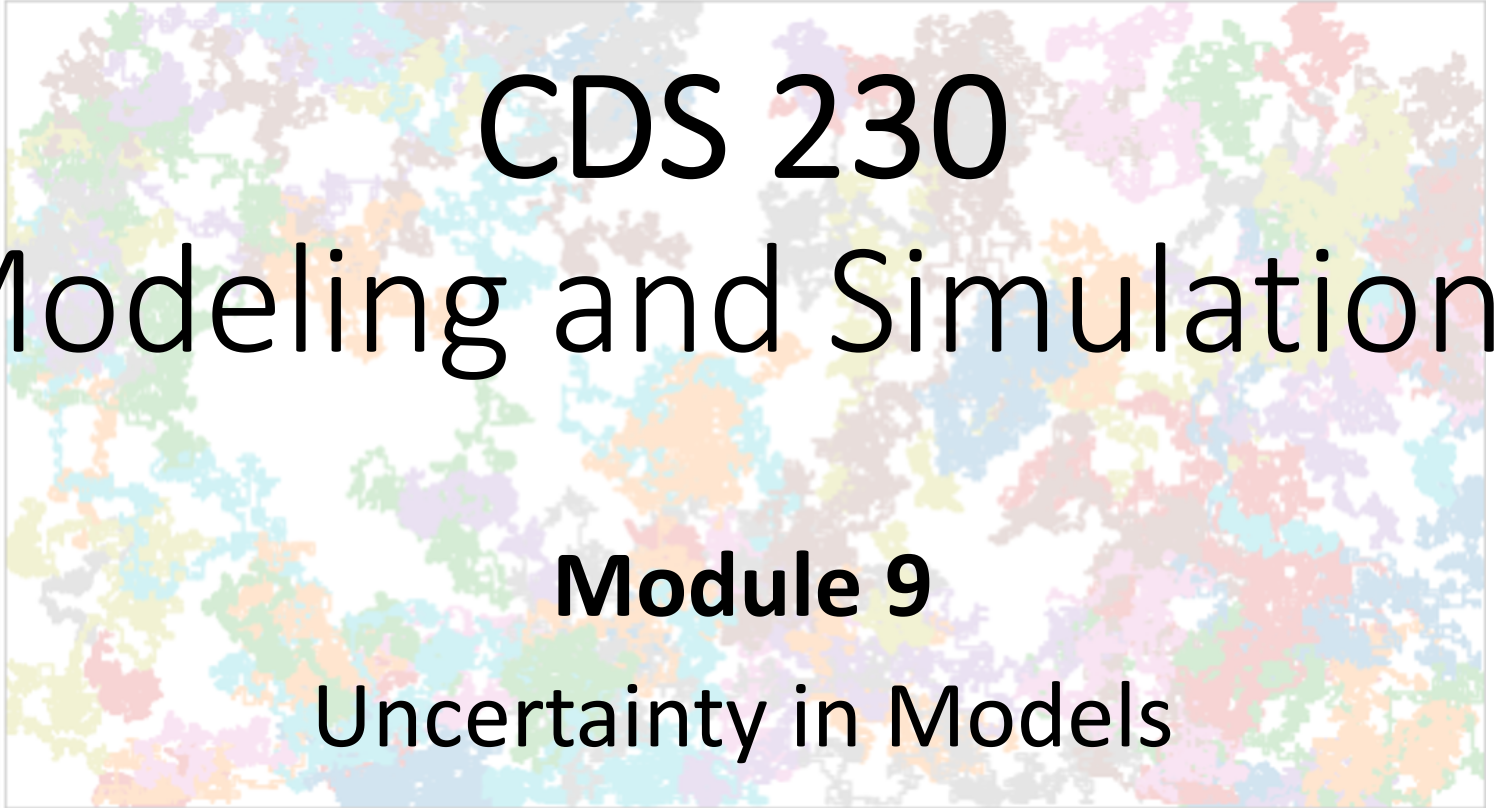


CDS 230
Modeling and Simulation I

Module 9

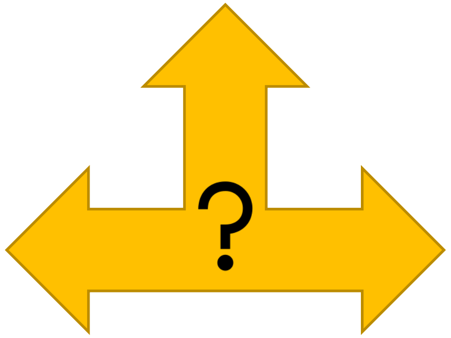
Uncertainty in Models

Dr. Hamdi Kavak
http://www.hamdikavak.com
hkavak@gmu.edu

GEORGE MASON UNIVERSITY

Center for Social Complexity

Uncertainty



- uncertain: ”not known or definite” (Oxford Dictionary)
- Many real-world systems contain uncertainty and variability
  - E.g.: traffic, queues, weather, voting, package delivery, disease spread…
- Can be of varying degrees
- If one or more components of a model has uncertainty, we call such models as **stochastic model** and use a probabilistic framework to describe the component’s behavior.
  - Unlike deterministic models that produce the same results for the same condition

Some sources of uncertainty

- Imperfect knowledge
  - (e.g., small sample in election polls, low resolution)
- Changes in the environment
  - (e.g., weather, human decisions)
- Time dependency
  - (e.g., different traffic patterns at different times, store visits)
- Presence of noise
  - (e.g., measurement precision or accuracy)
- Failure
  - (e.g., power outage, defect)

A small experiment*

- Pick a number: 1, 2, 3, or 4

- Tell me that number when I asked

- Let’s see the results…

*Saw this experiment in Dr. William Kennedy’s lectures.

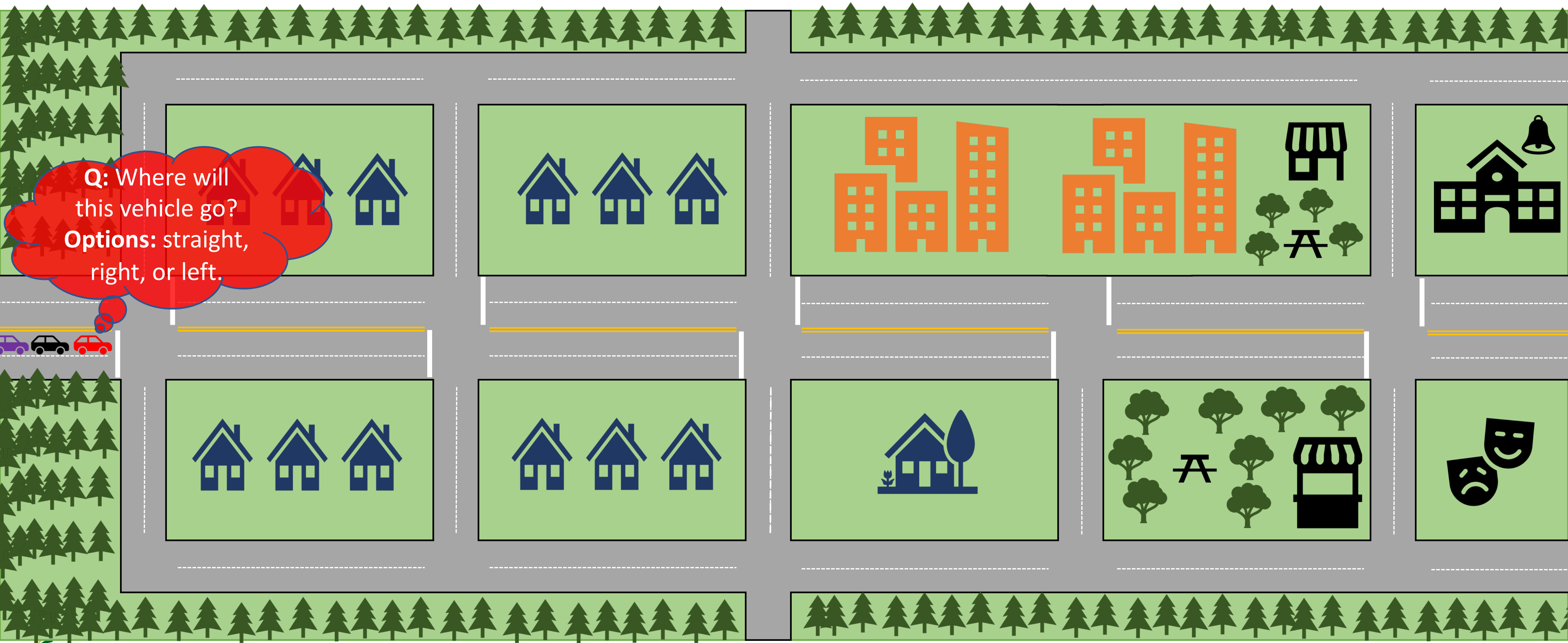
Traffic flow example

Goal: predicting how traffic will flow according to people's individual preferences.

