Cloud Computing

Big Data Processing

$whoami

Carlo Puliafito

- Assistant Professor

Department of Information Engineering (DII),
University of Pisa

- Office 5-049 (6th floor), Largo Lucio Lazzarino 1
- Email: carlo.puliafito@unipi.it
- No pre-set office hour; please, drop me an email ☺

Big Data: Use cases

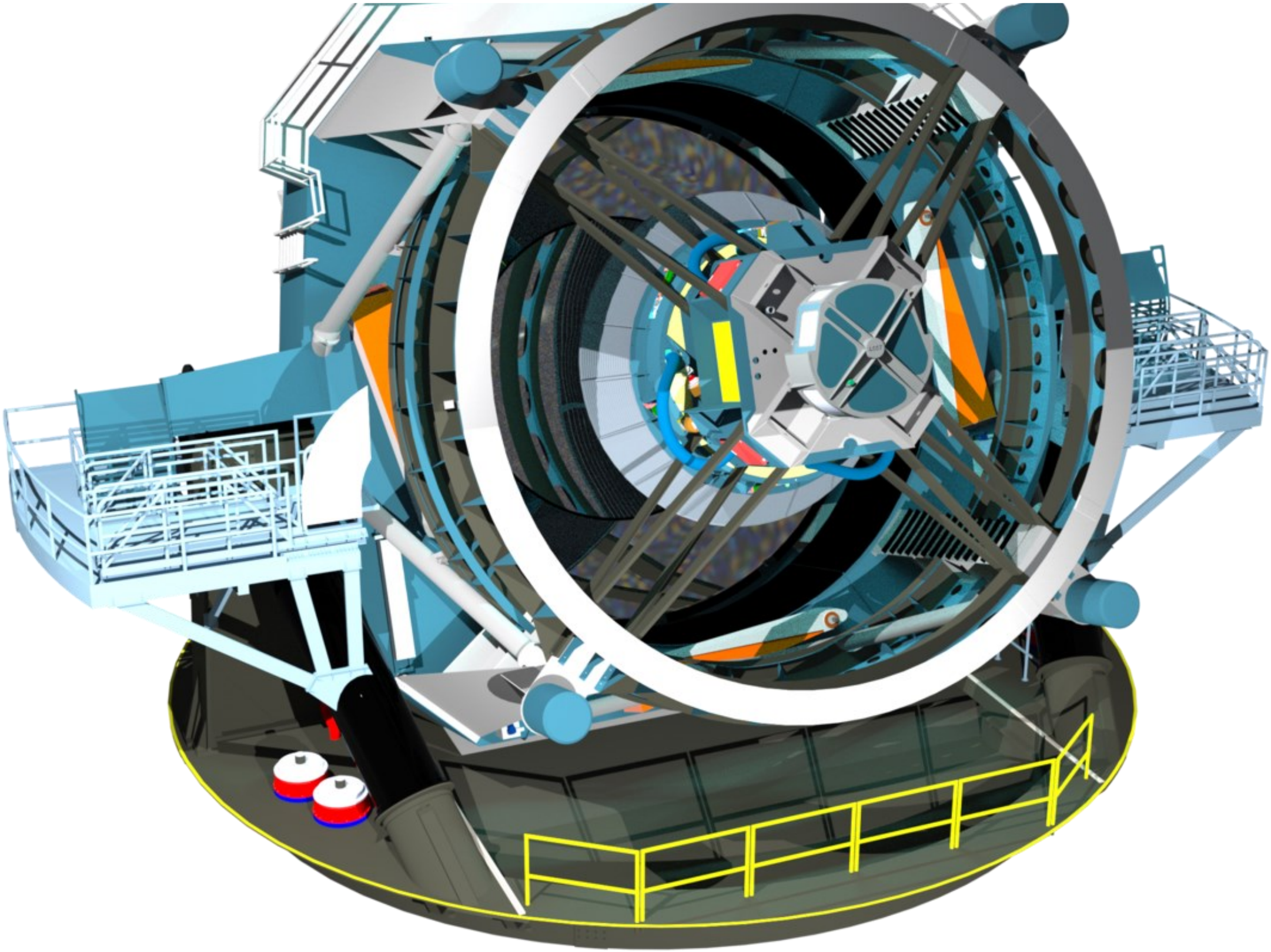
Large Hadron Collider
15 Petabytes (15×10^{15} bytes) of data a year

Physics

Source: https://upload.wikimedia.org/wikipedia/commons/e/eb/Views_of_the_LHC_tunnel_sector_3-4,_tirage_1.jpg

Rubin Observatory (Chile)
3.2 gigapixel primary camera – 0.5 petabytes per month

Astronomy



Next-generation DNA sequencing
1 gram of DNA can hold up to 215 petabytes of information

