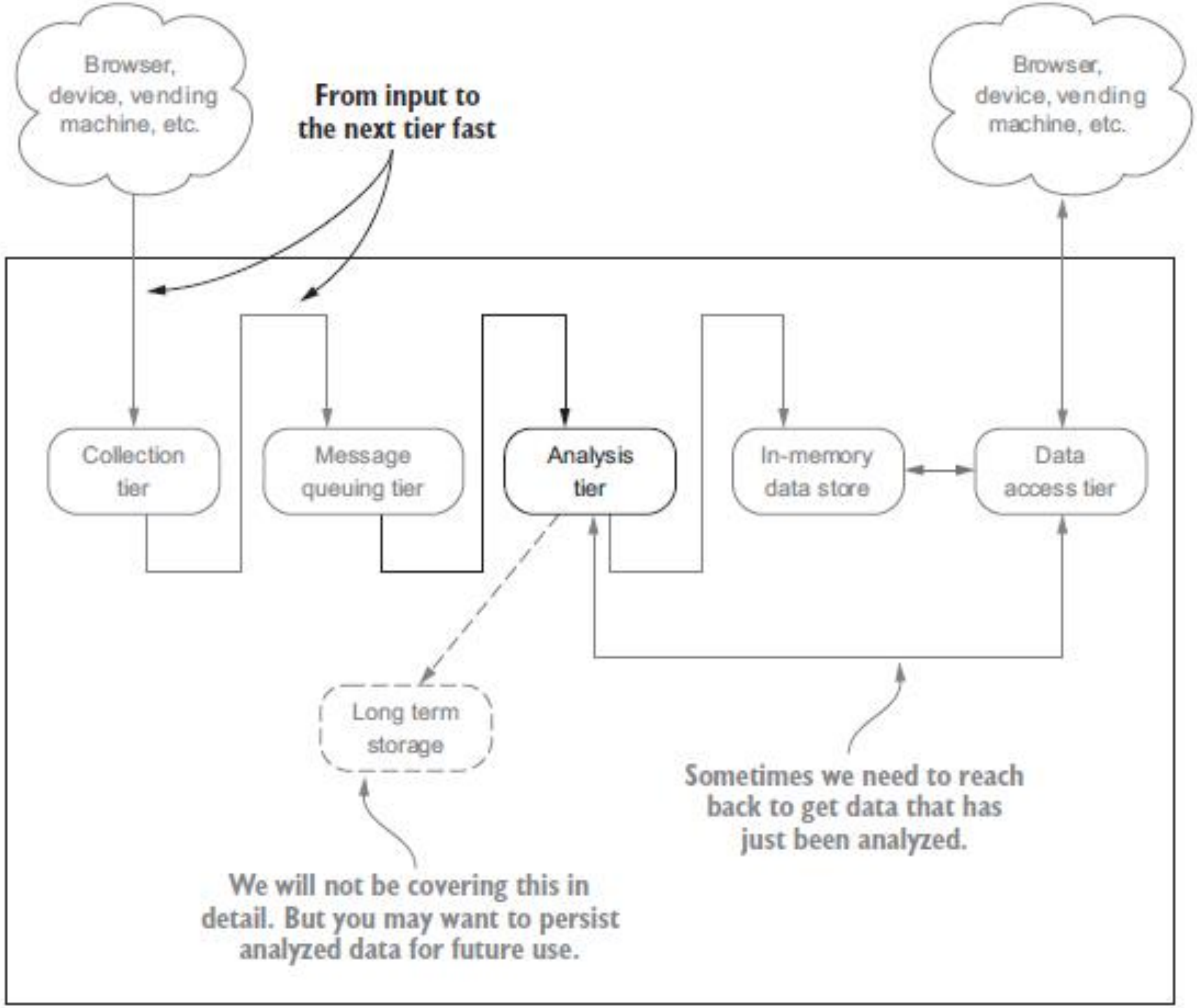
Day 6
Think before analyzing data stream

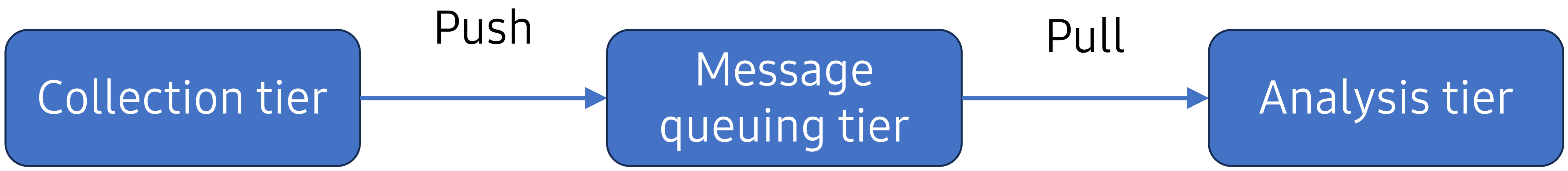
Lecturer: Le Minh Tan

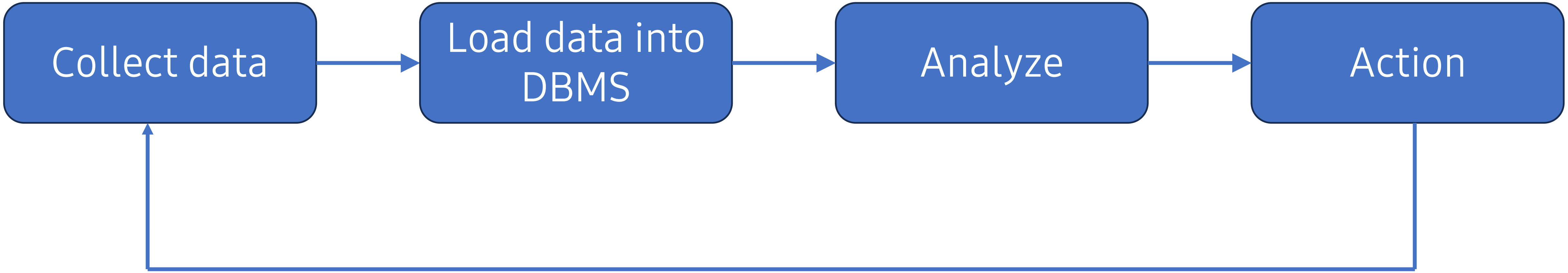

Browser, device, vending machine, etc.
From input to the next tier fast
Browser, device, vending machine, etc.
Collection tier
Message queuing tier
Analysis tier
In-memory data store
Data access tier
Long term storage
Sometimes we need to reach back to get data that has just been analyzed.
We will not be covering this in detail. But you may want to persist analyzed data for future use.

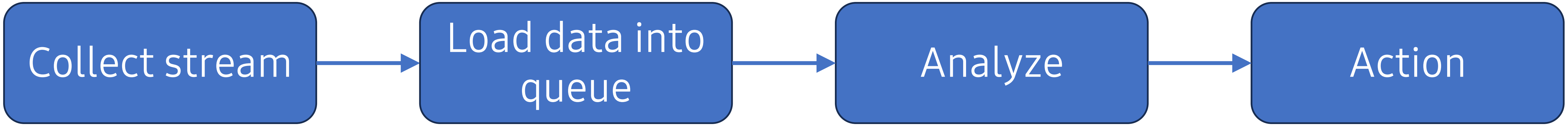
Outline

I. In-flight data analysis
II. Distributed stream-processing architecture
III. Time
IV. Summarization techniques

In-flight data


Collection tier
Push
Message queuing tier
Pull
Analysis tier


Collect data
Load data into DBMS
Analyze
Action

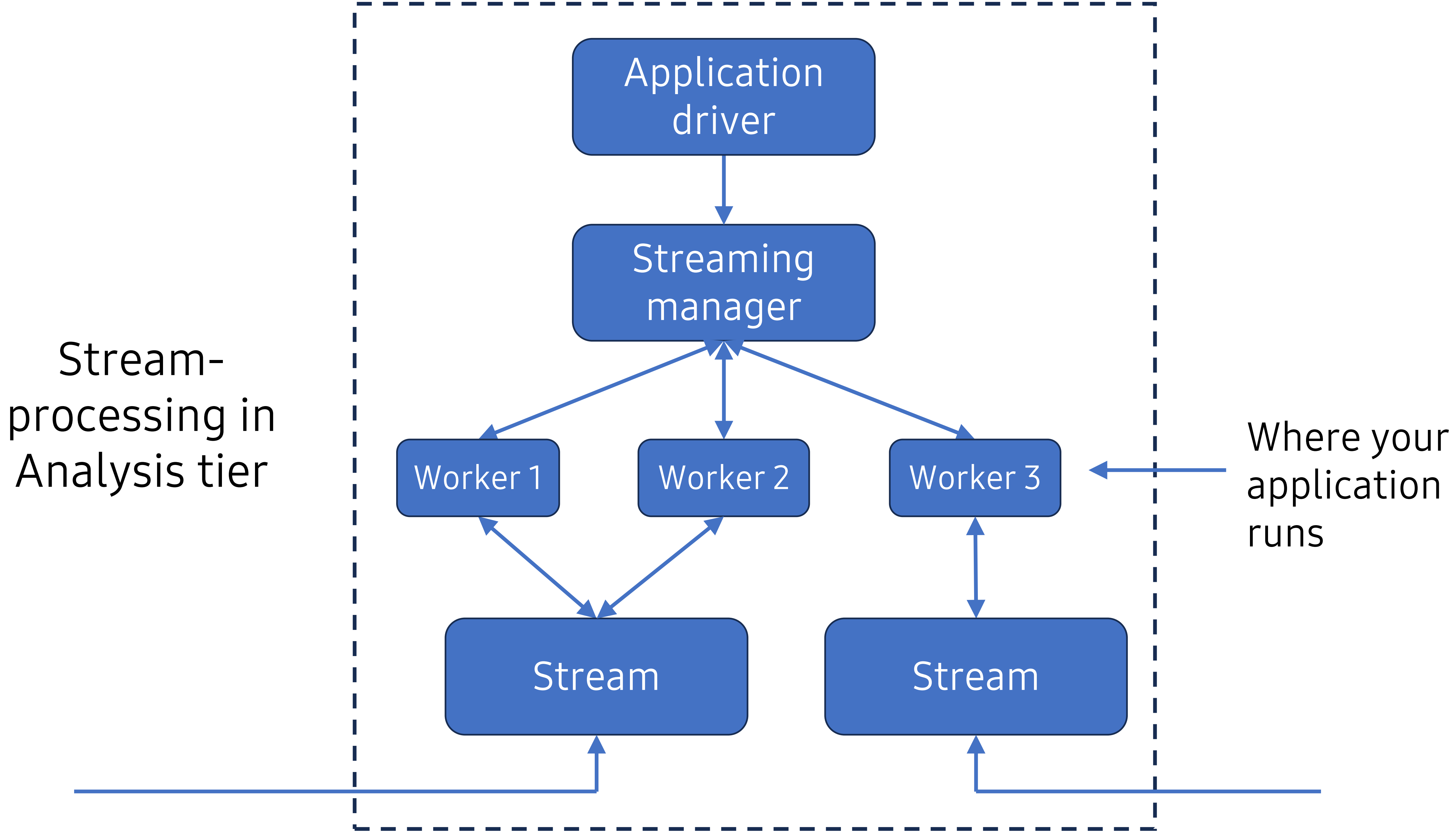

Collect stream
Load data into queue
Analyze
Action

Case study: Smart home

More example

- A restaurant may want to know how effective their pricing strategies are on ordering app:
  - How many % increase in voucher usage?
  - % of customers picking the related dishes again?

https://en.wikipedia.org/wiki/Pricing_strategies



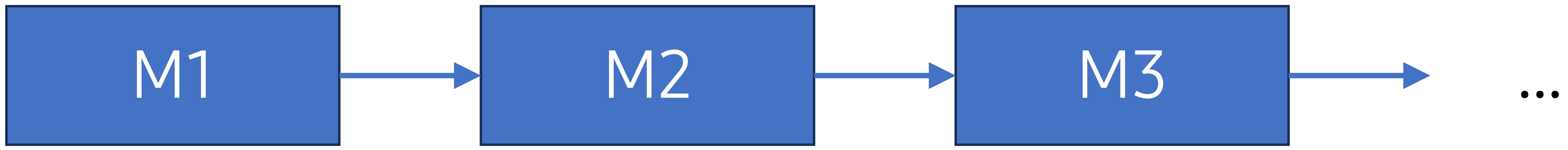
Application driver
Streaming manager
Worker 1
Worker 2
Worker 3
Stream
Stream
Stream-processing in Analysis tier
Where your application runs

Apache stream-processing technologies

- Spark (streaming)
- Storm
- Flink
- Samza

Key features

- Guarantees
  - At-most-once
    - Message is dropped
  - At-least-once
    - Message is safe, but can be reread
  - Exactly-once
    - Ignore duplicates


M1
M2
M3
...