Day 1: Monday – afternoon

Day: Monday

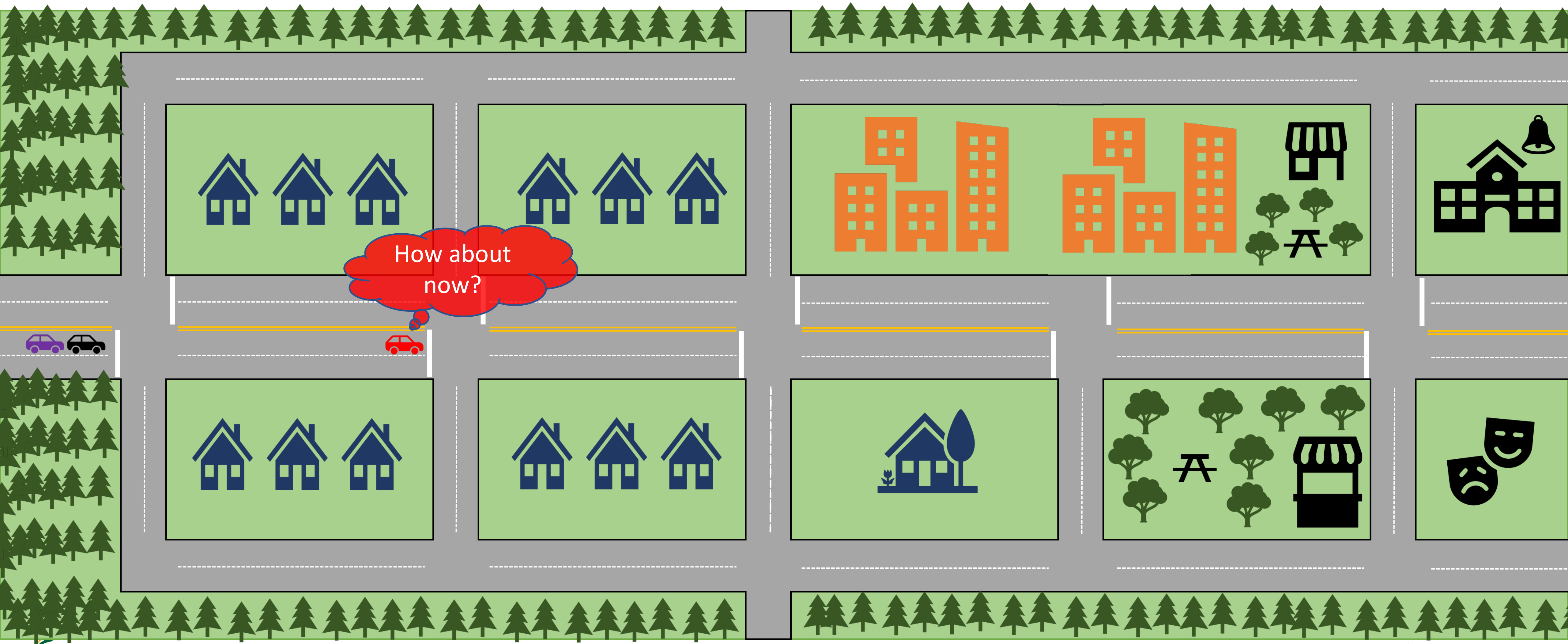
Traffic flow example



Q: Where will this vehicle go? **Options:** straight, right, or left.