Medicine

Source: https://wyss.harvard.edu/news/save-it-in-dna/

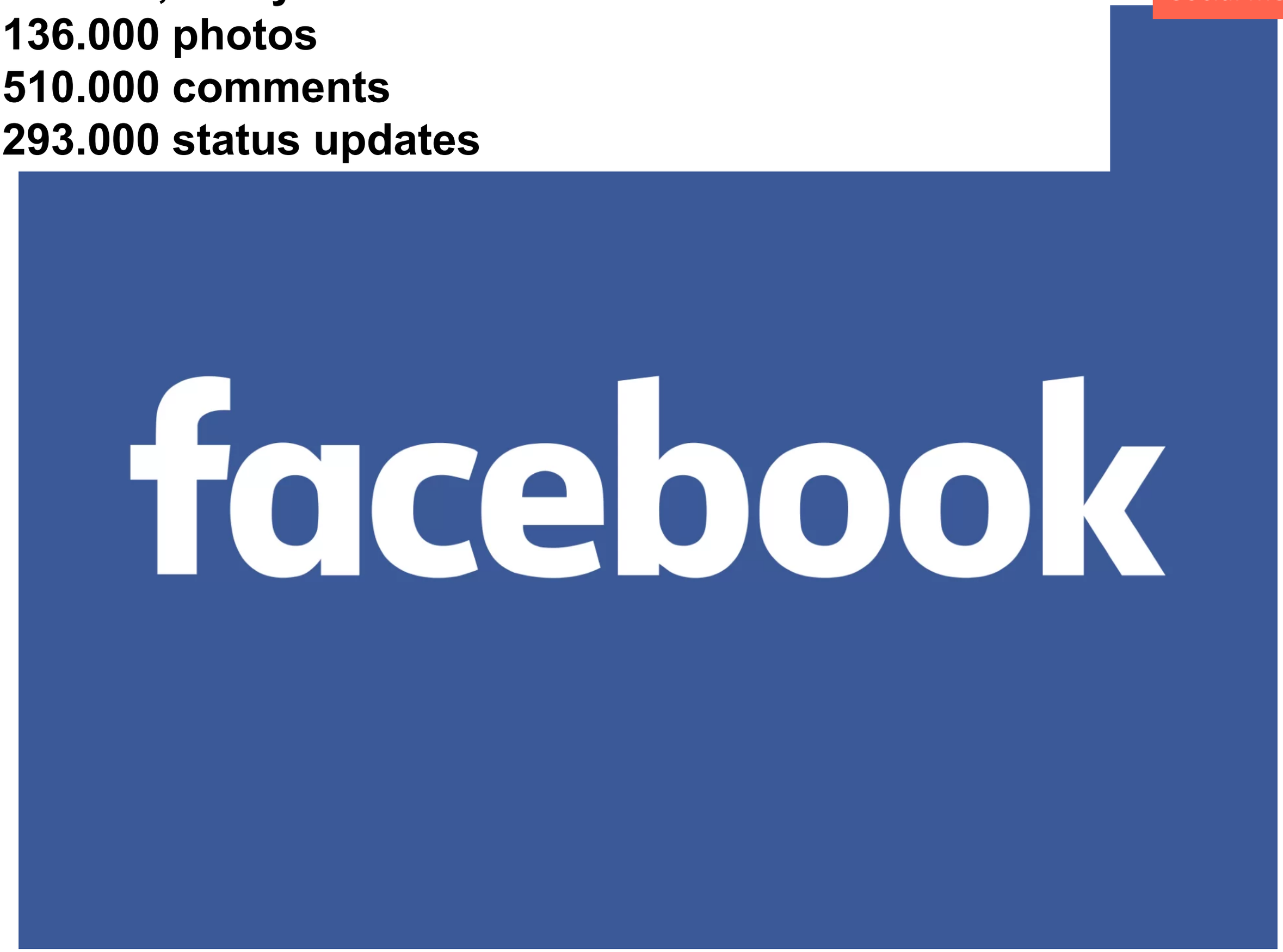
Connected humans: Web & Social media

- Google carries 4,497,420 searches
- Youtube users view 4,500,000 videos
- Twitter users post 511,200 tweets
- Instagram users post 277,777 tales
- Skype users make 231,840 calls
- Uber users take 9,772 drives
- Netflix users stream 694,444 hours of video
- Giphy toils up 4,800,000 gifs
- Airbnb books 1,389 reservations
- Tinder users swipe 1,400,000 times
- Twitch users view 1,000,000 videos

Social media

Facebook, every 60 seconds:

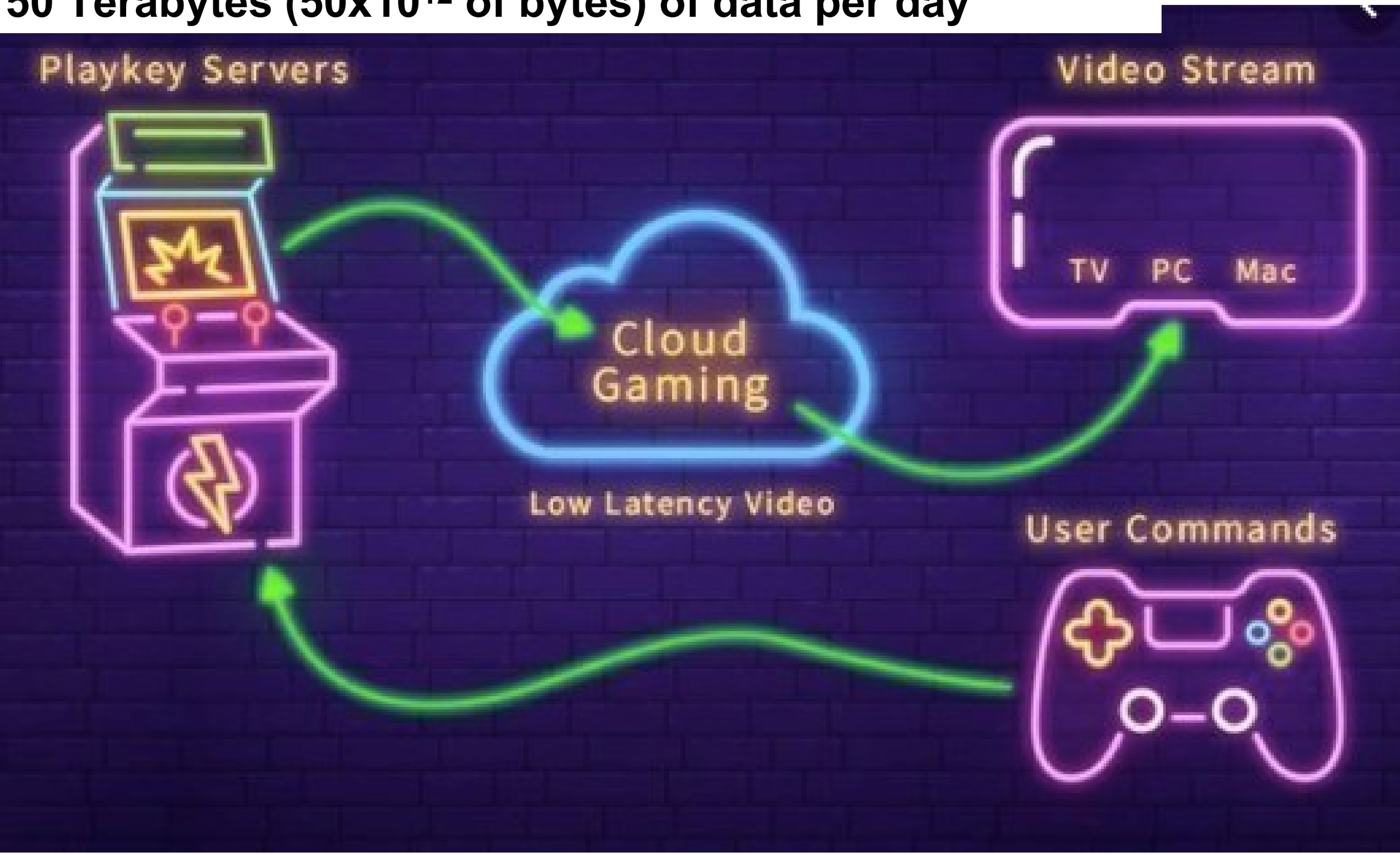
- **136.000 photos**
- **510.000 comments**
- **293.000 status updates**



Source: https://www.simplilearn.com/how-facebook-is-using-big-data-article

Over 2 billion gamers per day = 50 Terabytes (50×10^{12} of bytes) of data per day

Games



Source: https://www.datatrained.com/post/the-use-of-big-data-in-the-video-game-industry/