Assumptions/constraints

- One-pass: Re-processing is impossible.
- Concept drift: The predictive models are outdated.
- Resource: Flow rate strains the system.
- Domain: Bussiness requirements come in.

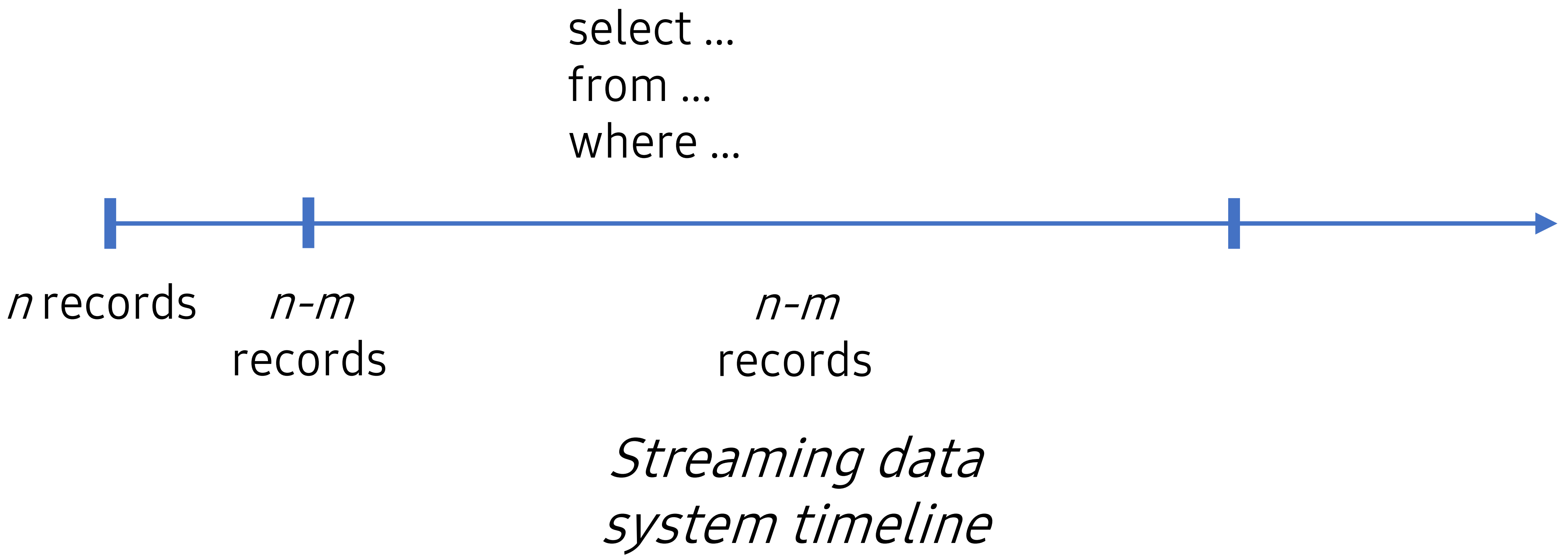
Time

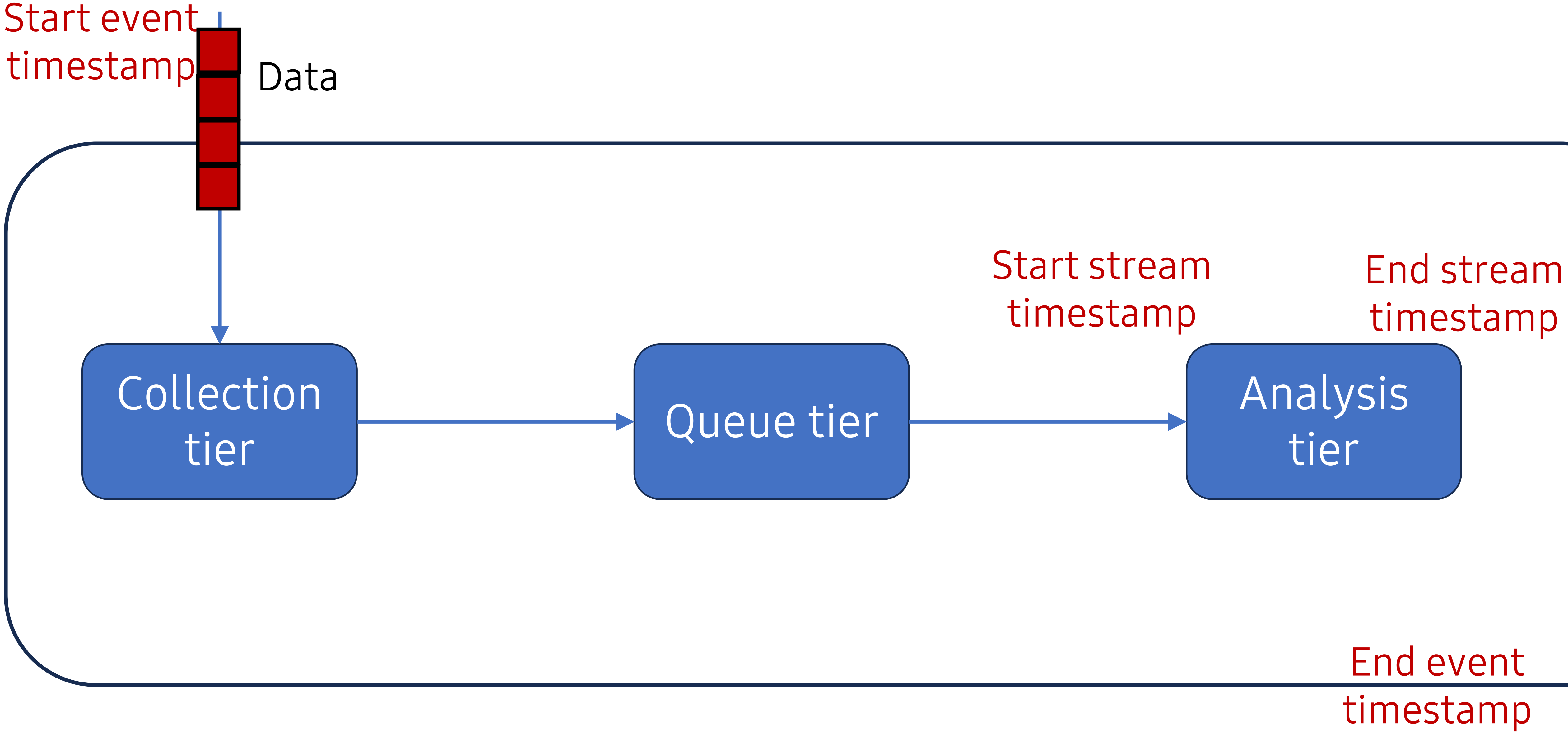
select …
from …
where …

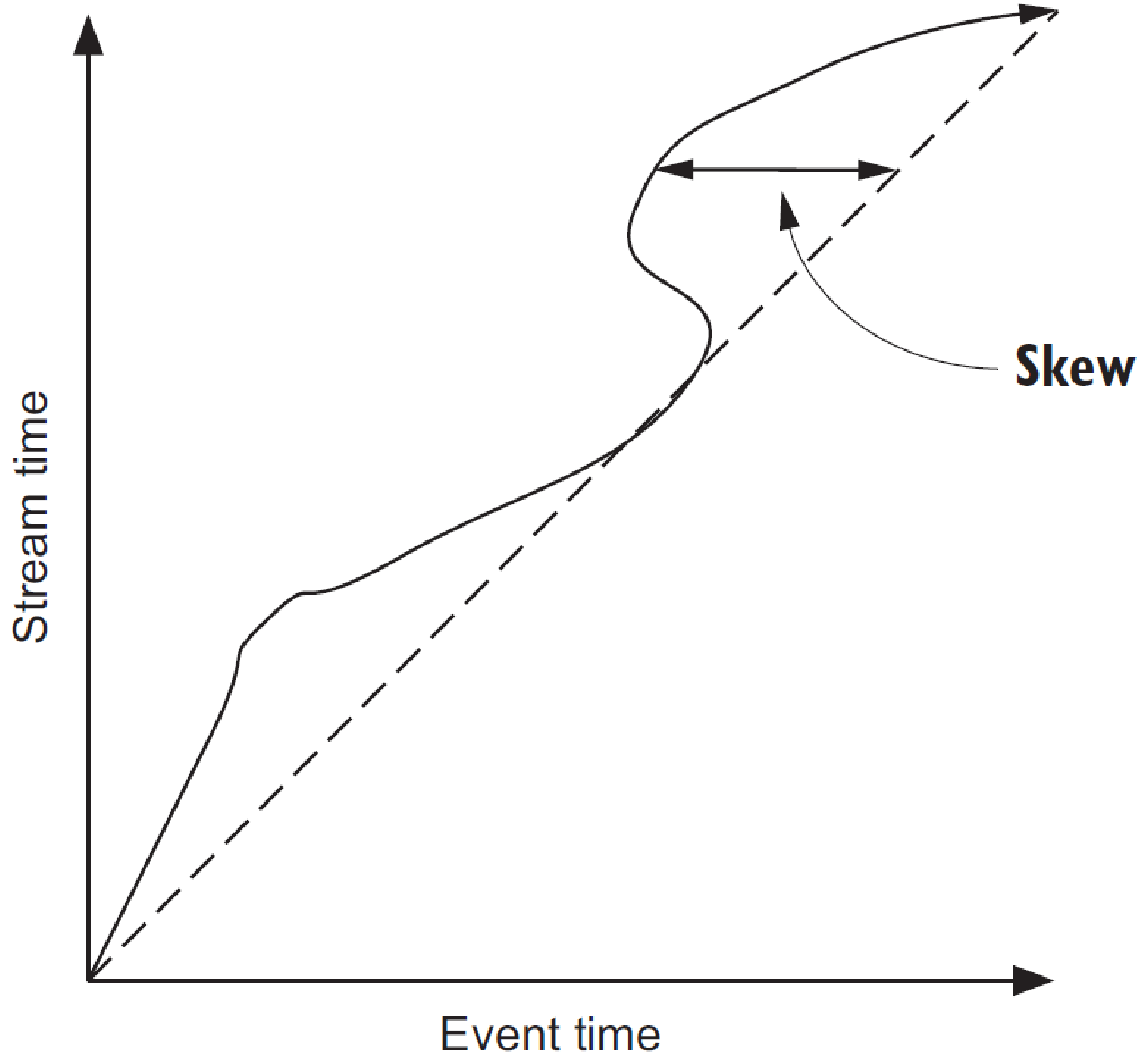
n records

n records

Static data
system timeline


select ...
from ...
where ...
n records
n-m records
n-m records
Streaming data system timeline