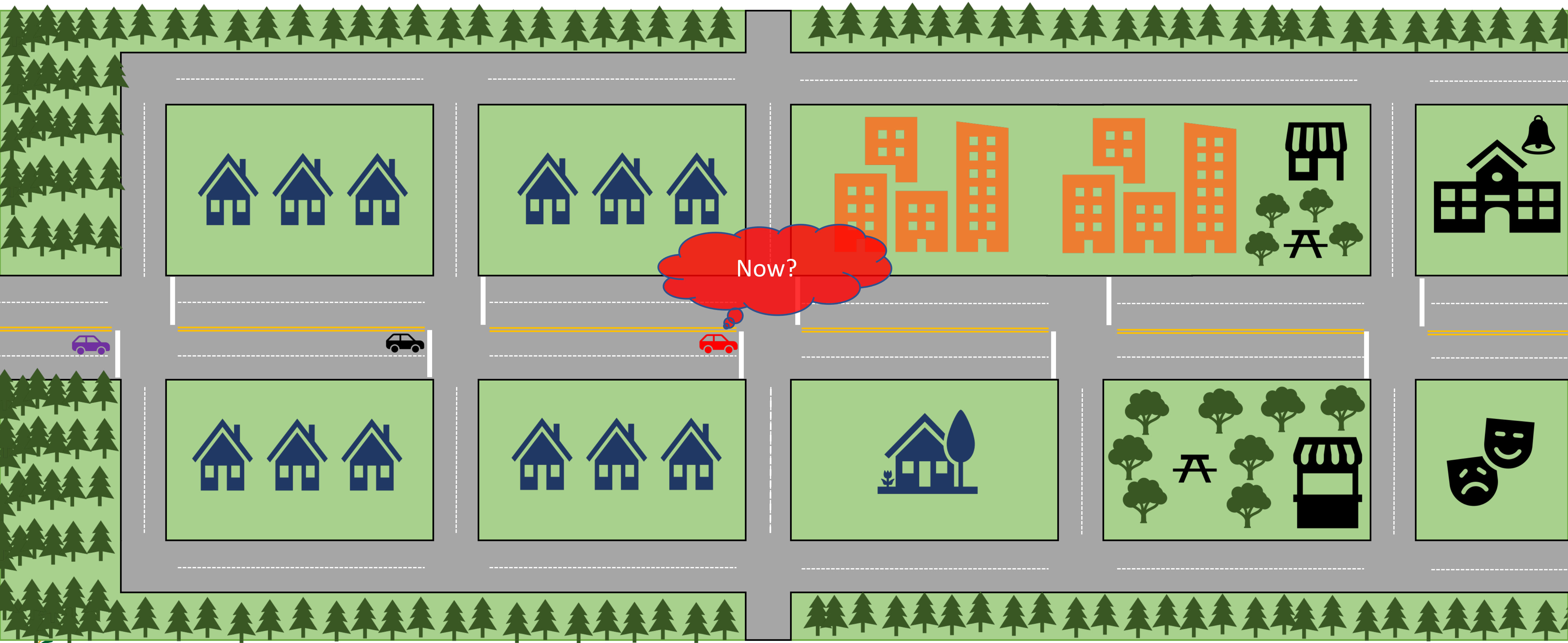
Day: Monday

Traffic flow example



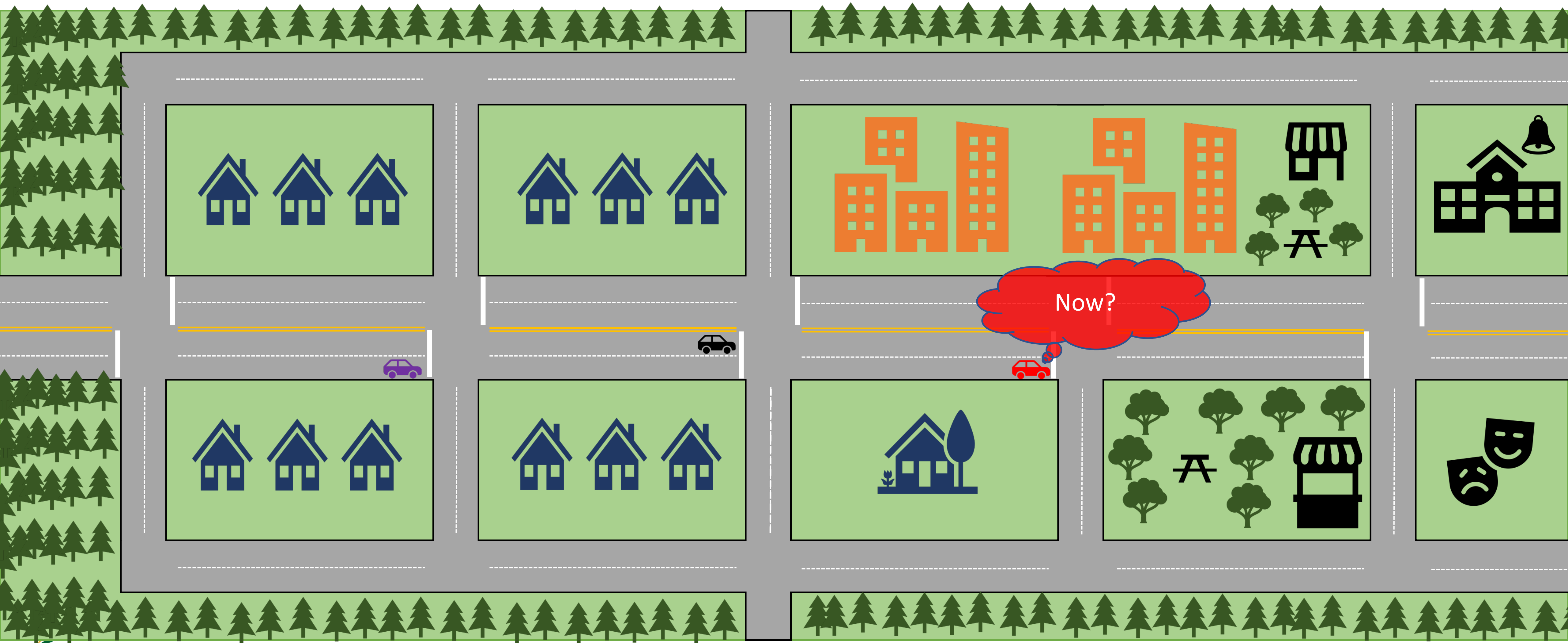
Day: Monday

Traffic flow example



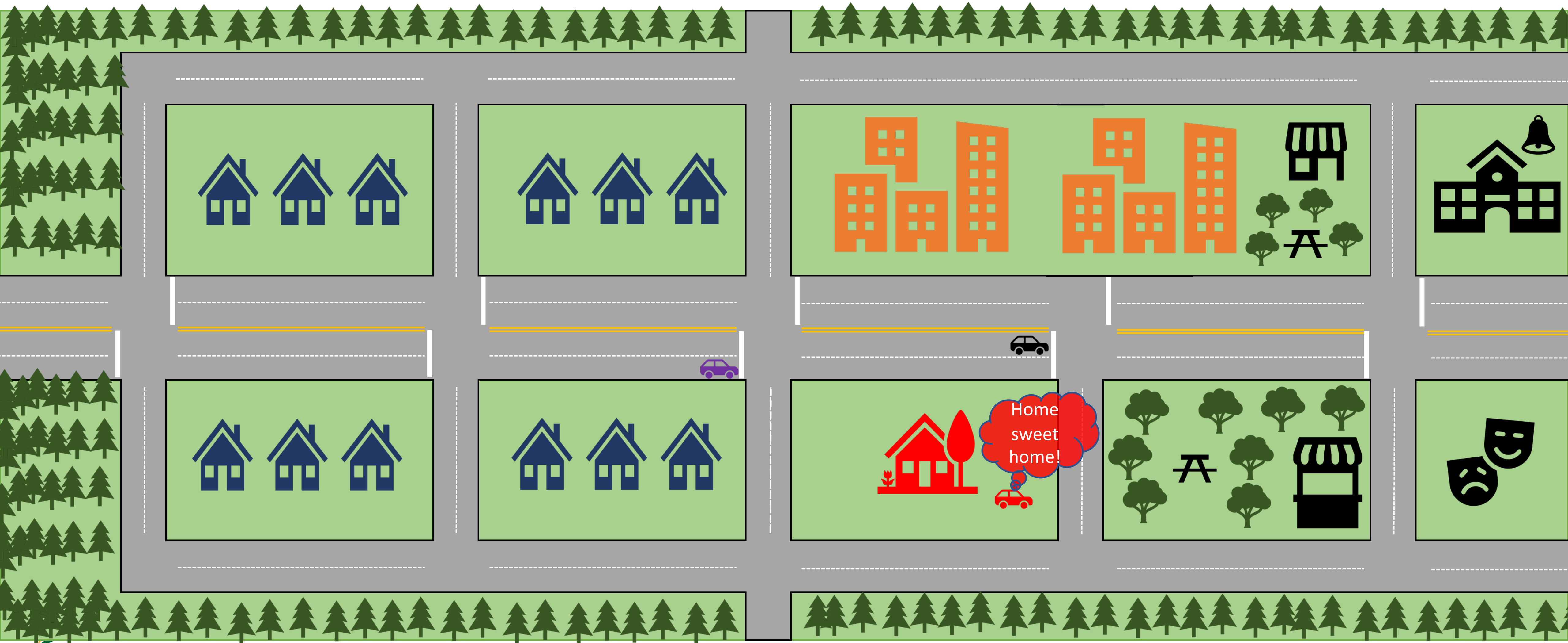
Day: Monday

Traffic flow example



Traffic flow example

Day: Monday
Knowledge: This person arrives home (**red**) in the afternoon.

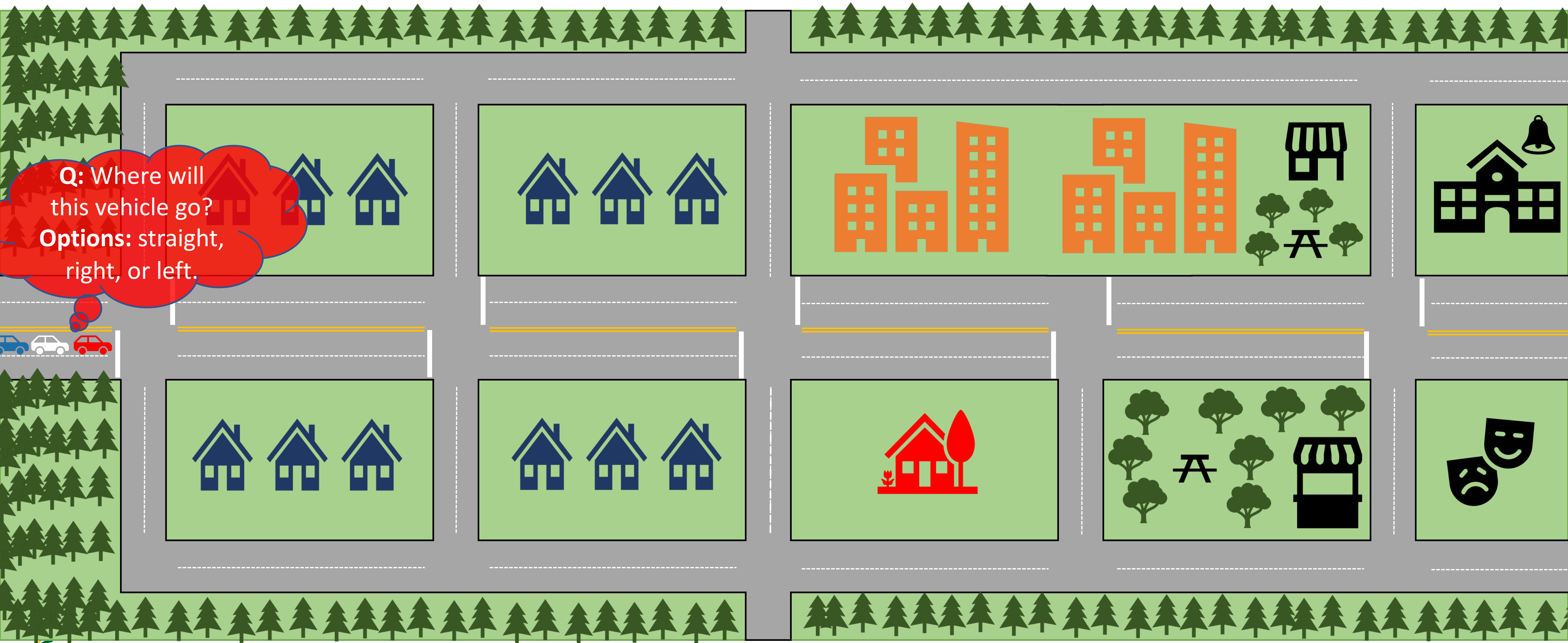


Traffic flow example

Day 2: Tuesday – afternoon

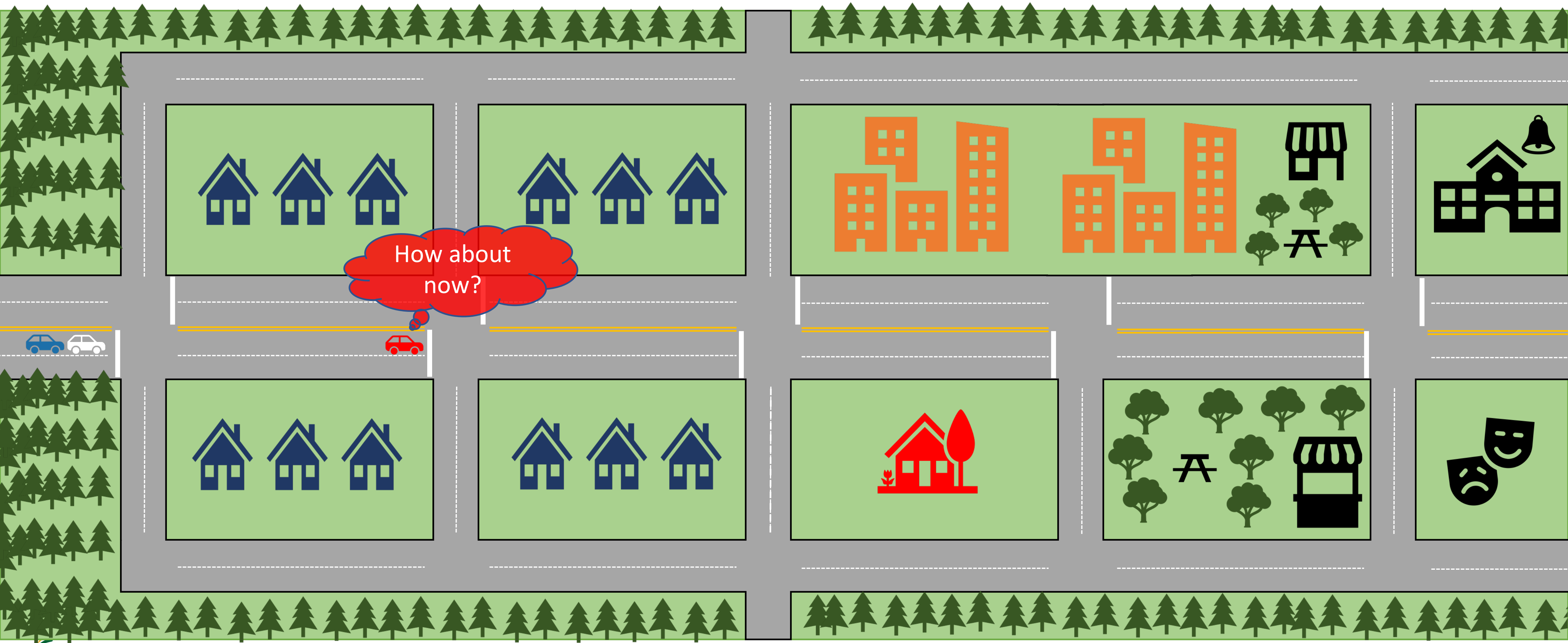
Day: Tuesday

Traffic flow example


Q: Where will this vehicle go?
Options: straight, right, or left.