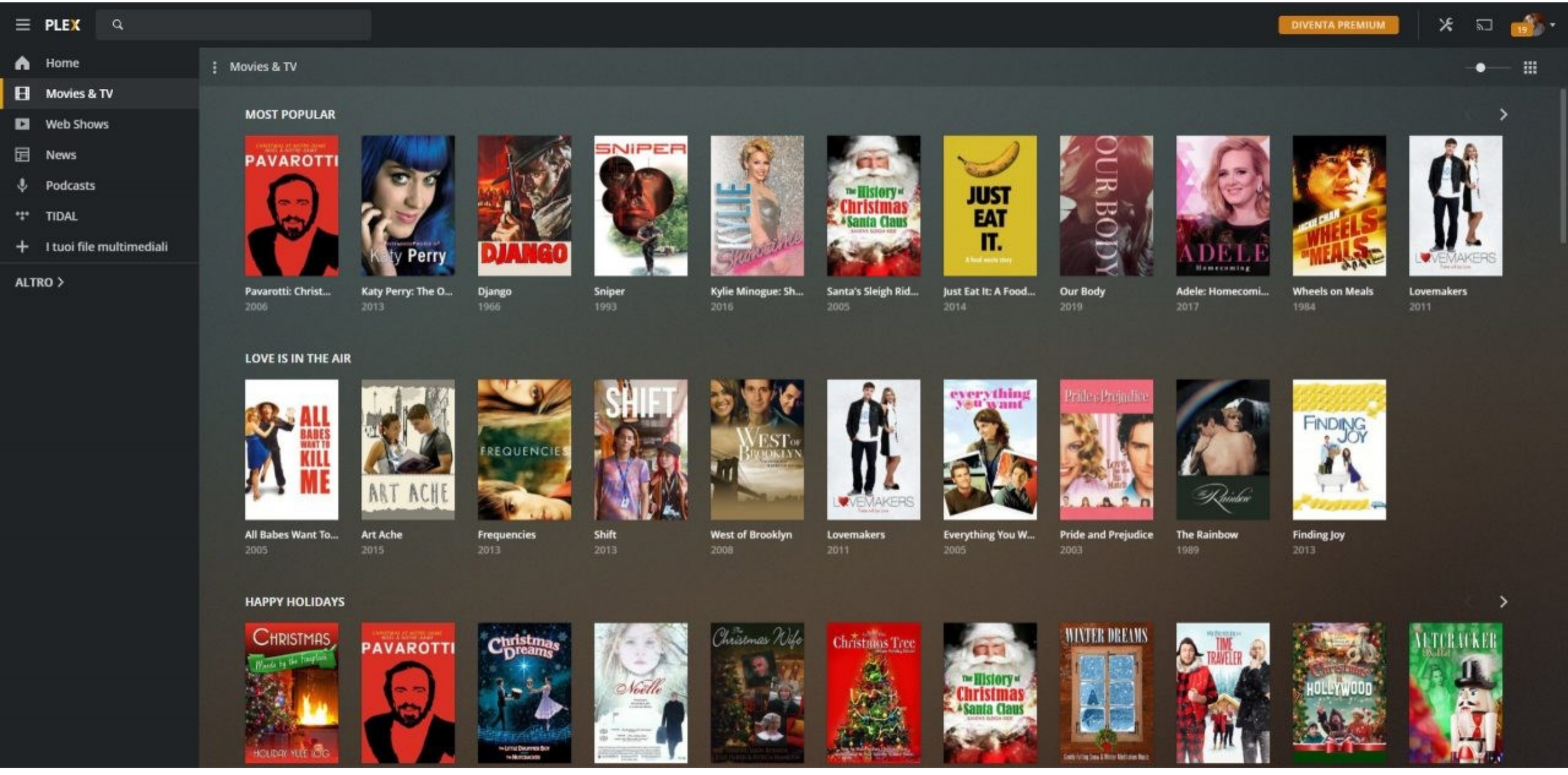
Finance

Source: https://upload.wikimedia.org/wikipedia/commons/1/18/NASDAQ_studio.jpg

Events

Source: https://upload.wikimedia.org/wikipedia/commons/2/24/Huge_crowd_turns_