Start event
timestamp
Data
Collection tier
Queue tier
Start stream
timestamp
End stream
timestamp
Analysis tier
End event
timestamp

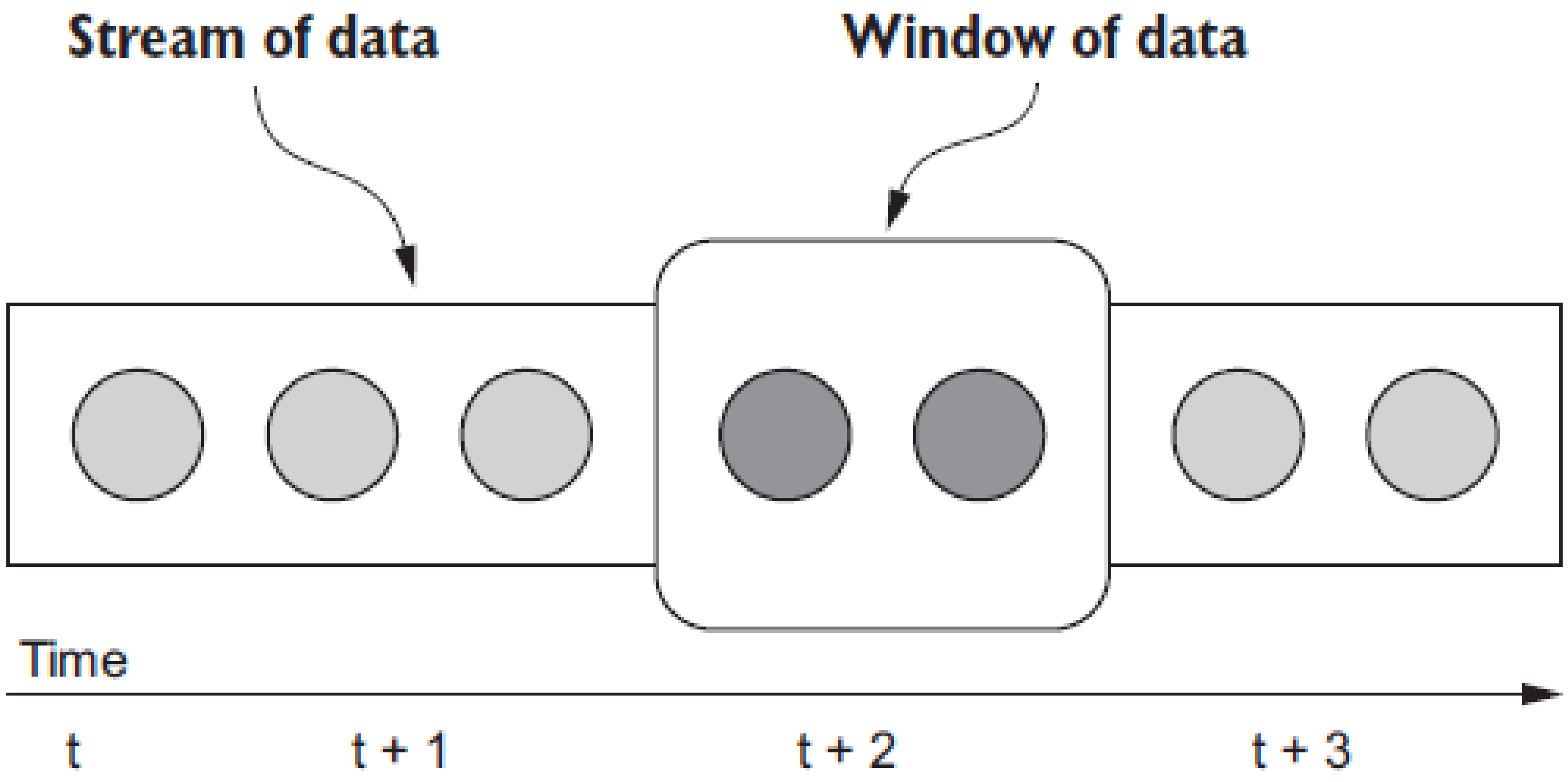

Stream time
Skew
Event time

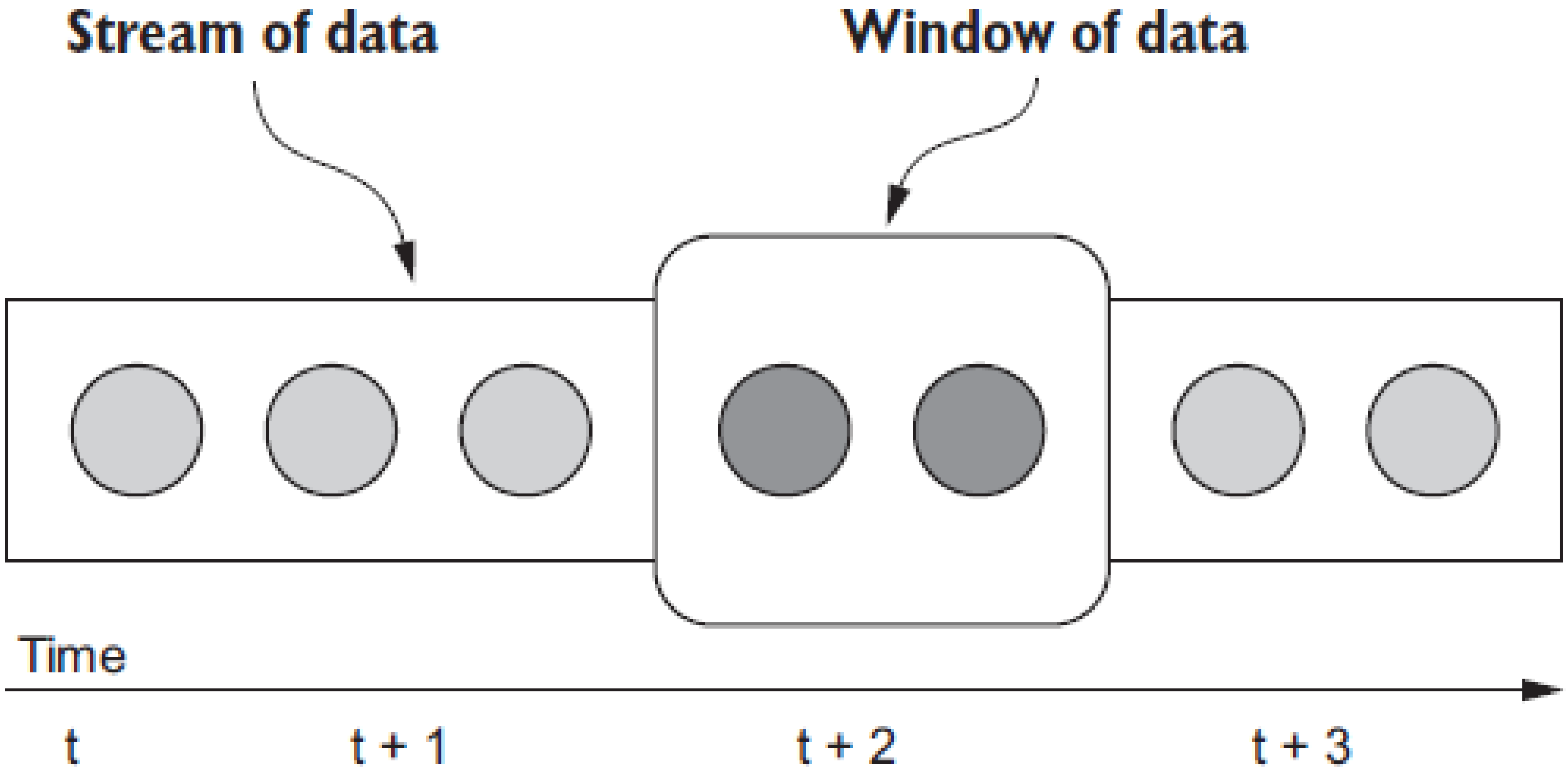
Total number of dishes ordered in 1 min?
The result is refreshed every 5 seconds

Total number of dishes ordered in 1 min?
The result is refreshed every 5 seconds

$$f(x) = \sum g(x)$$

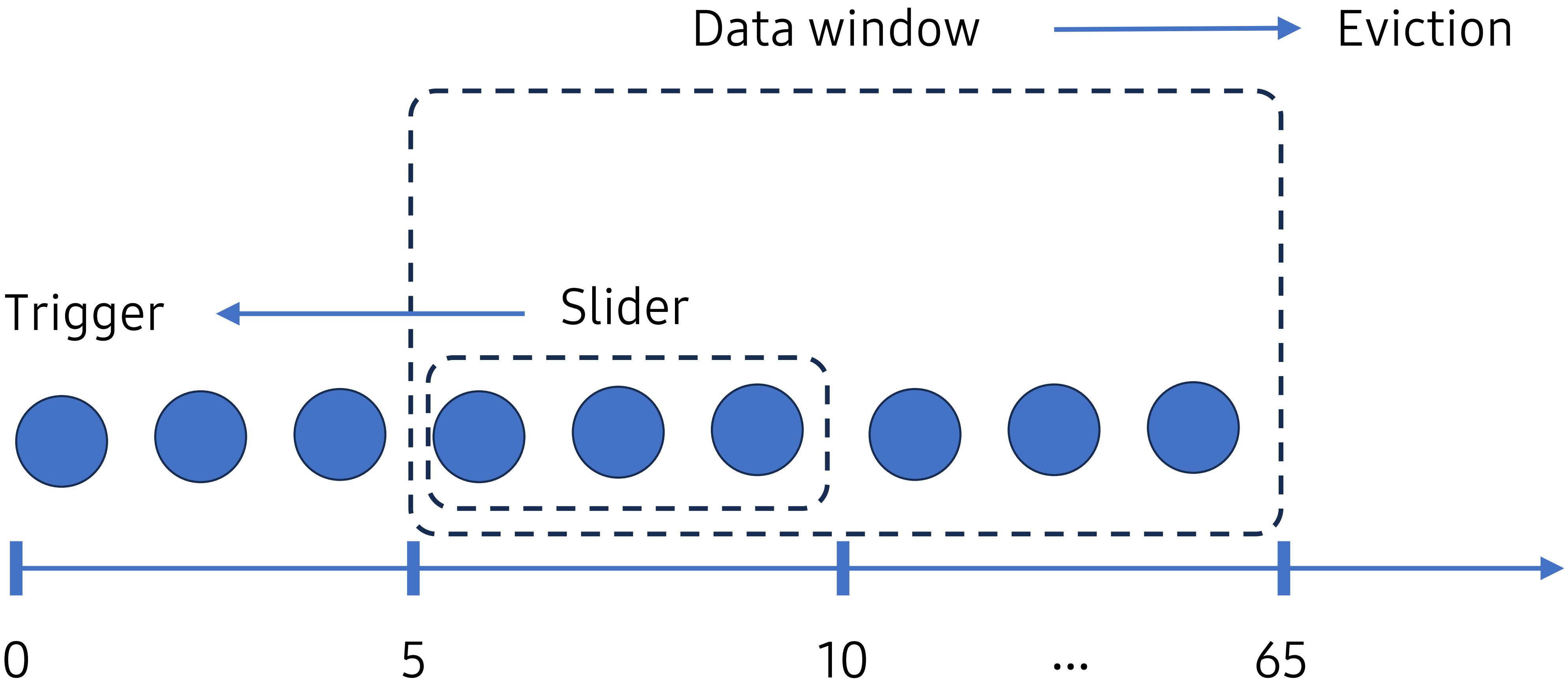
x: List of orders in orders
$g(x)$: Count dishes of x