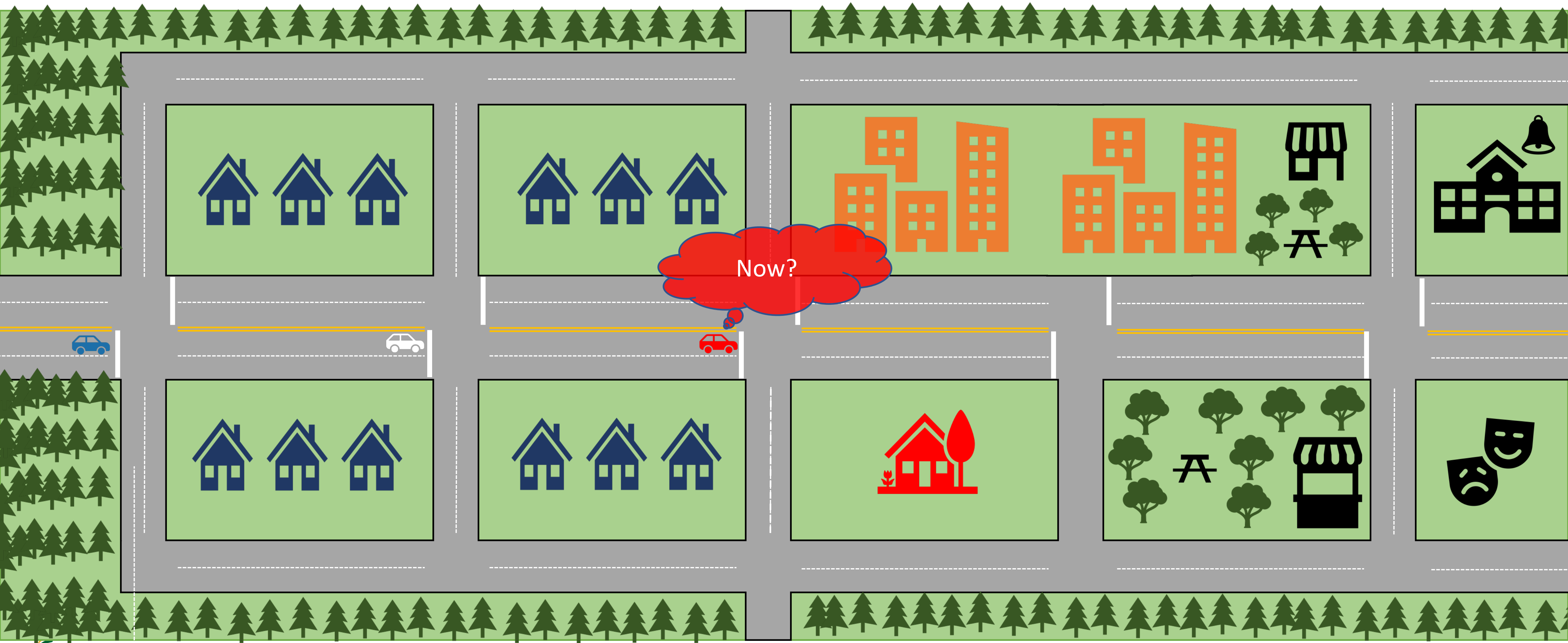
Day: Tuesday

Traffic flow example



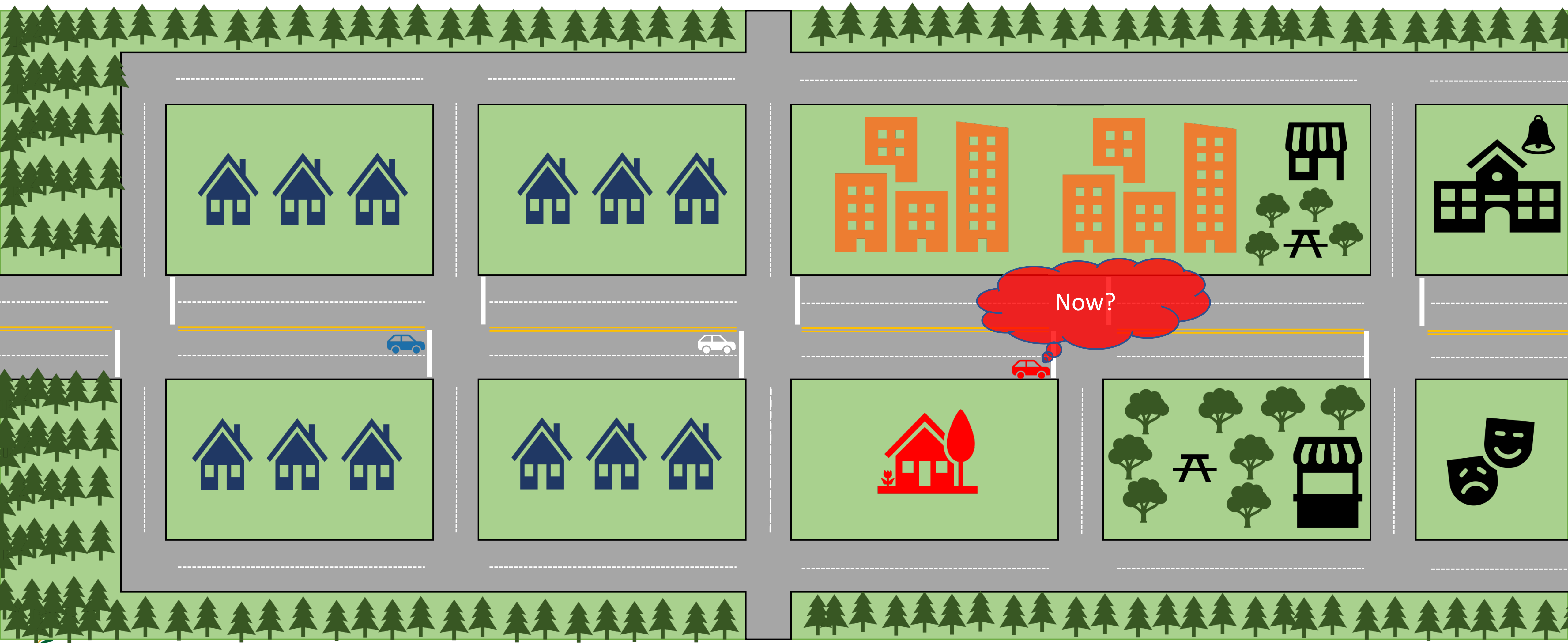
Day: Tuesday

Traffic flow example



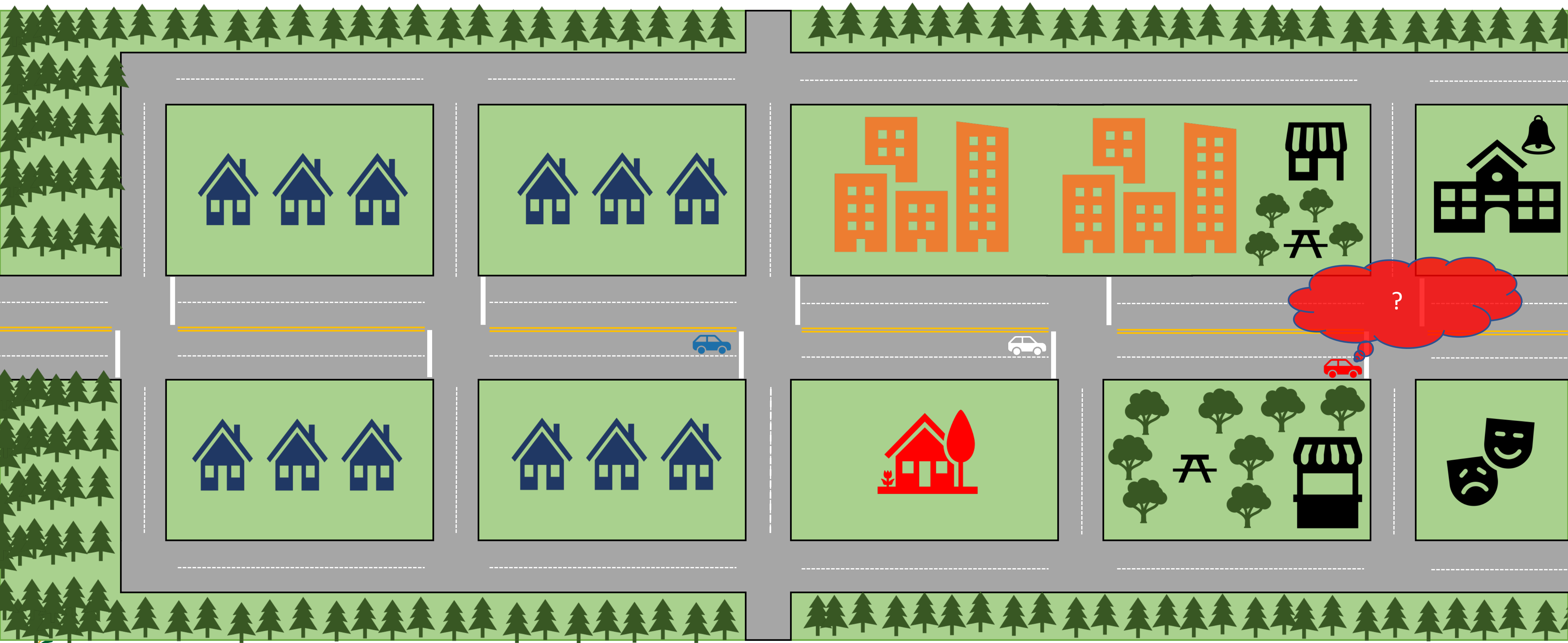
Day: Tuesday

Traffic flow example



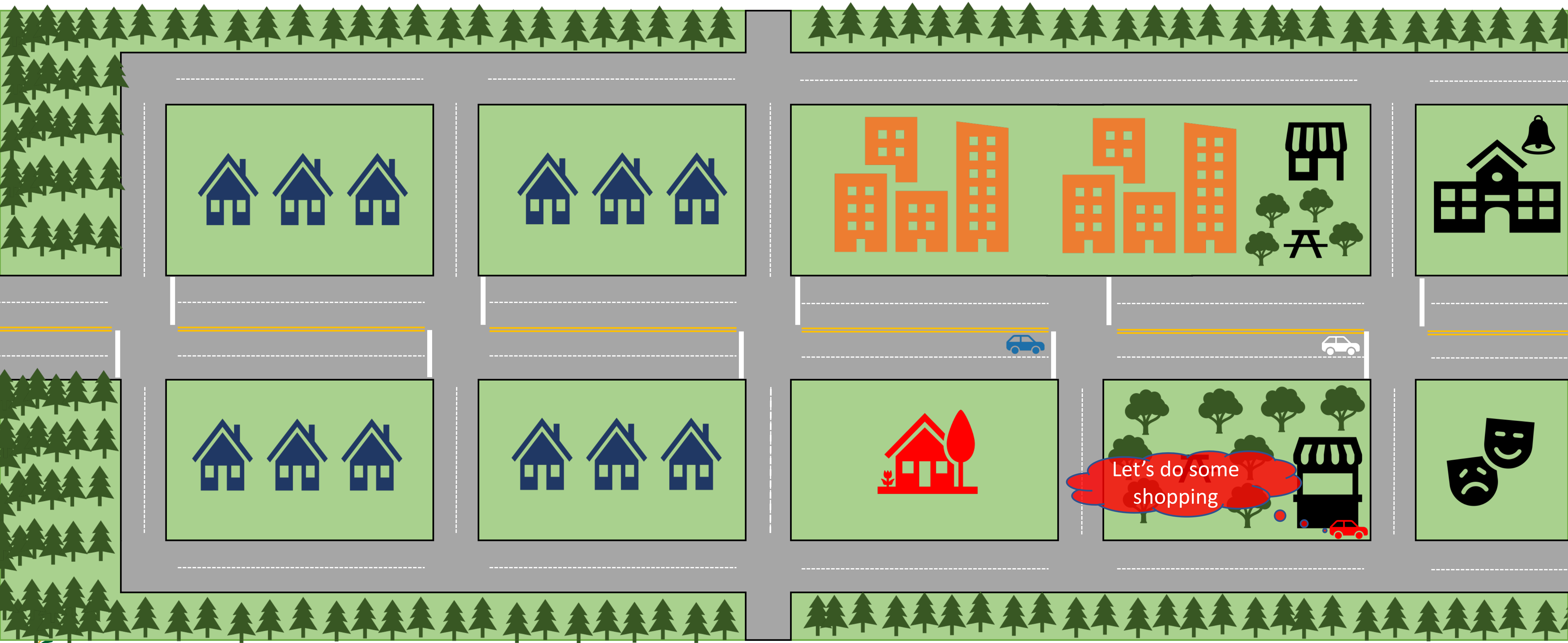
Day: Tuesday

Traffic flow example



Traffic flow example

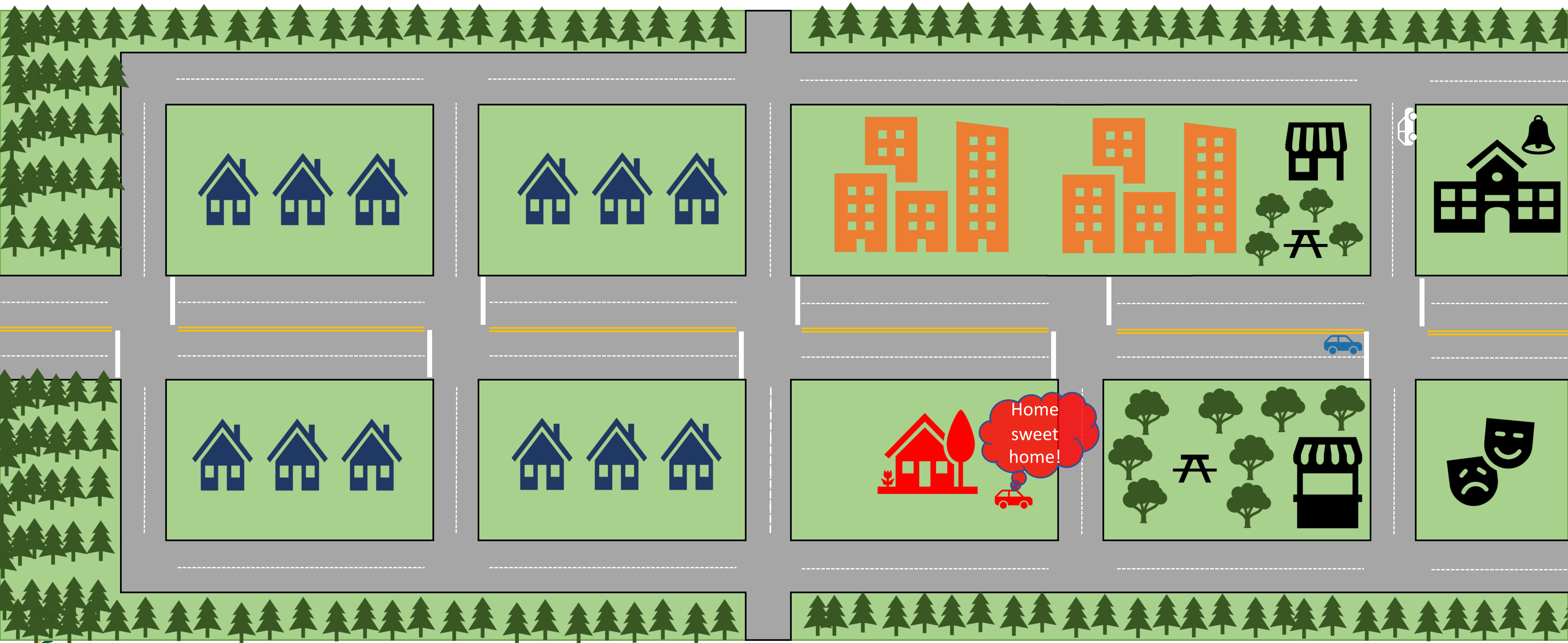
Day: Tuesday
Shopping: Bread



Traffic flow example

Day: Tuesday

Knowledge: This person sometimes stops by the nearest shop before arriving home in the afternoon.