out_for_MTV_EXIT_concert_against_human_trafficking_and_exploitation.jpg

Videos

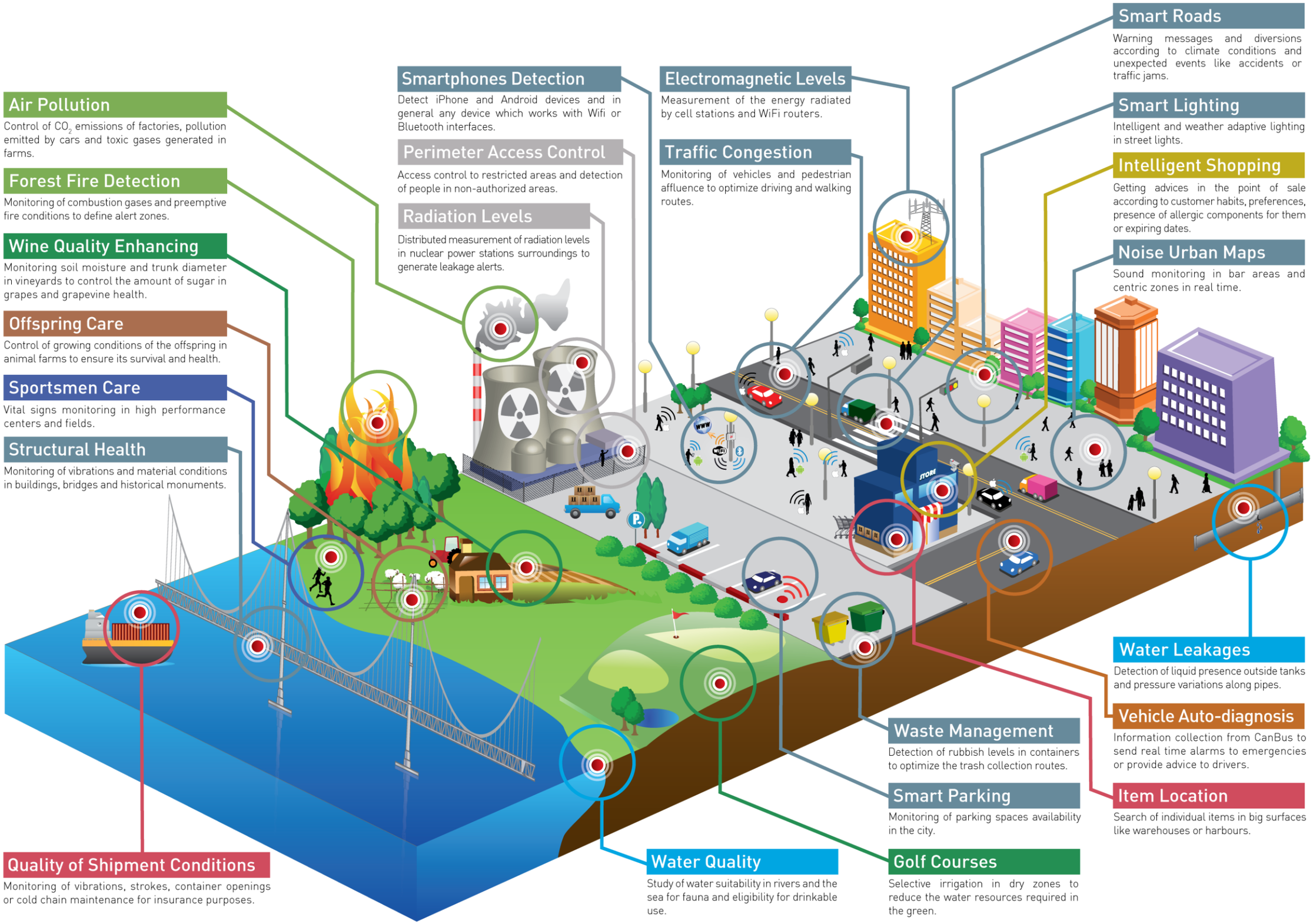


Source: https://www.smartworld.it/wp-content/uploads/2019/12/plex-movie-e-tv-1280x625.jpg