Stream of data
Window of data
Time
t
t + 1
t + 2
t + 3

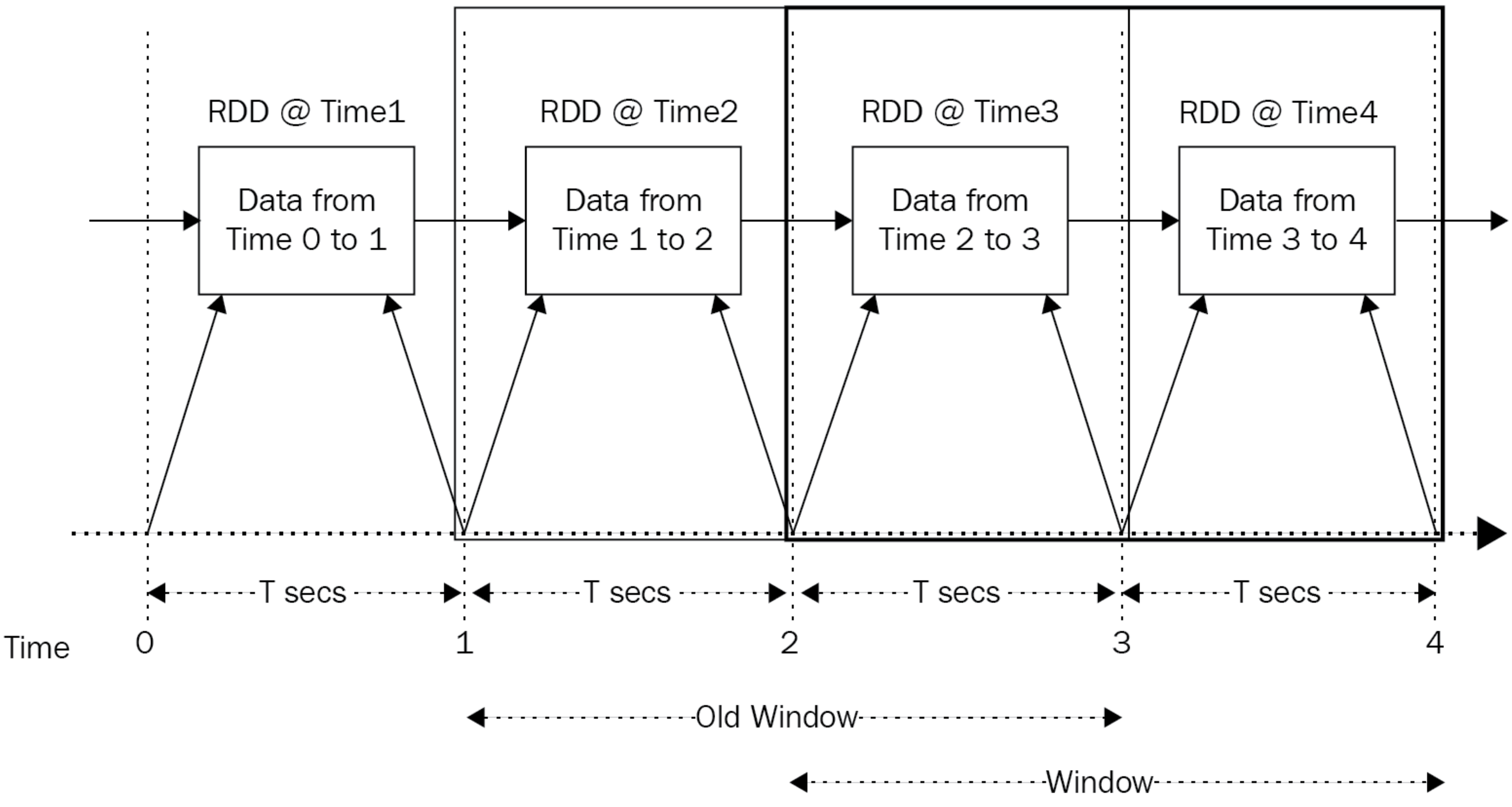


- Trigger policy: Rules to start processing data in window.
- Eviction policy: Rules to decide if a piece of data should be evicted from window.
- Both policies use time or quantity of data.

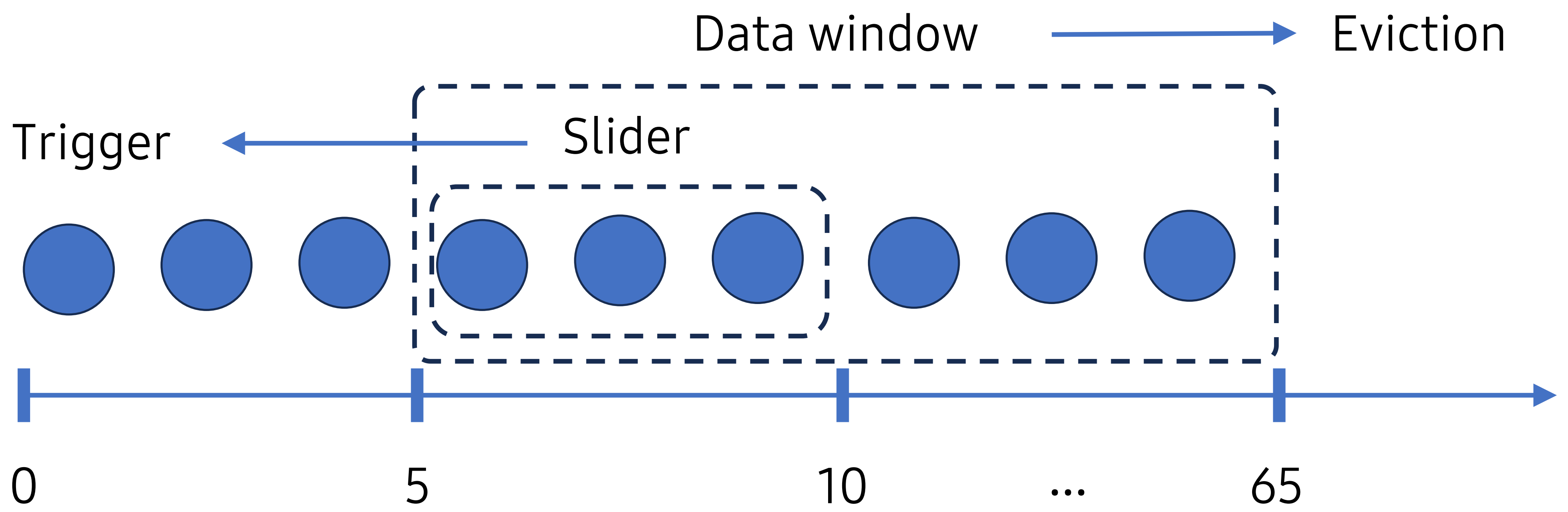
#1: Sliding window (slider): Time policy