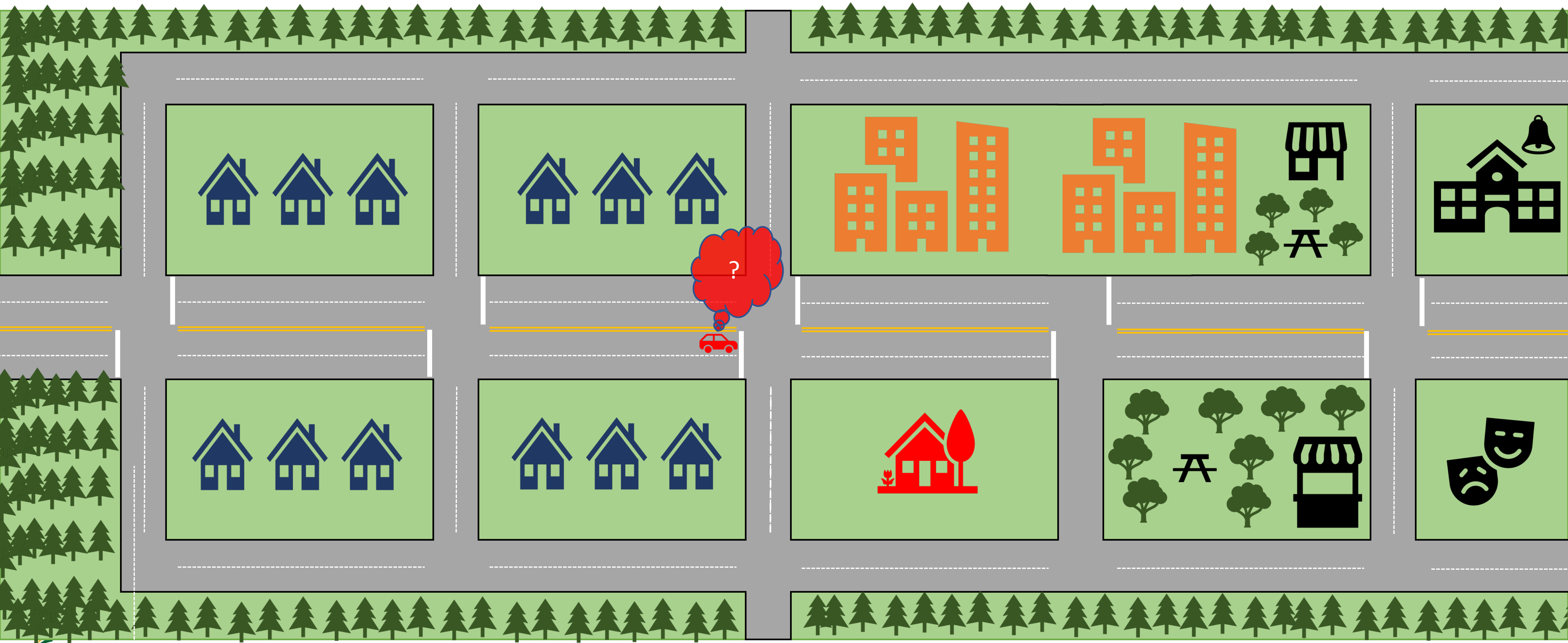


Traffic flow example

Day 3: Wednesday – afternoon

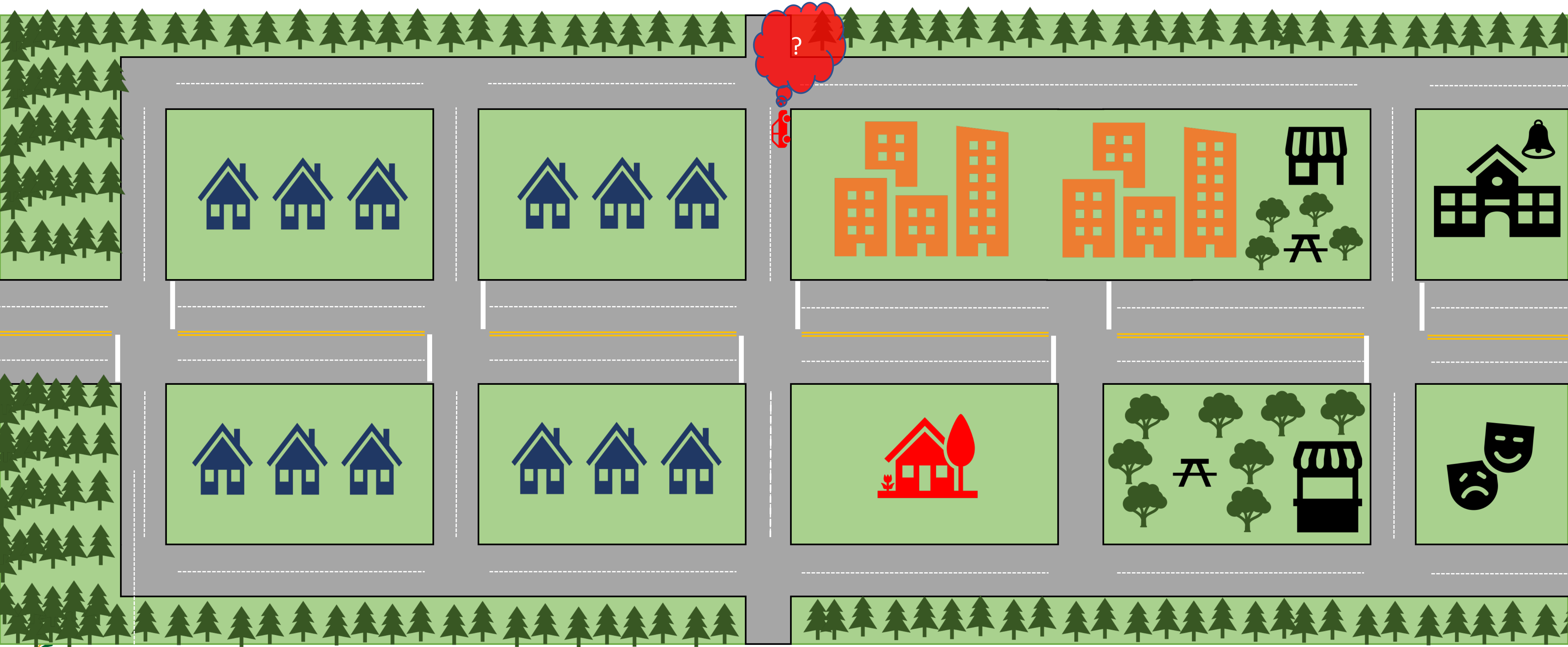
Day: Wednesday

Traffic flow example



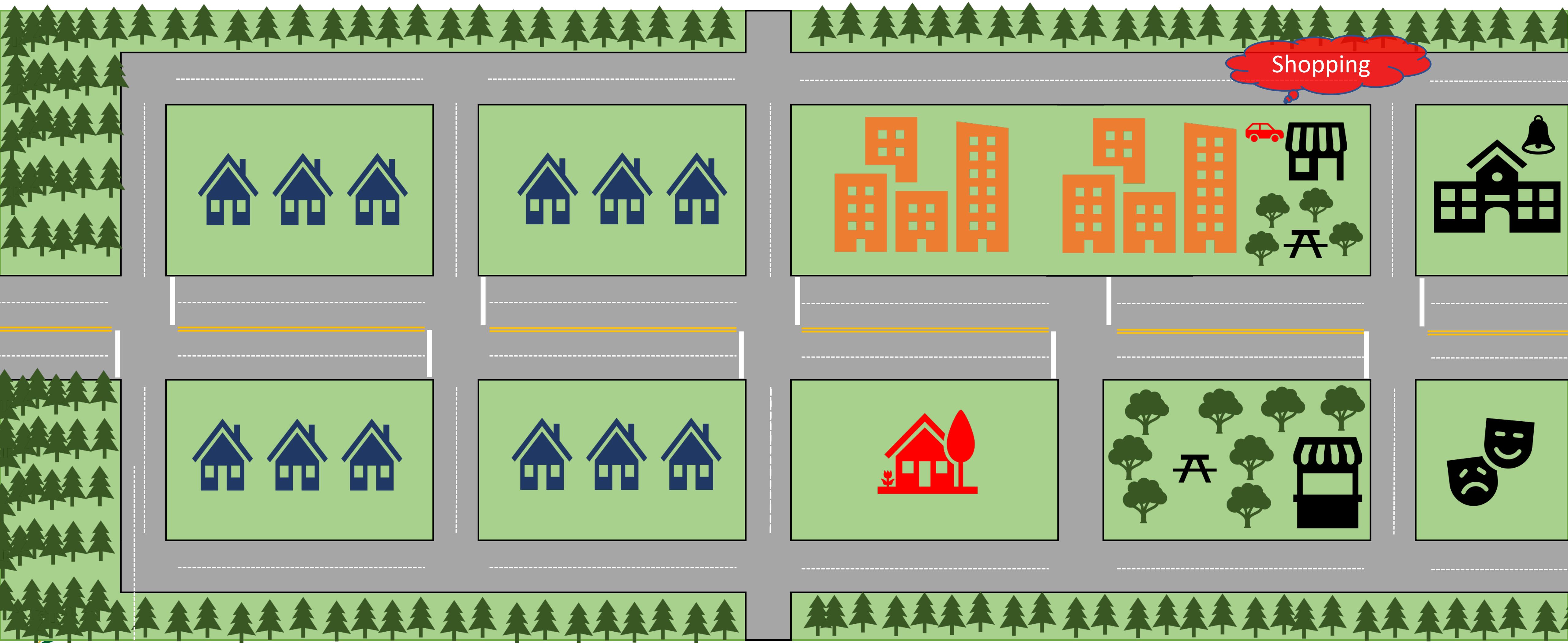
Day: Wednesday

Traffic flow example



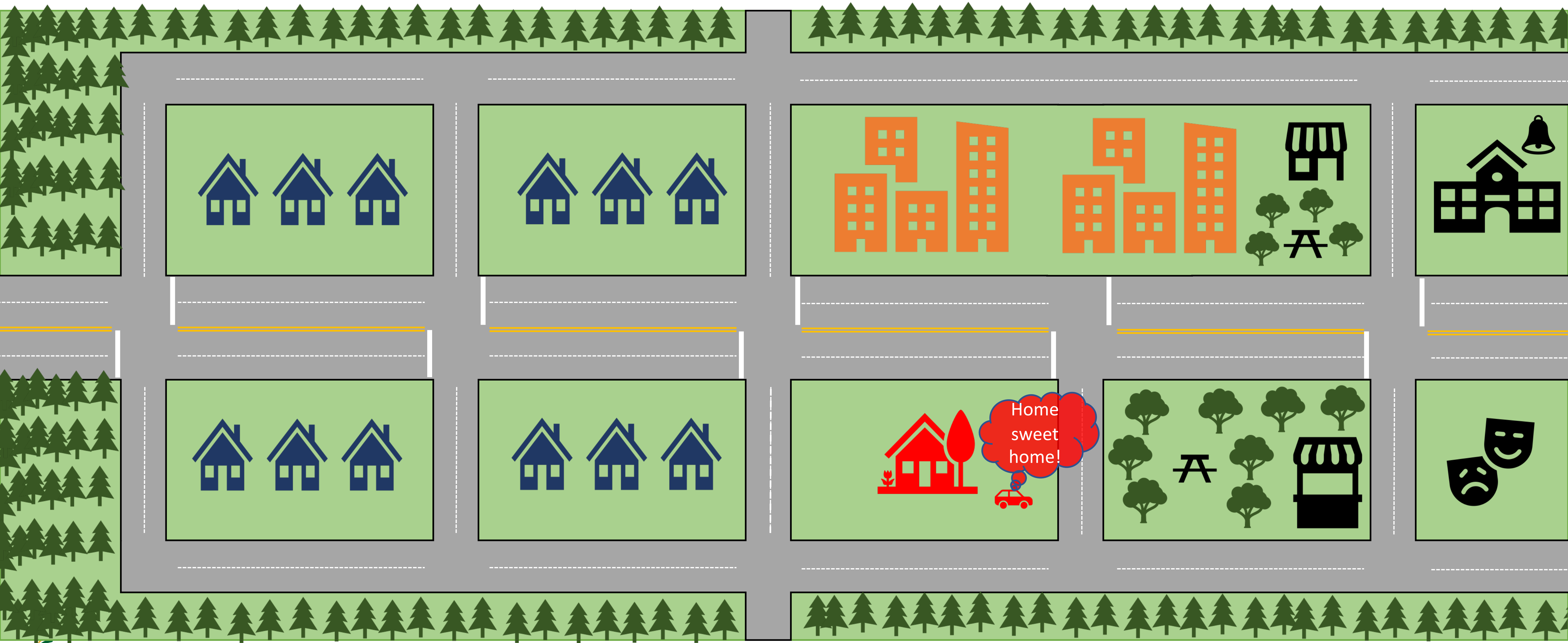
Traffic flow example

Day: Wednesday
Shopping: Groceries



Traffic flow example

Day: Wednesday
Knowledge: This person sometimes stops by different shops before arriving home in the afternoon.

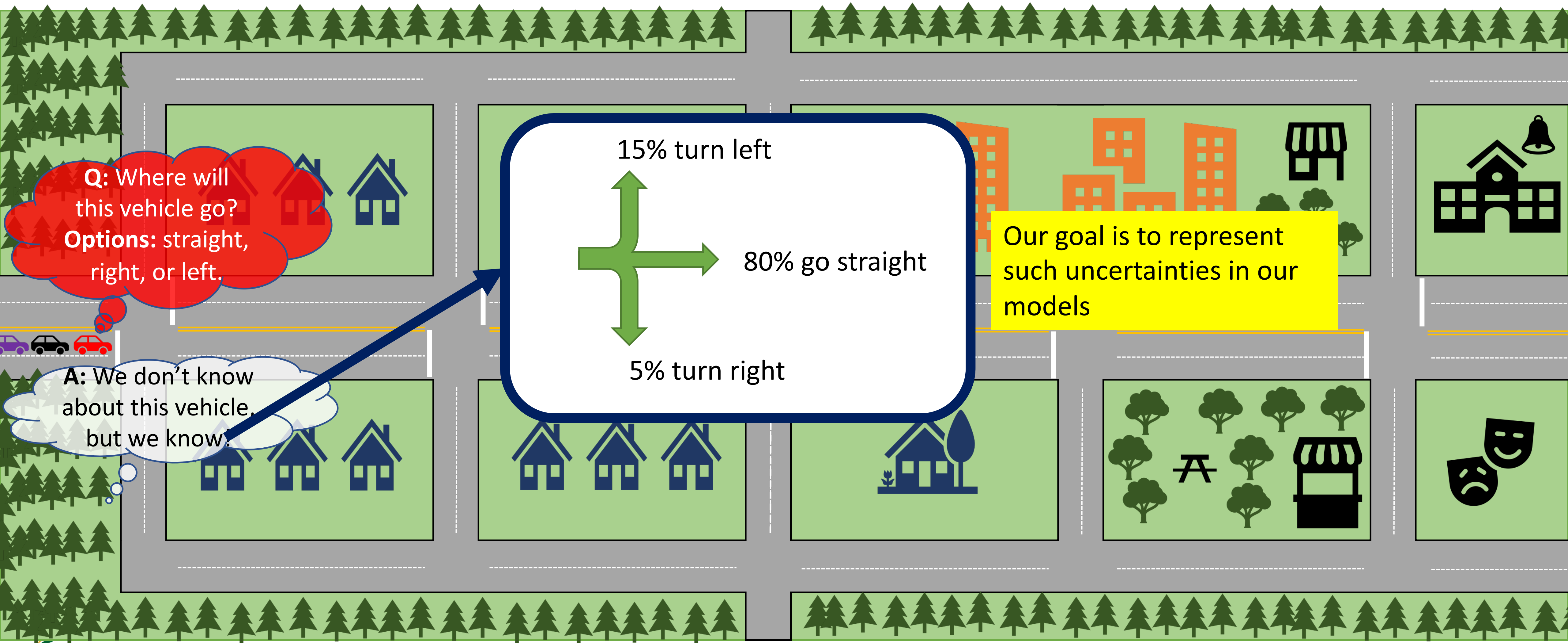


Long story short

- Humans naturally follow some regularities ….
- However, it is highly challenging to perfectly capture why and when a person will visit a particular place or will do a certain thing.
- Neither is it feasible or ethical to collect such data.
- How do we study such uncertain systems?
    - Introduce randomness in our models to capture uncertainty
    - … and make simplifications

Day: Monday

Traffic flow example



Q: Where will this vehicle go? **Options:** straight, right, or left.