Connected objects: Internet of Things

- 400 million IoT devices with cellular connections at the end of 2016
- 1.5 billion IoT devices with cellular connections in 2022
- 144.4 million smart homes in the US
- 53 billion US dollars for the global smart home market by 2022

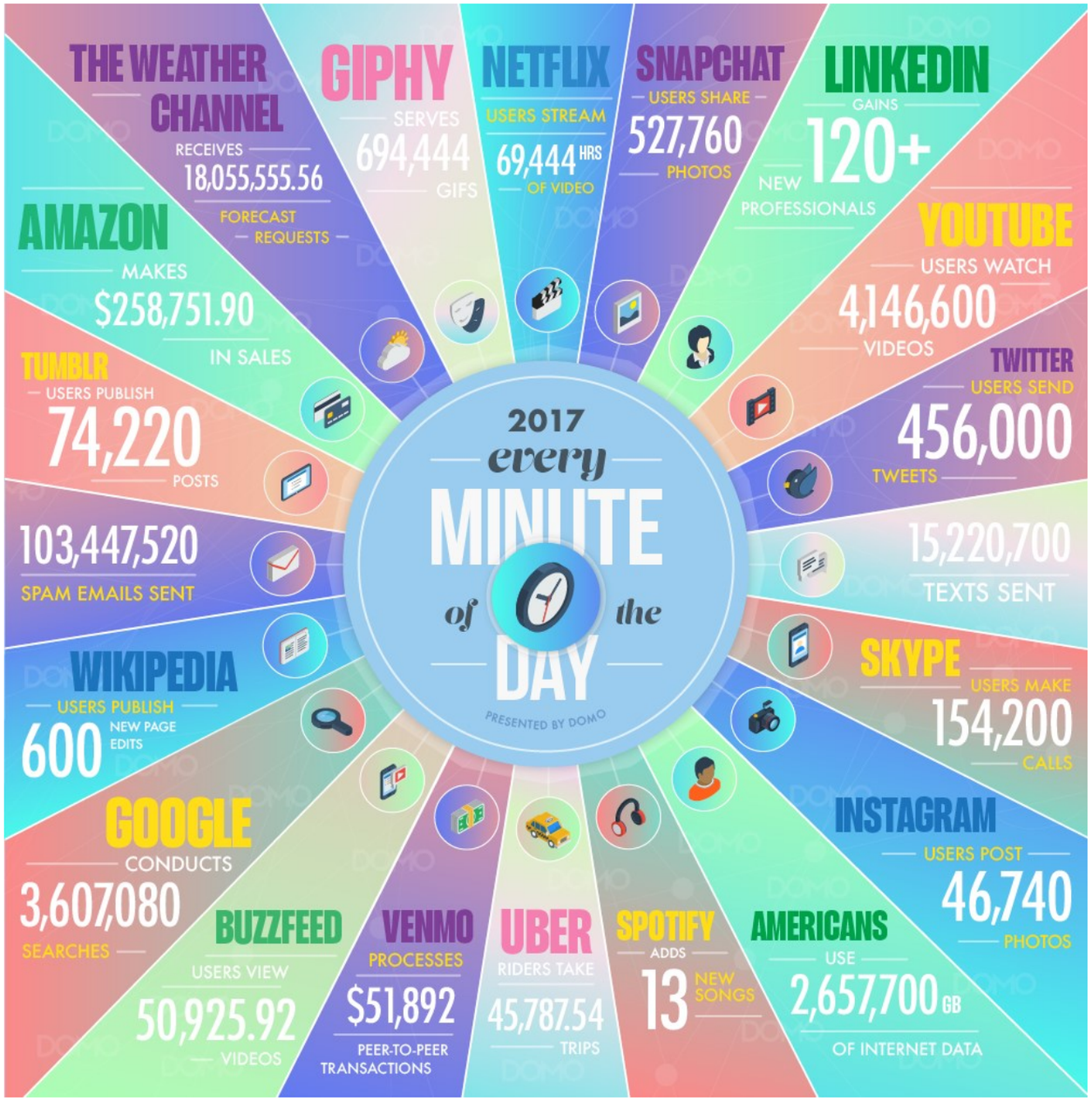
Smart Cities and Communities

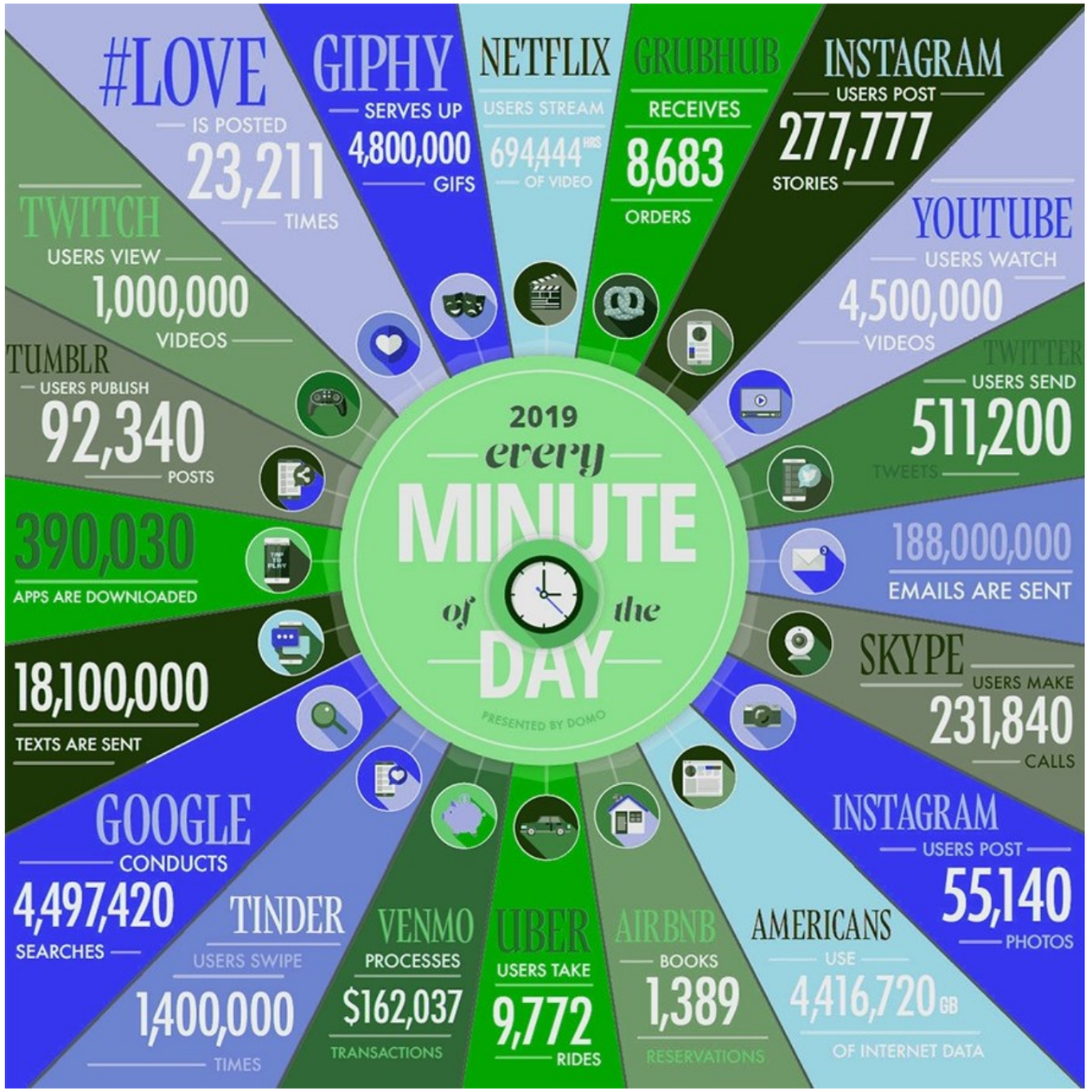


Source: http://www.libelium.com/wp-content/themes/libelium/images/content/applications/libelium_smart_world_infographic_big.png

How much data in the world?