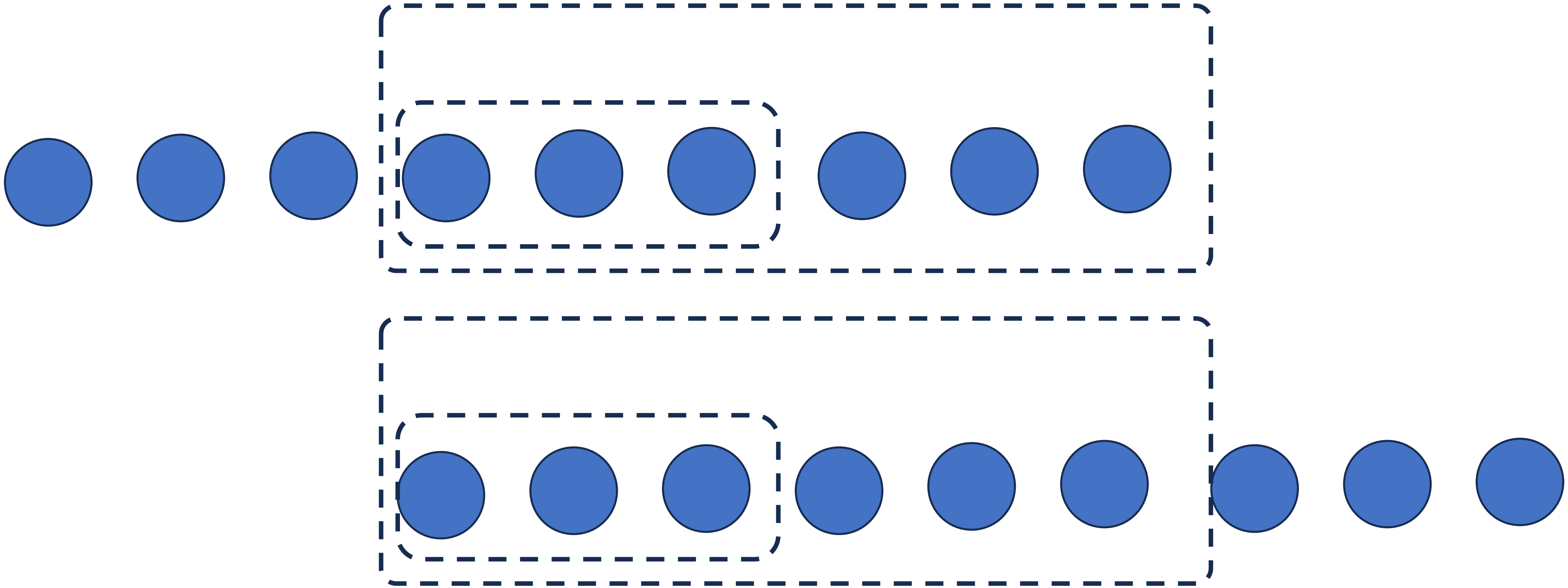


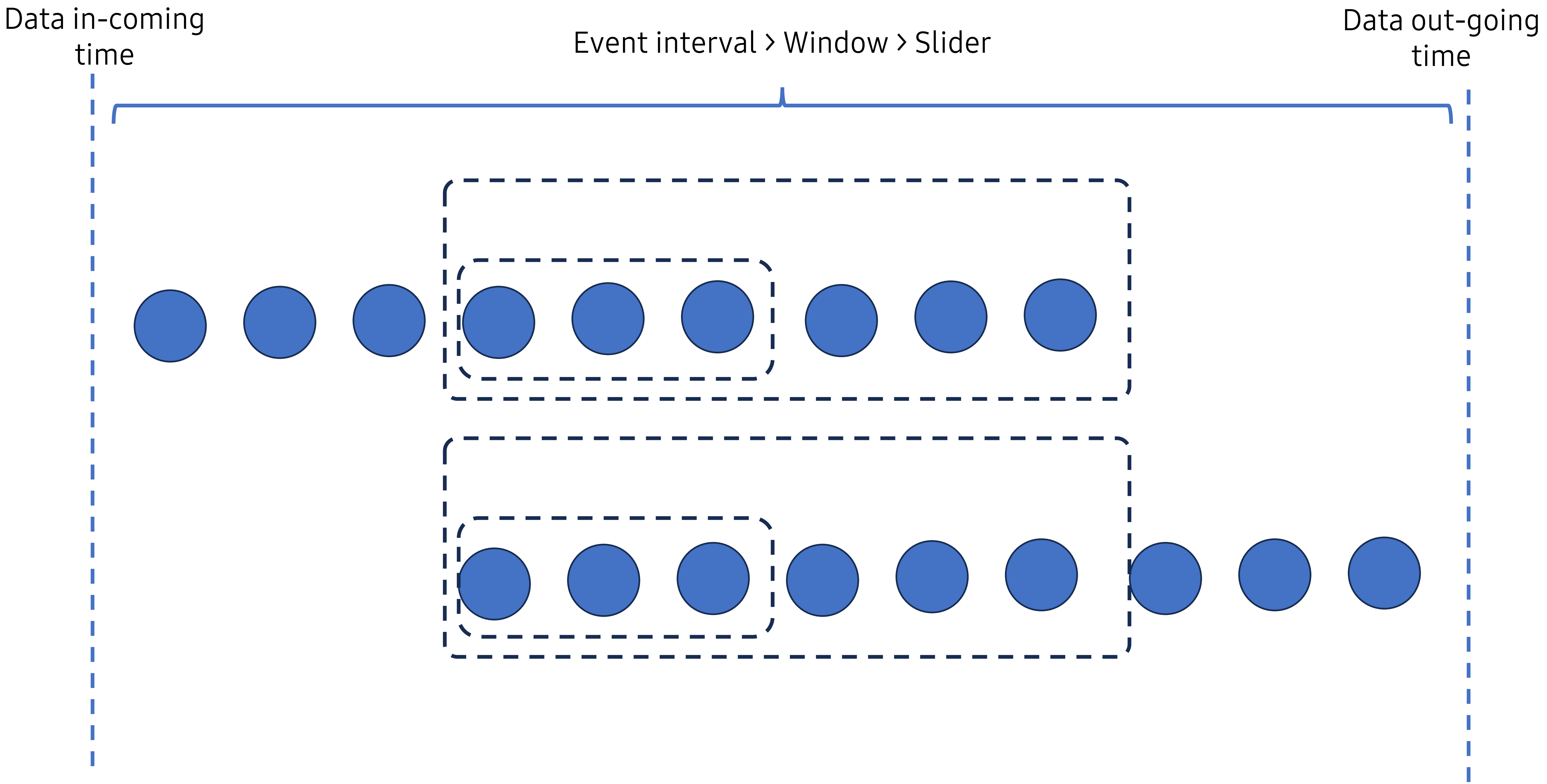
DStream with sliding Window

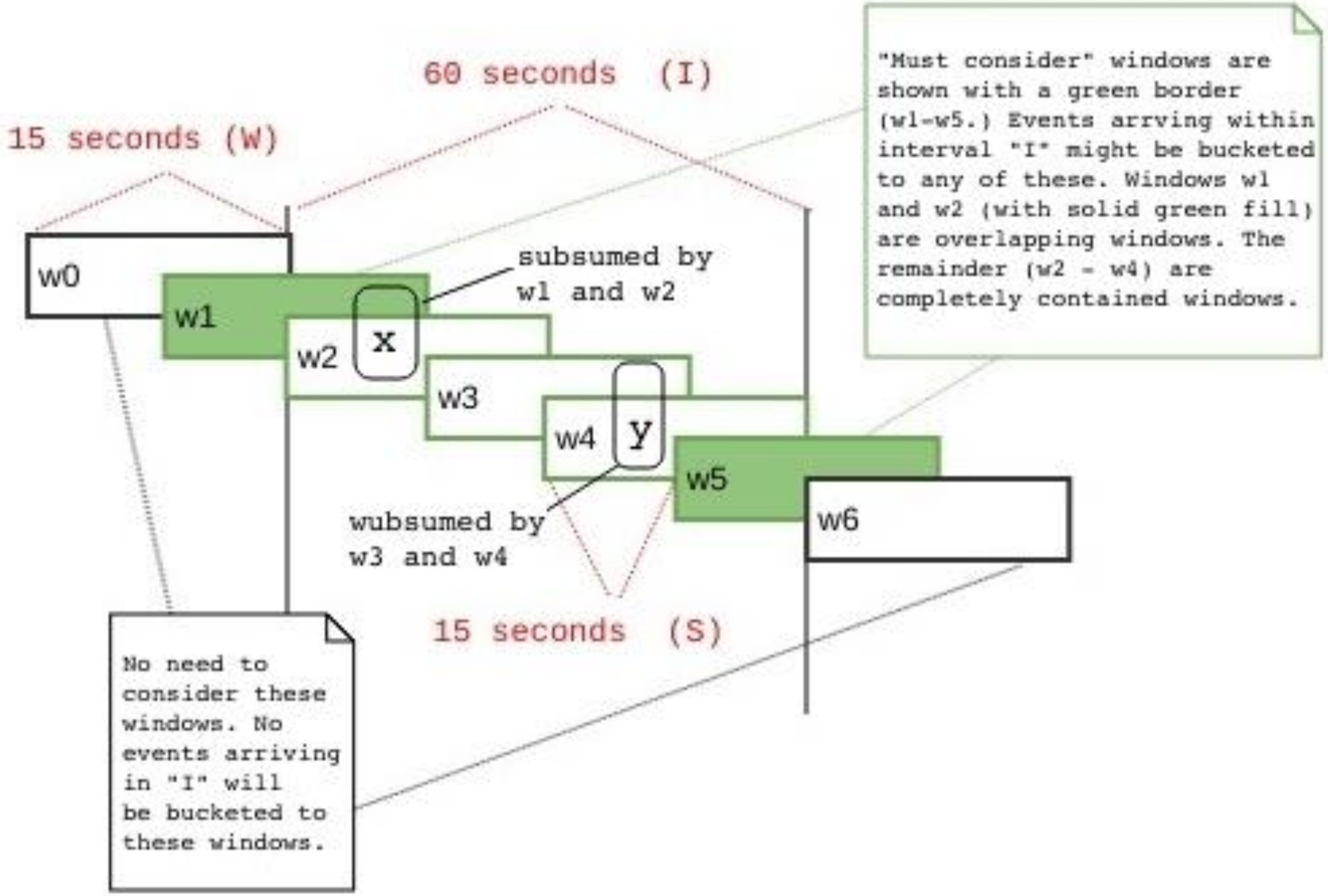


https://www.oreilly.com/library/view/big-data-analytics/9781788628846/ee099fa2-62c7-4cba-a883-635bbc326f9a.xhtml

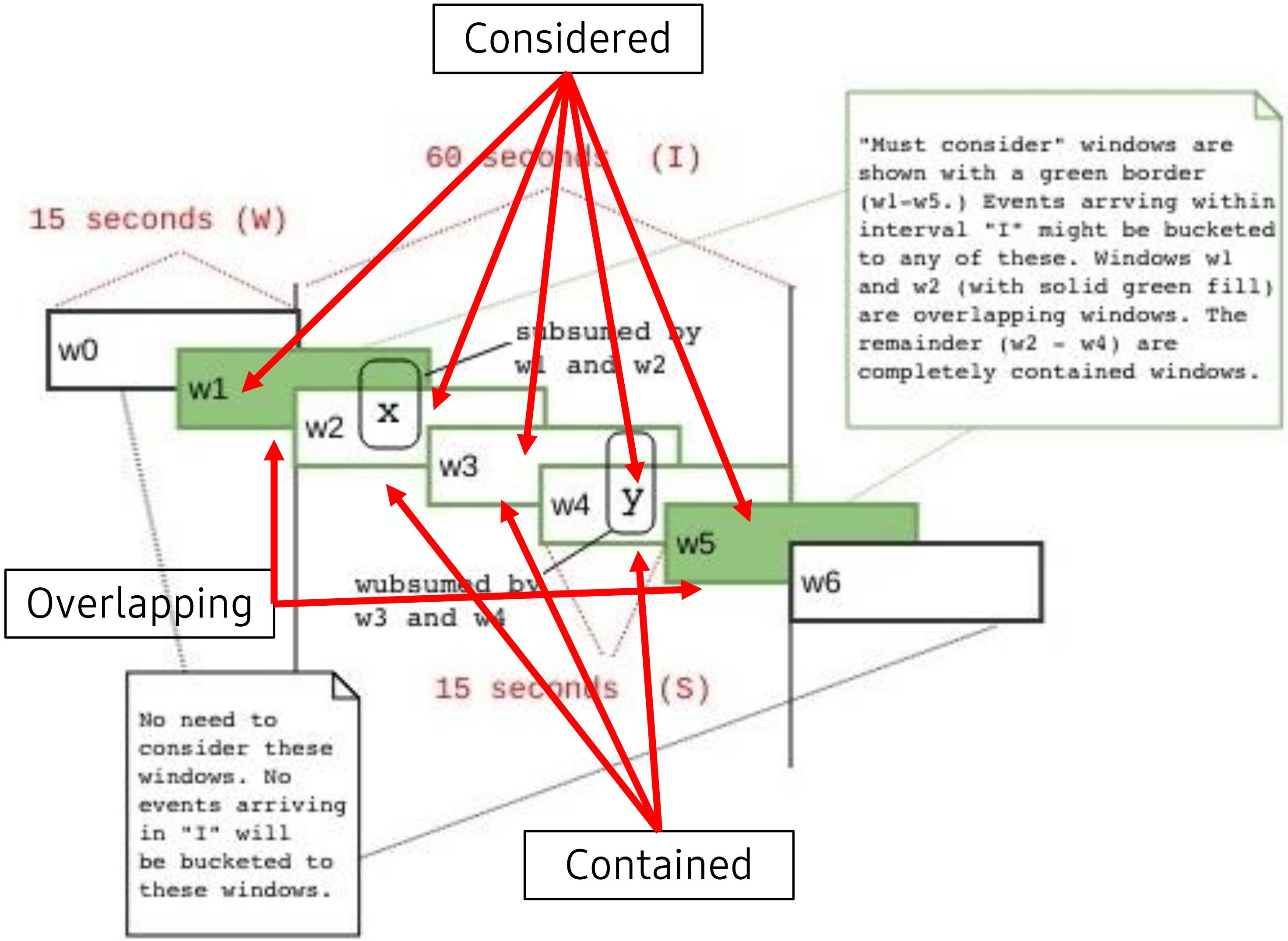

Data window
Eviction
Trigger
Slider
0
5
10
...
65




Data in-coming time
Event interval › Window › Slider
Data out-going time



http://datalackey.com/2019/07/01/sliding-window-processing-spark-structured-streaming-vs-dstreams/



http://datalackey.com/2019/07/01/sliding-window-processing-spark-structured-streaming-vs-dstreams/