A: We don't know about this vehicle, but we know:

15% turn left

80% go straight

5% turn right

Our goal is to represent such uncertainties in our models

Random numbers

- Uncertainty can be represented using random numbers.
- E.g.: rolling a die will result in one of the six possible cases {1,2,3,4,5,6}.

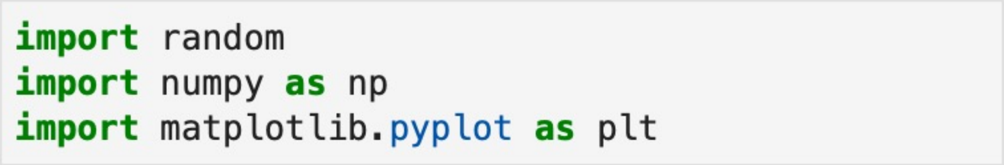
```
die = np.random.randint(1,7)
dice = np.random.randint(1,7,size=2)
```

Random numbers

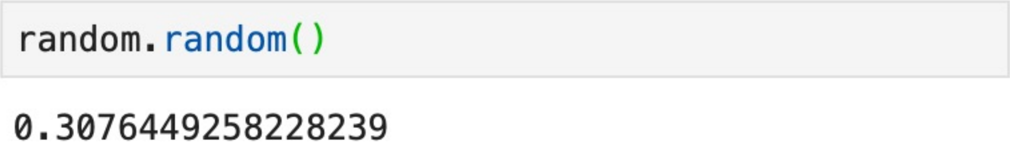
- In simulation models, random numbers are often implemented based on a family of algorithms called **pseudo-random number generators**.
  - Produces a sequence of numbers based on an initial **seed** value.
  - Fast
  - Reproducible/deterministic
- Many programming languages and simulation tools use the Mersenne Twister pseudo-random number generator.
  - Linear Congruential Generator (LCG) is also very popular.

Random numbers in Python

- Can use `random` or `np.random` modules.
- Default random number generation [0,1)

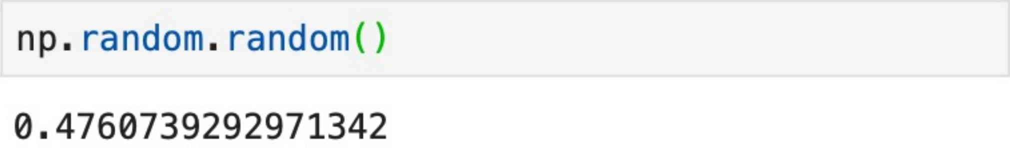


```
import random
import numpy as np
import matplotlib.pyplot as plt
```



```
random.random()
```

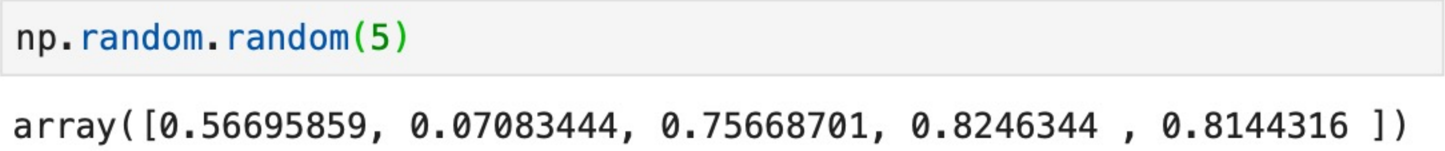
```
0.3076449258228239
```



```
np.random.random()
```

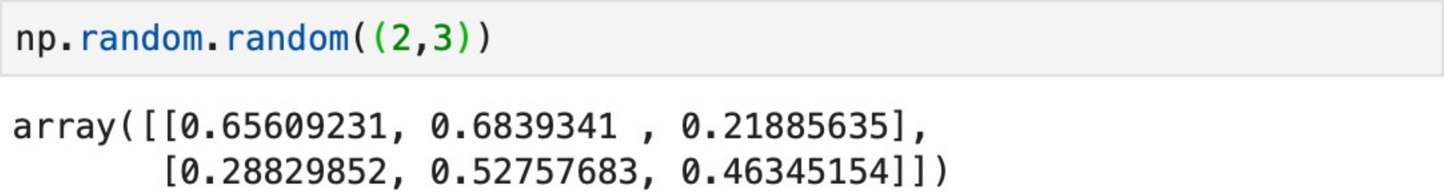
```
0.4760739292971342
```

- We prefer NumPy’s random number generator because it can return NumPy arrays/matrices.



```
np.random.random(5)
```

```
array([0.56695859, 0.07083444, 0.75668701, 0.8246344 , 0.8144316 ])
```



```
np.random.random((2,3))
```

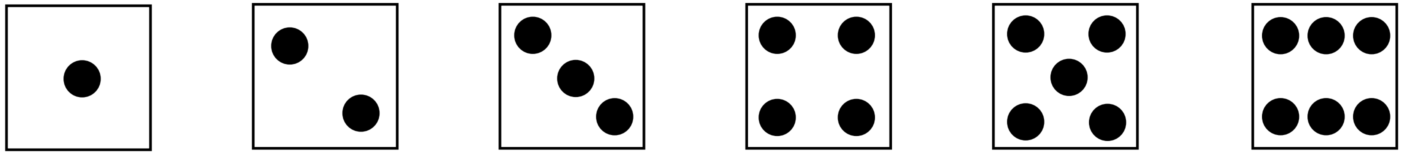
```
array([[0.65609231, 0.6839341 , 0.21885635],
       [0.28829852, 0.52757683, 0.46345154]])
```

Random integers

- `np.random.randint(x)` generates a random integer between `0` (inclusive) and `x` (exclusive).
- `np.random.randint(x,y)` generates a random integer between `x` (inclusive) and `y` (exclusive).
- You can generate multiple integers by adding the `size` argument.
  - `np.random.randint(x,size=k)`
  - `np.random.randint(x,y,size=m)`
  - `np.random.randint(x,y,size=(n,p))`

Let’s roll some dice using `randint`

- Roll a fair die



- Now, roll two fair dice

- Finally, roll five fair dice.

Now you can play Yahtzee!

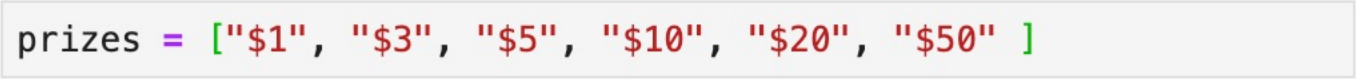
The concept of equal chances

- Random numbers we learned so far have equal chances
  - E.g.: `np.random.random()` all float numbers [0,1) have equal probability to occur.
  - `np.random.randint(3)` here 0, 1, and 2 have equal probability to occur.
- This concept is also known as `uniform` distribution.
- `np.random.uniform(x,y)` generates float numbers between `x` (inclusive) and `y` (exclusive).
  - `np.random.random()` = `np.random.uniform(0.0,1.0)`
- The `size` argument can still be used as shown previously.
  - `np.random.uniform(x,y,size=k)`
  - `np.random.uniform(x,y,size=(m,n))`

choice() function

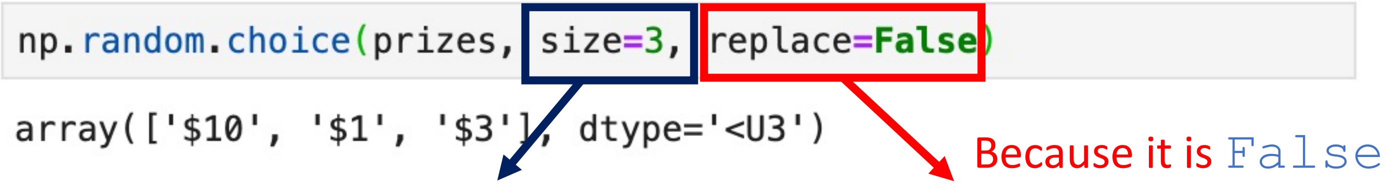
- We can use np.random.choice() function which takes a list or NumPy array and returns one of these values, each with equal chances.

- Assume you have a list of prizes that you want to draw one at a time and give away.



```
prizes = ["$1", "$3", "$5", "$10", "$20", "$50" ]
```

- You can use the choice function to do that.



```
np.random.choice(prizes, size=3, replace=False)
```

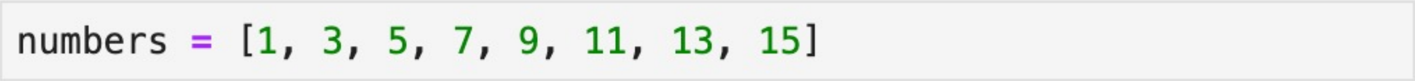
```
array(['$10', '$1', '$3'], dtype='<U3')
```

Number of draws

Because it is False, the same element won’t be drawn again.