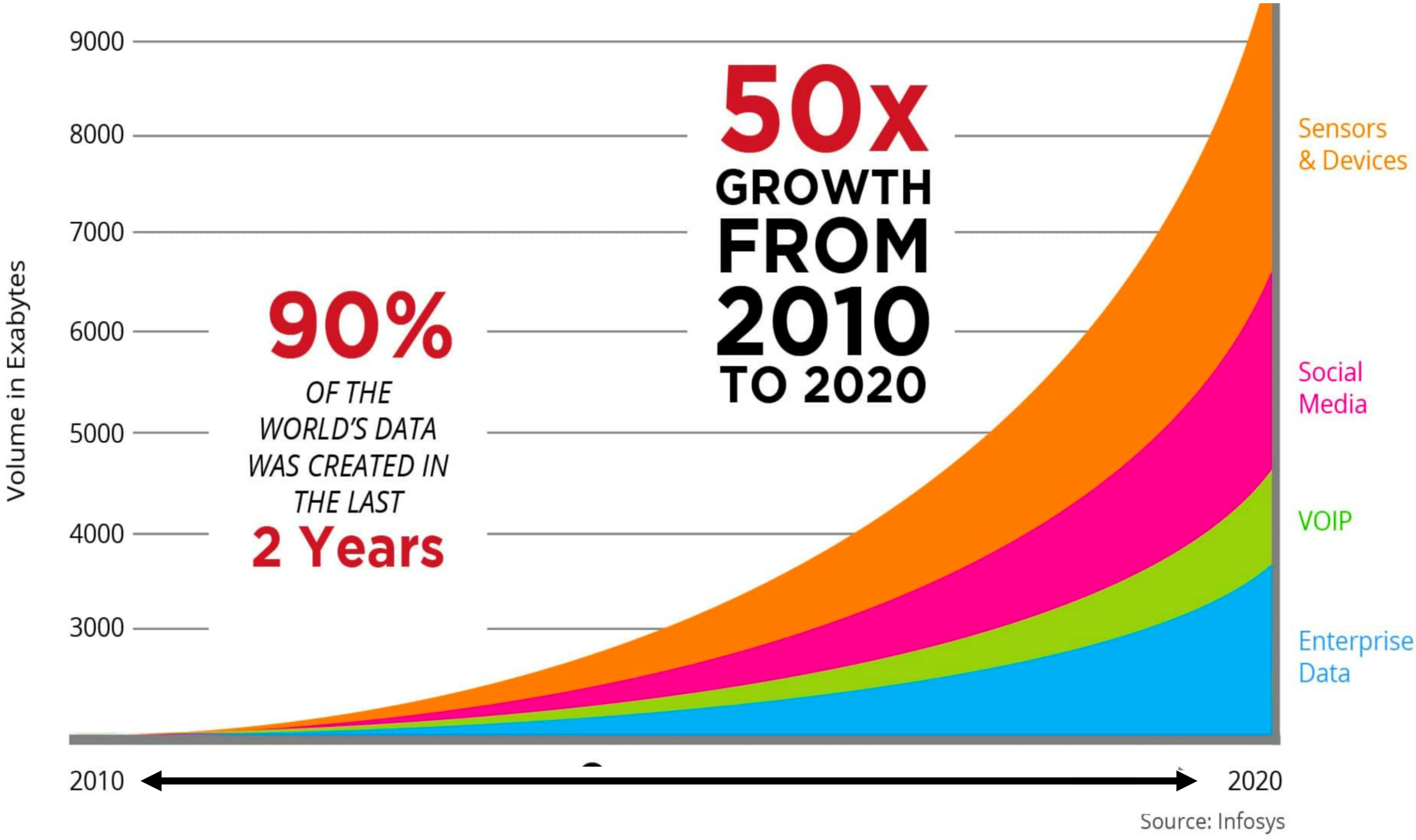
Source: https://sparkdatabox.com/blog/wp-content/uploads/2019/10/IMG-20191017-WA0004-1.jpg