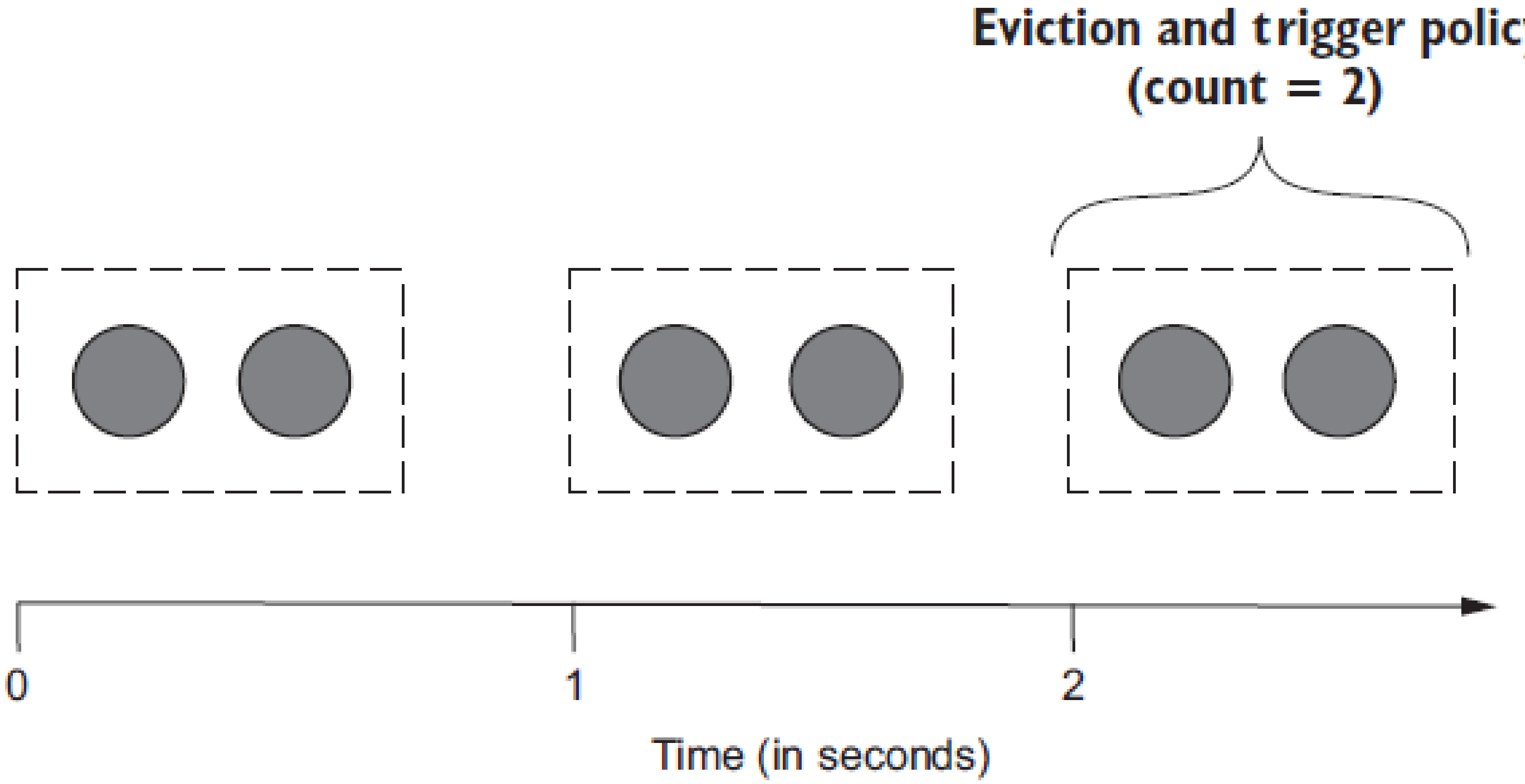
Some formulas

- Number of windows to be considered: $k + kn - 1$
- Number of completely contained windows: $kn - k + 1$
- Number of overlapping windows: $2(k - 1)$
- $k = w/s$
- $n = i/w$
- w: Window time
- s: Slider time
- i: Event time/Interval time

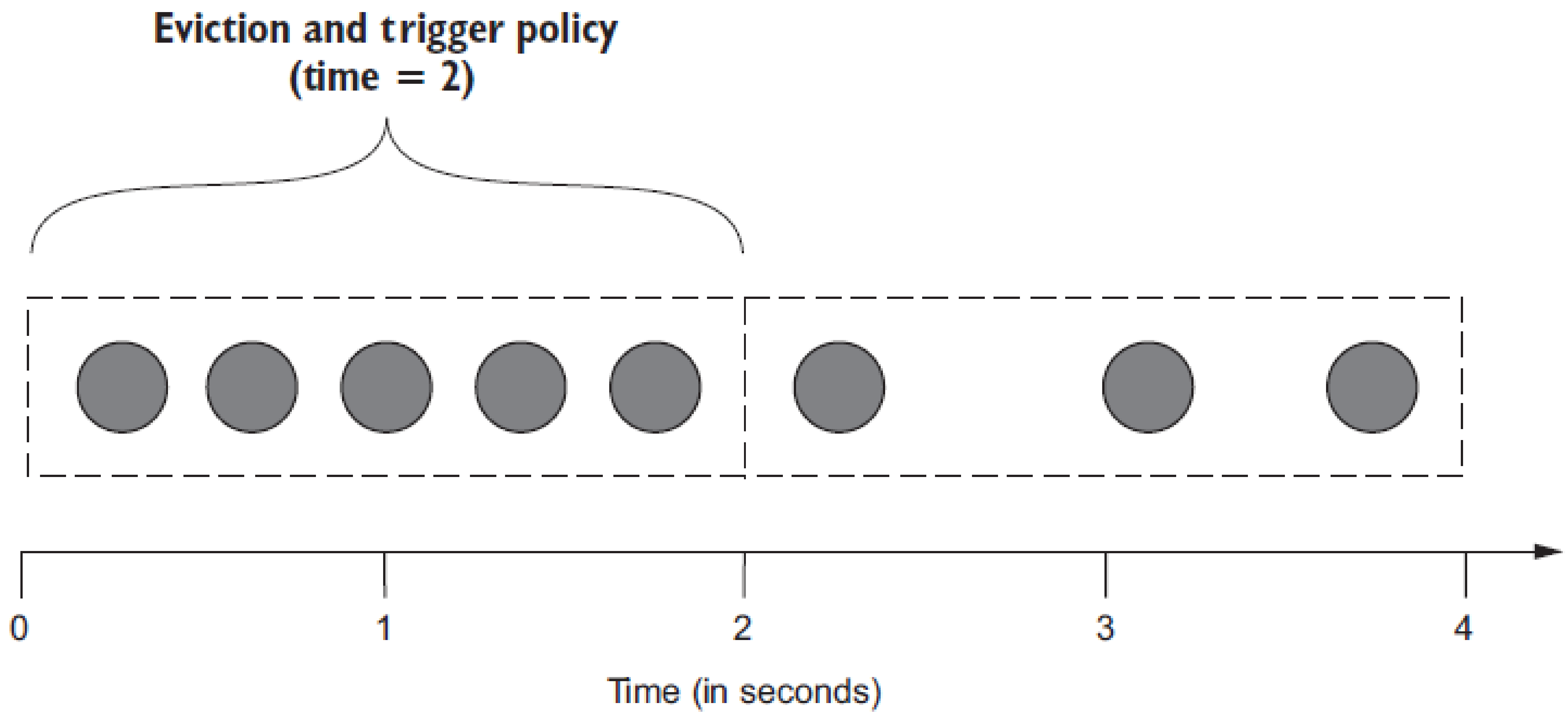
Framework	Sliding window	Event or stream time
Spark streaming	Yes	Stream time
Storm	No	N/A
Flink	Yes	Both
Samza	No	N/A

#2: Tumbling: Quantity policy

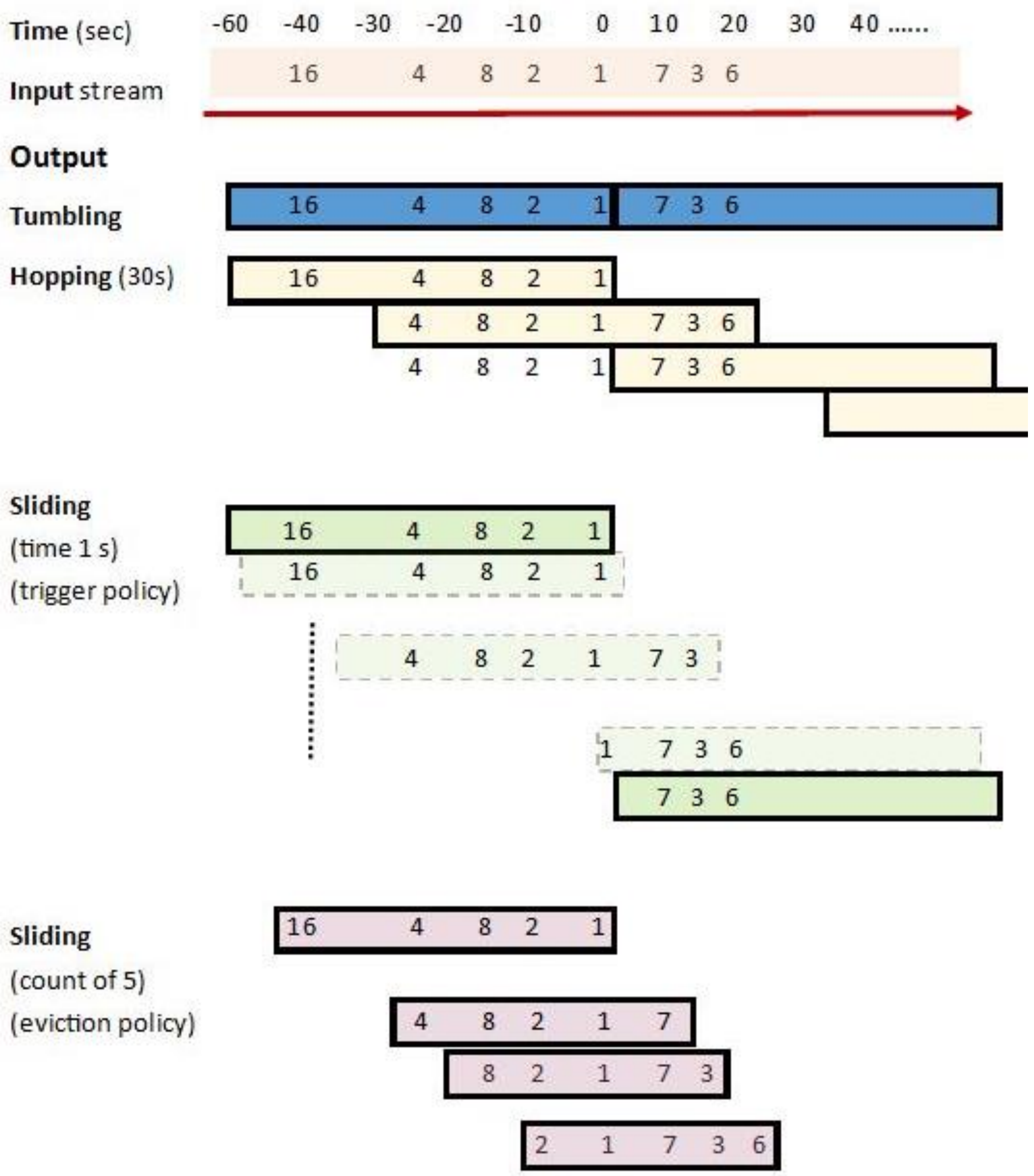
- Eviction policy: When full.
- Trigger policy: Count of items.
- 2 types: Count-based & temporal.



Count-based type



Temporal type



https://stackoverflow.com/questions/12602368/sliding-vs-tumbling-windows

Summary

- 2 strategies:
  - Sliding window (time-based)
  - Tumbling (quantity-based)
    - Count-based type
    - Temporal type