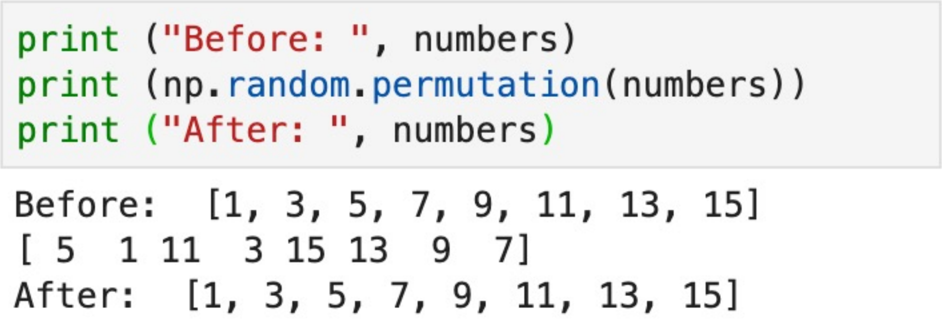
Shuffling

- Given a list/array, NumPy’s random submodule can help you shuffle the order of elements.



```
numbers = [1, 3, 5, 7, 9, 11, 13, 15]
```

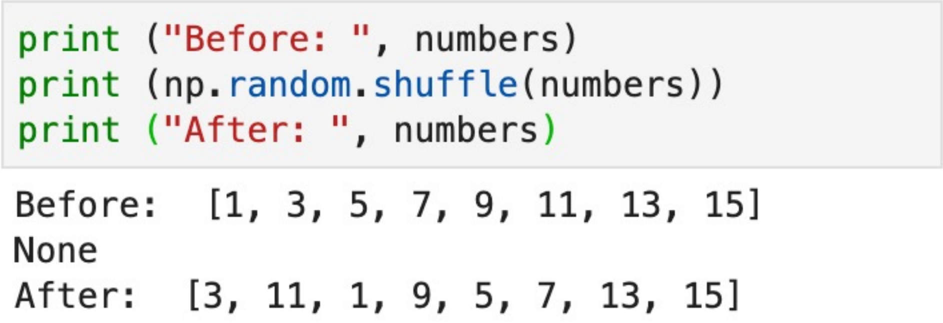
- Original list unchanged



```
print ("Before: ", numbers)
print (np.random.permutation(numbers))
print ("After: ", numbers)
```

```
Before:  [1, 3, 5, 7, 9, 11, 13, 15]
[ 5  1 11  3 15 13  9  7]
After:  [1, 3, 5, 7, 9, 11, 13, 15]
```

Original list changed



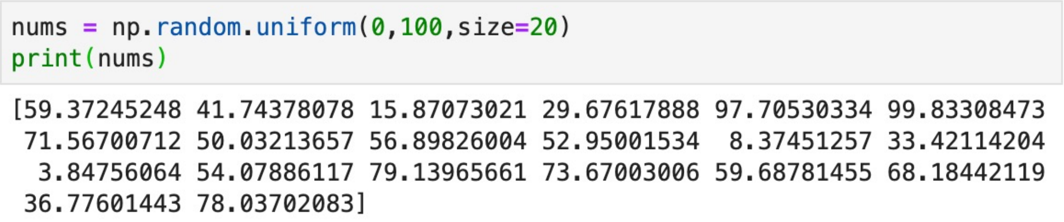
```
print ("Before: ", numbers)
print (np.random.shuffle(numbers))
print ("After: ", numbers)
```

```
Before:  [1, 3, 5, 7, 9, 11, 13, 15]
None
After:  [3, 11, 1, 9, 5, 7, 13, 15]
```

Reproducing the same sequence

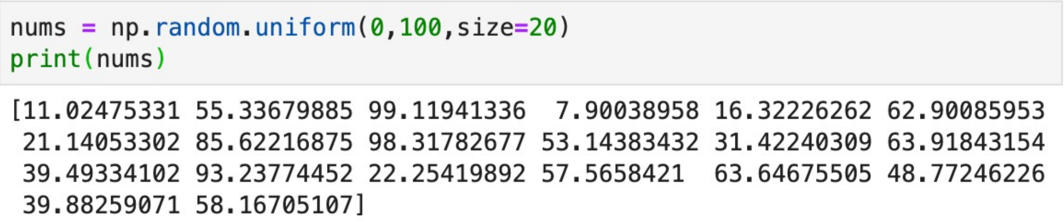
- Recall: pseudo-random number generators are deterministic given the same seed.

Without changing the seed



```
nums = np.random.uniform(0,100,size=20)
print(nums)
```

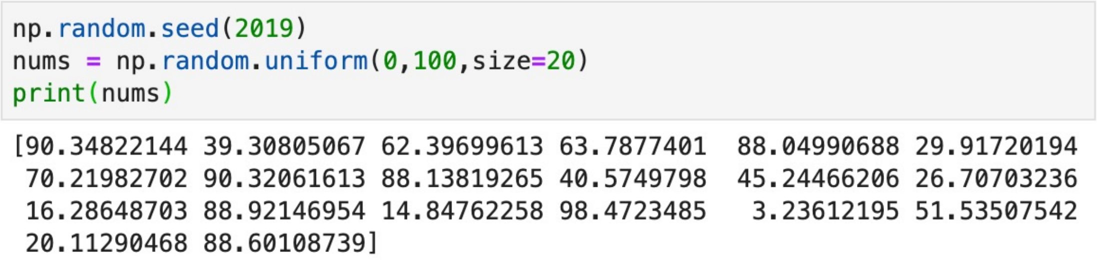
```
[59.37245248 41.74378078 15.87073021 29.67617888 97.70530334 99.83308473
 71.56700712 50.03213657 56.89826004 52.95001534  8.37451257 33.42114204
  3.84756064 54.07886117 79.13965661 73.67003006 59.68781455 68.18442119
 36.77601443 78.03702083]
```



```
nums = np.random.uniform(0,100,size=20)
print(nums)
```

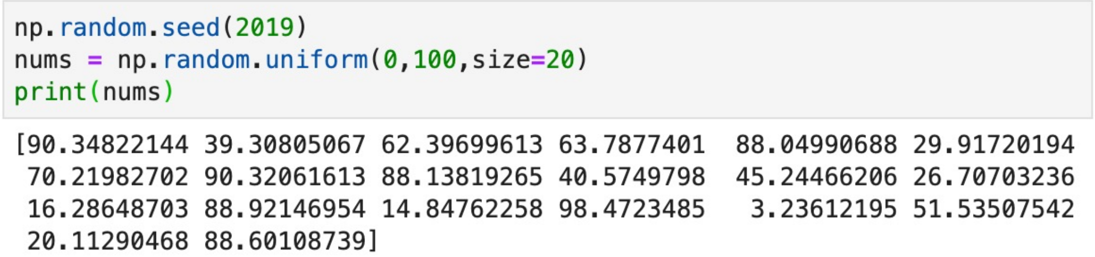
```
[11.02475331 55.33679885 99.11941336  7.90038958 16.32226262 62.90085953
 21.14053302 85.62216875 98.31782677 53.14383432 31.42240309 63.91843154
 39.49334102 93.23774452 22.25419892 57.5658421  63.64675505 48.77246226
 39.88259071 58.16705107]
```

Setting the seed



```
np.random.seed(2019)
nums = np.random.uniform(0,100,size=20)
print(nums)
```

```
[90.34822144 39.30805067 62.39699613 63.7877401  88.04990688 29.91720194
 70.21982702 90.32061613 88.13819265 40.5749798  45.24466206 26.70703236
 16.28648703 88.92146954 14.84762258 98.4723485   3.23612195 51.53507542
 20.11290468 88.60108739]
```



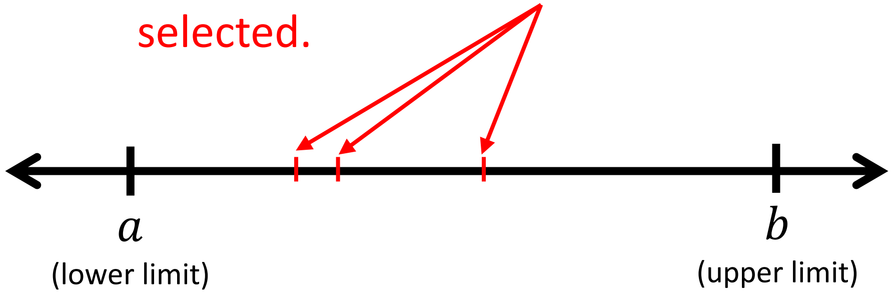
```
np.random.seed(2019)
nums = np.random.uniform(0,100,size=20)
print(nums)
```

```
[90.34822144 39.30805067 62.39699613 63.7877401  88.04990688 29.91720194
 70.21982702 90.32061613 88.13819265 40.5749798  45.24466206 26.70703236
 16.28648703 88.92146954 14.84762258 98.4723485   3.23612195 51.53507542
 20.11290468 88.60108739]
```

Equal chances (uniform distribution): recap

Decimal numbers

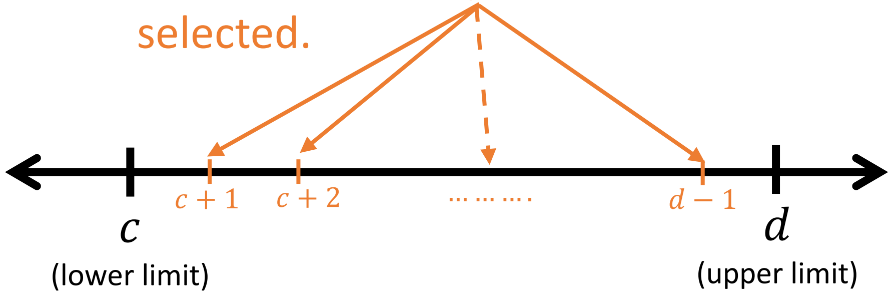
Any arbitrary points within lower (a) and upper (b) limits have equal chances to be selected.



```
np.random.uniform(a,b)
```

Integers

Any integers within lower (c) and upper (d) limits have equal chances to be selected.



```
np.random.randint(c,d+1)
```