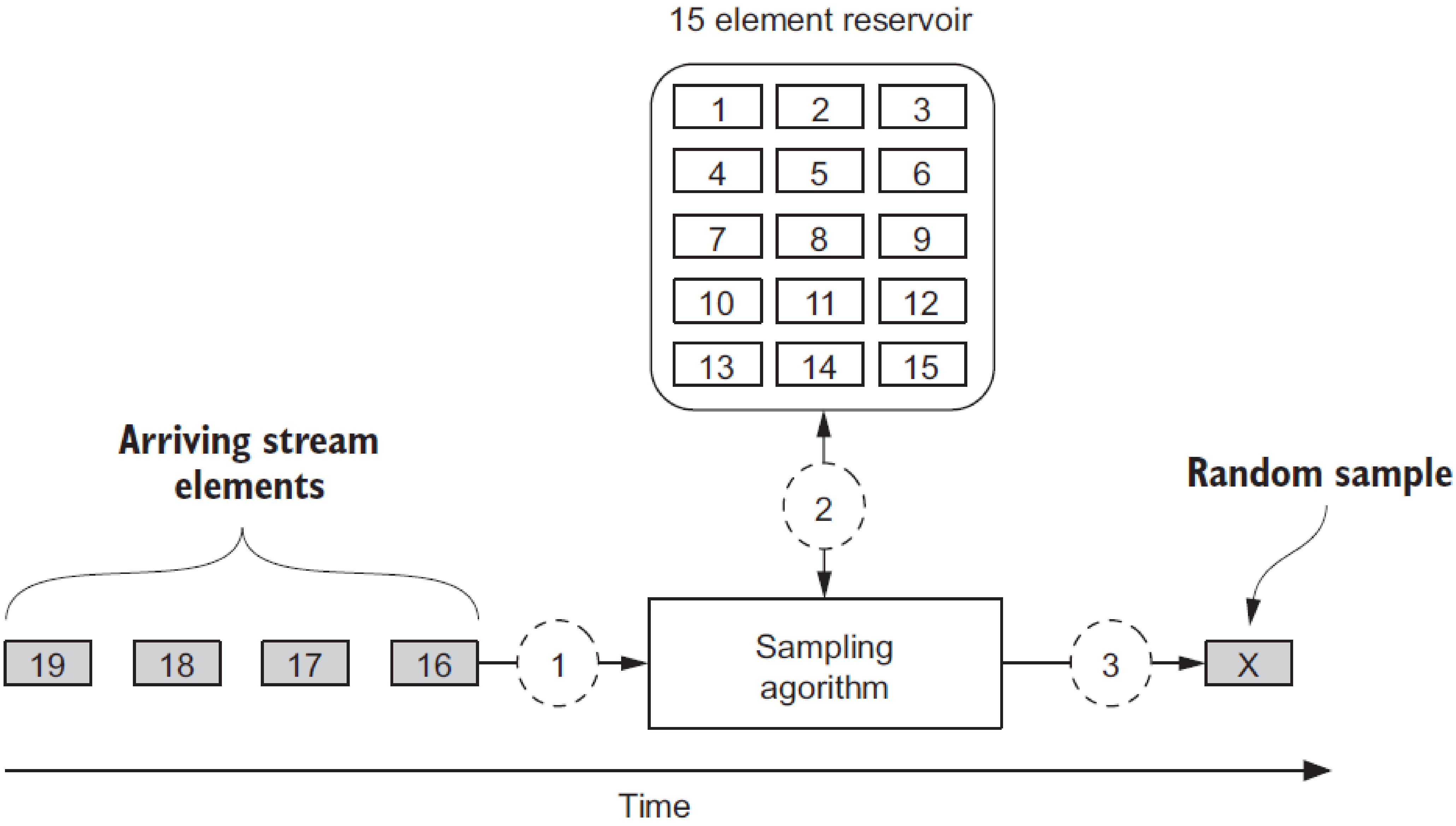
Case study: Fail2ban

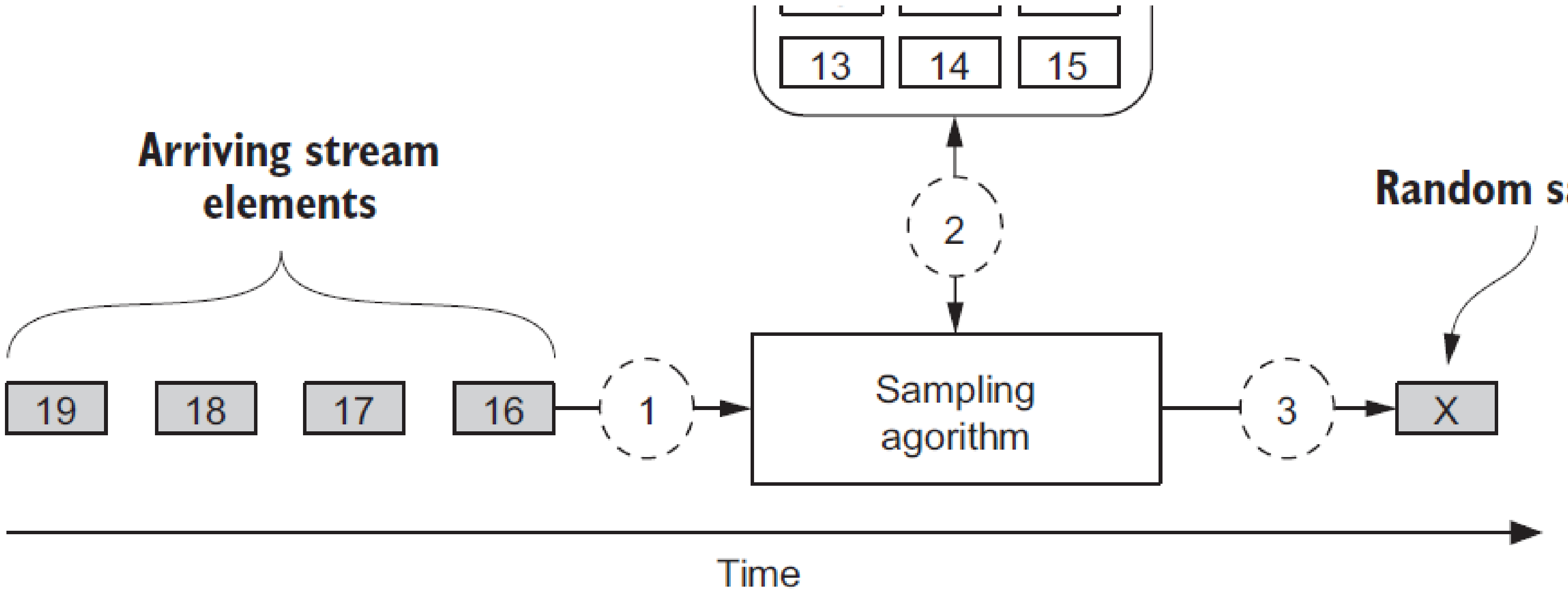
Summarization techniques

- Facts about streaming data:
  - The ending timestamp is unknown.
  - The data may not be perfectly suitable to find the best answer.
  - Then how do we know the facts about the data?

Problem #1: Random sampling

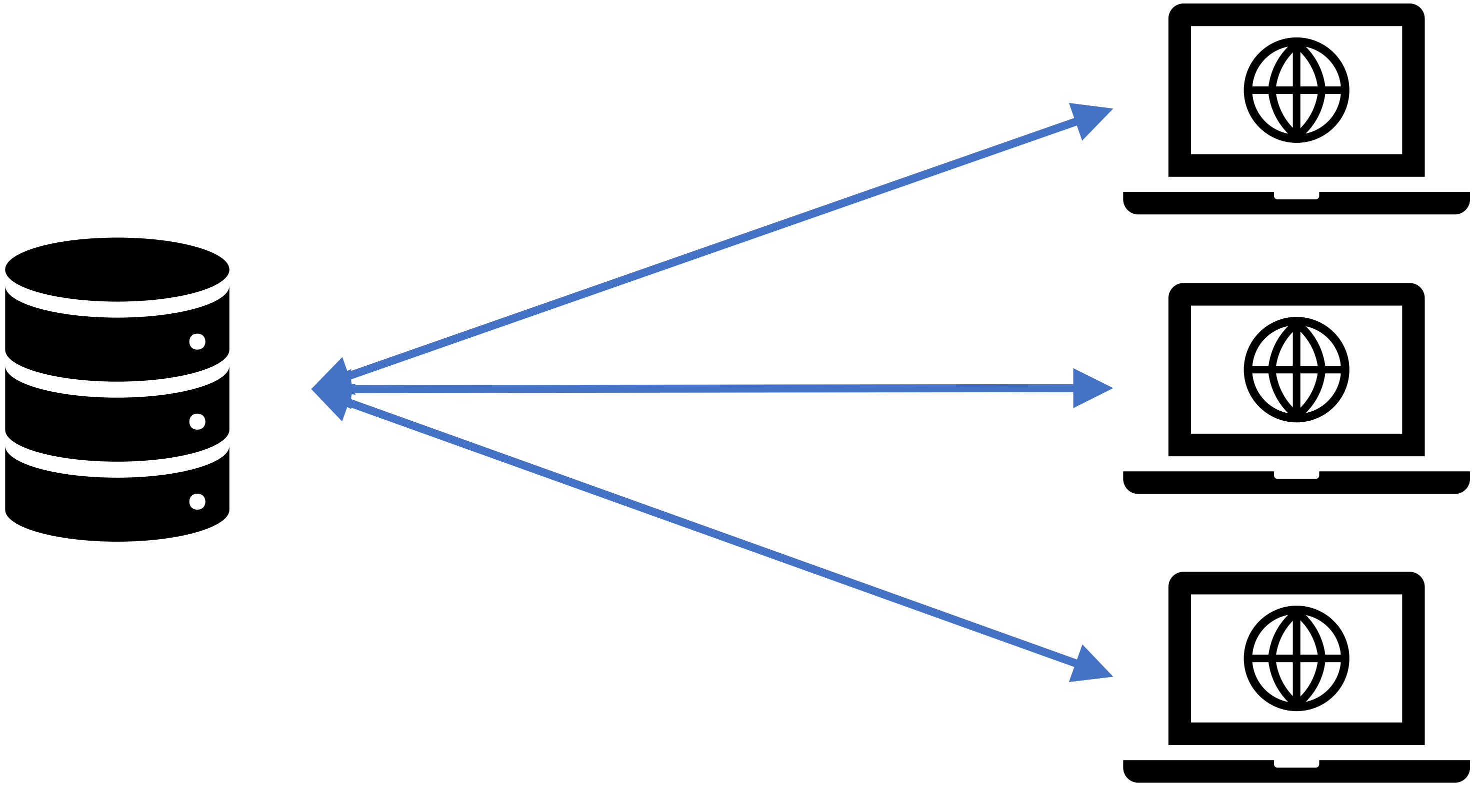
- Take a sample that is:
  - Random
  - Viable
- The idea is to use a reservoir that is always stores x elements.


15 element reservoir
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Arriving stream elements
19
18
17
16
1
2
Sampling agorithm
3
Random sample
X
Time



1. When the 16th item arrives, the propability that it will be added is $p = \frac{k}{n} = \frac{15}{16}$, given k and n are size of reservoir and the element number we are processing.
2. Flip a coin: Generate a random number r between 0 & 1. If $r > p$, we choose the data.
3. Store the data by replacing a random element in resertvoir with it.

Problem #2: Count distinct elements



How many distinct seen ads in the last minute?

Count-distinct problem (Cardinality estimation)

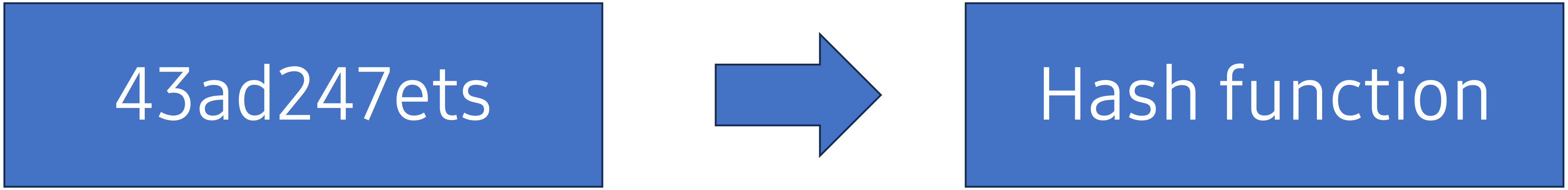
Probabilistic, again!

- Bit-pattern-based: Detect the patterns of bit at the beginning of the binary value of each element.
  - LogLog, HyperLogLog, HyperLogLog++
- Order statistics-based: Order statatics, such as the smallest values that appears.
  - MinCount, Bar-Yossef

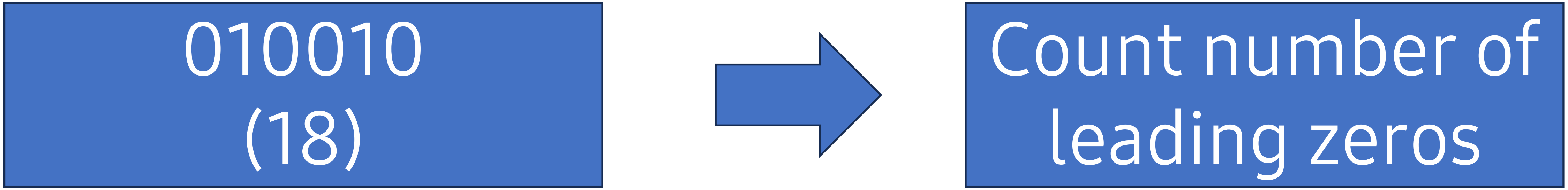
HyperLogLog

1. Get the ID of element.
2. Hash the ID into hash function.
3. Convert into binary string, determine which **register value** (bin) to update and **the value** to update with.
   - Use n (precision) least/most significant bits.
   - $n = \log_2 m$
4. Determine the number of leading zeros v at the right of index at (3), and add 1 to it.
5. Update index at (3) with the value of v.

Example

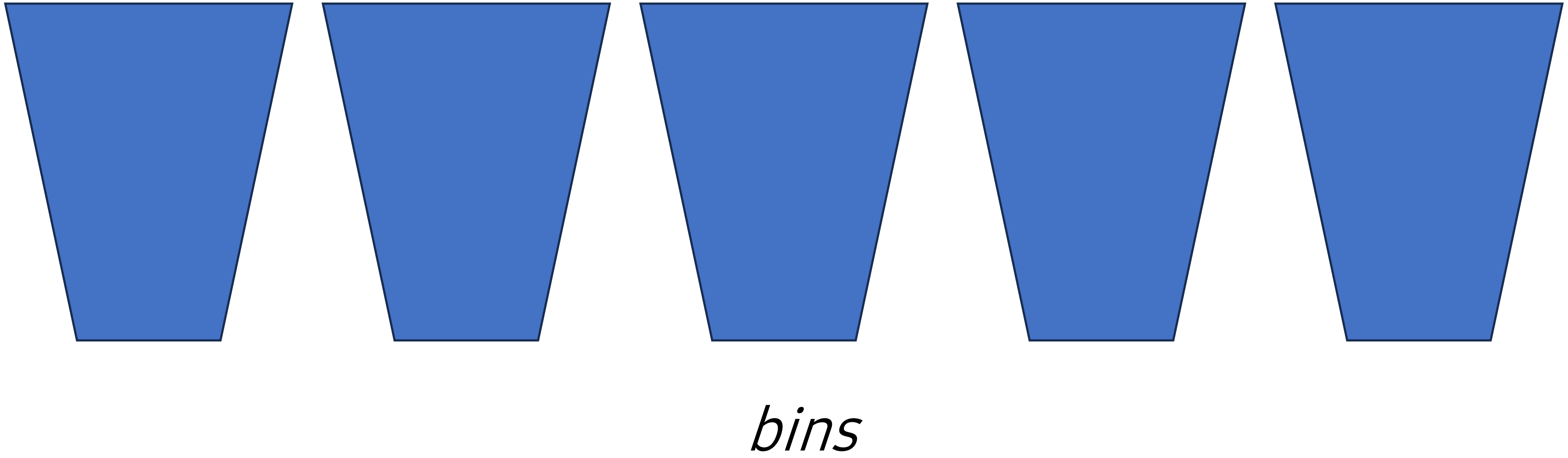


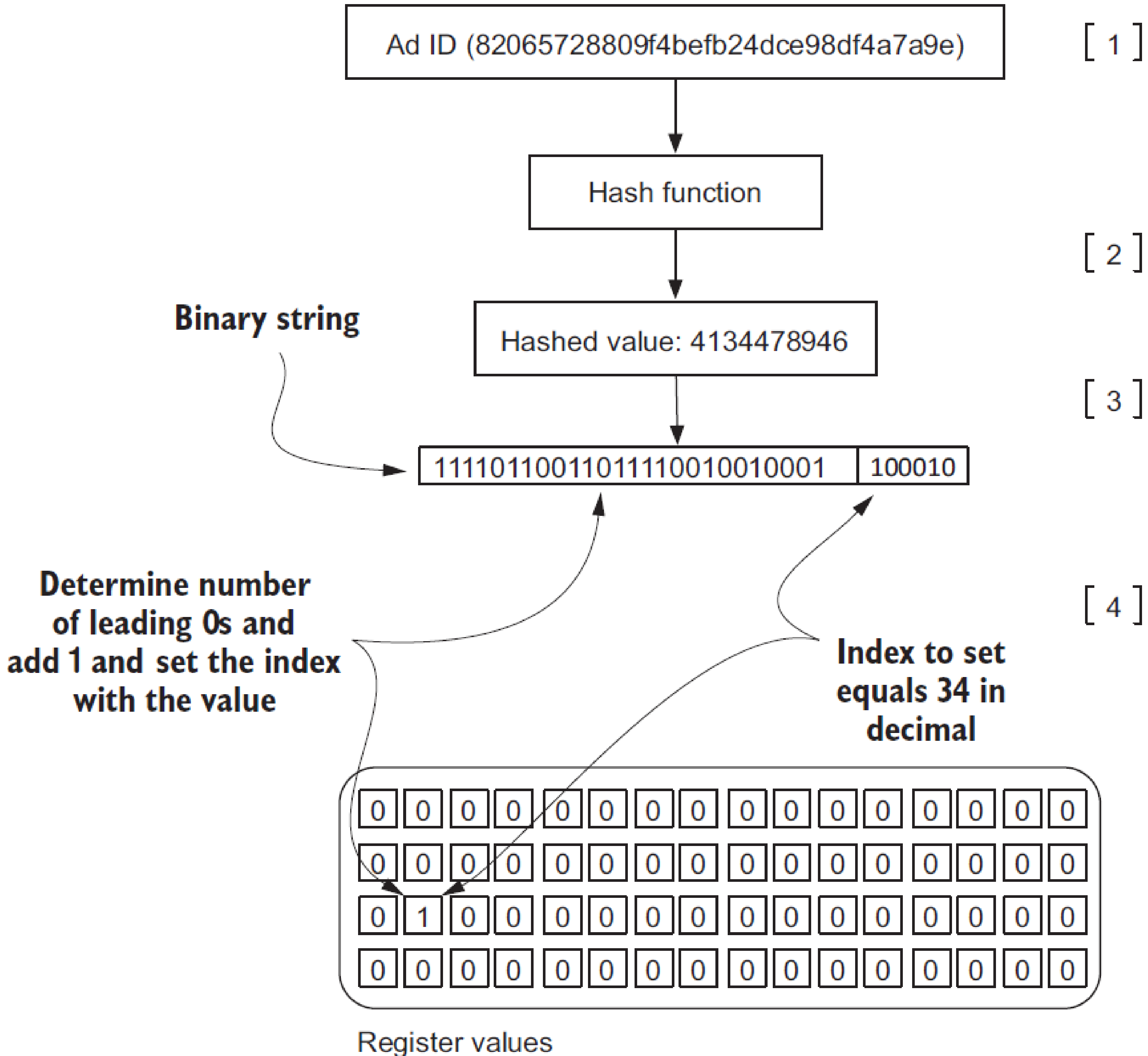
https://cryptii.com/


010010
(18)
Count number of
leading zeros

101101 101011 010100 010100 110101 001001
001100 100100 111111 001111 100011 011100
010010 111110 011011 000001 010111 001101
110001 101001 010100 10

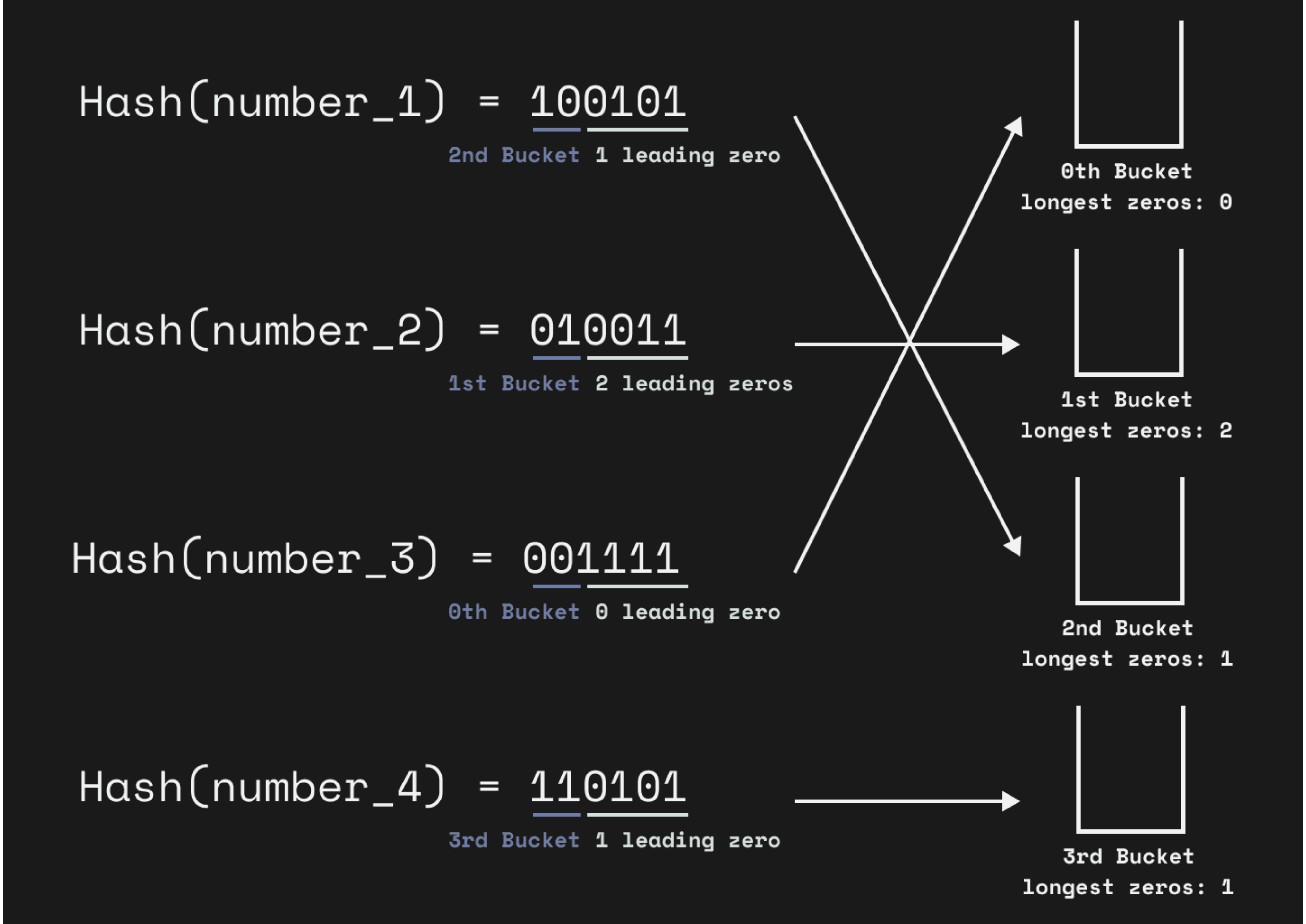
$$bins[18] = Max(bins[18], 1 + 1) = 2$$





Ad ID (82065728809f4befb24dce98df4a7a9e)
[1]
Hash function
[2]
Binary string
Hashed value: 4134478946
[3]
11110110011011110010010001
100010
Determine number of leading 0s and add 1 and set the index with the value
[4]
Index to set equals 34 in decimal
Register values

$$Cardinality = m.2^{\frac{\sum_i bins[i]}{m}}$$


Hash(number_1) = 100101
2nd Bucket 1 leading zero
Hash(number_2) = 010011
1st Bucket 2 leading zeros
Hash(number_3) = 001111
0th Bucket 0 leading zero
Hash(number_4) = 110101
3rd Bucket 1 leading zero
0th Bucket
longest zeros: 0
1st Bucket
longest zeros: 2
2nd Bucket
longest zeros: 1
3rd Bucket
longest zeros: 1

Homework (group)

- Demonstrate HyperLogLog using Python.