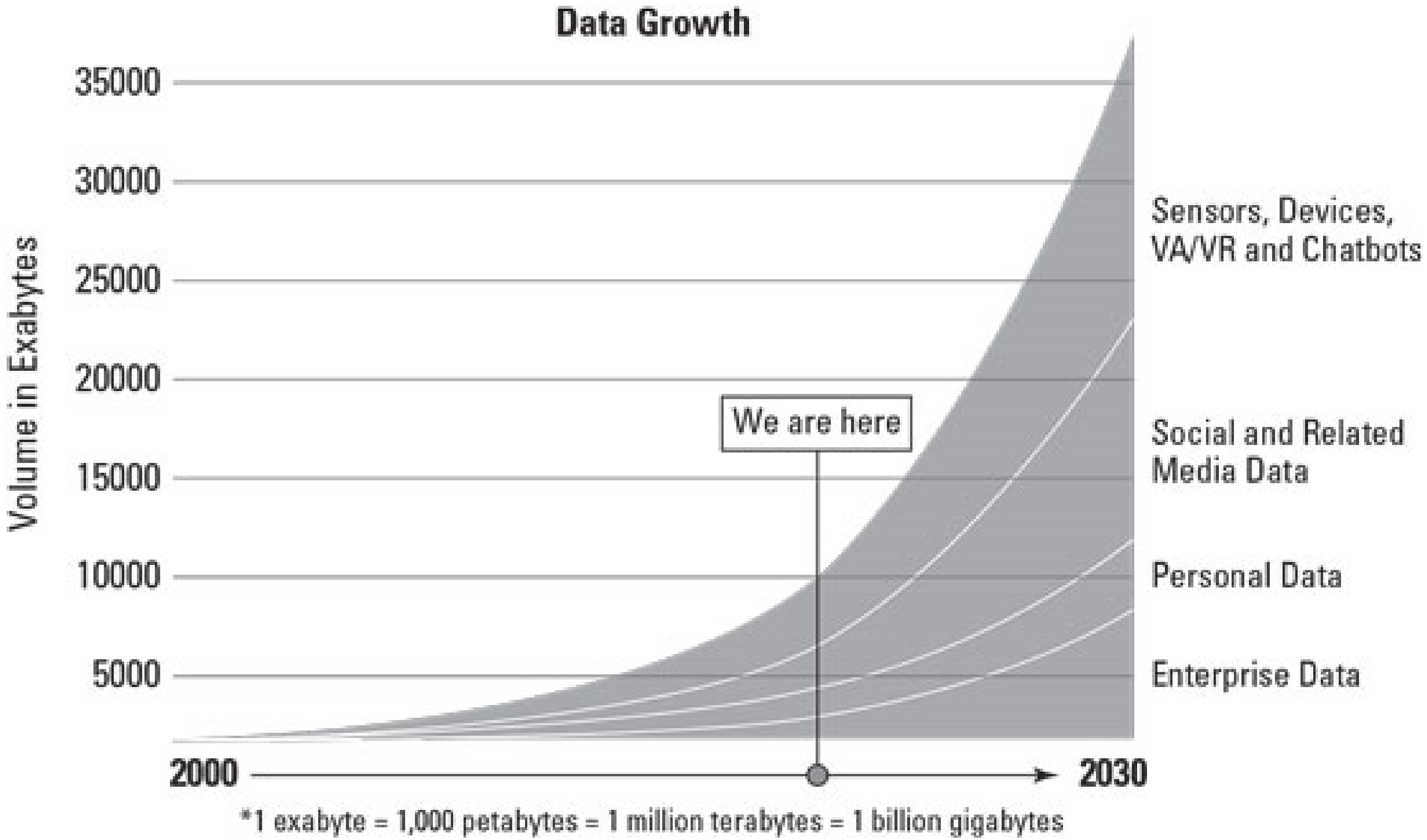


Source: https://web-assets.domo.com/blog/wp-content/uploads/2017/07/17_domo_data-never-sleeps-5-01.png

1 Exabyte = 10^{18} bytes



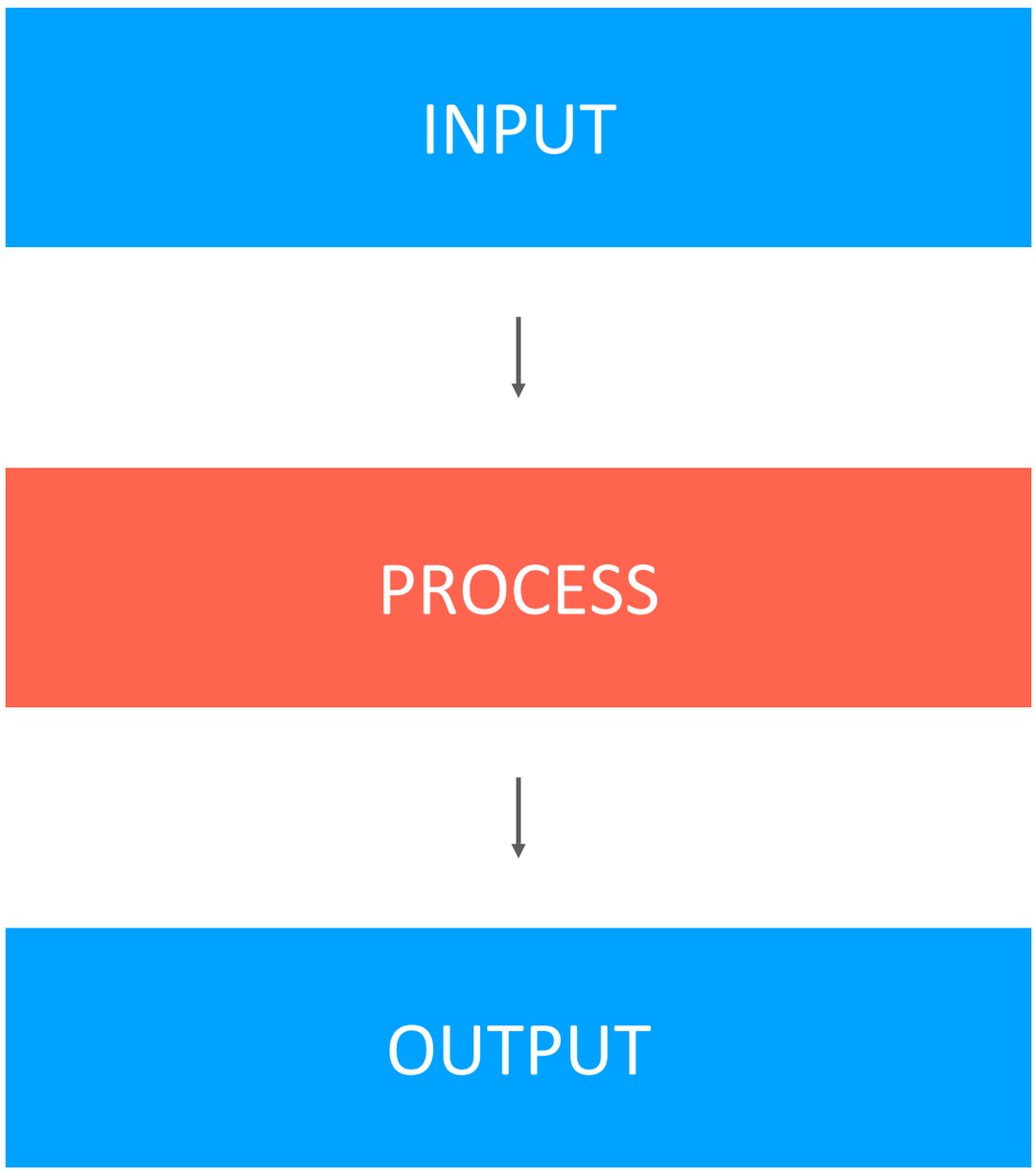
Source: https://therbootcamp.github.io/ML_2019May/_sessions/LookAhead/LookAhead.html#3



Source: https://www.dummies.com/article/technology/information-technology/ai/general-ai/how-government-and-nonprofits-battle-budgets-with-ai-272656/

Preliminaries on Parallel Programming

Typical Application



What if?

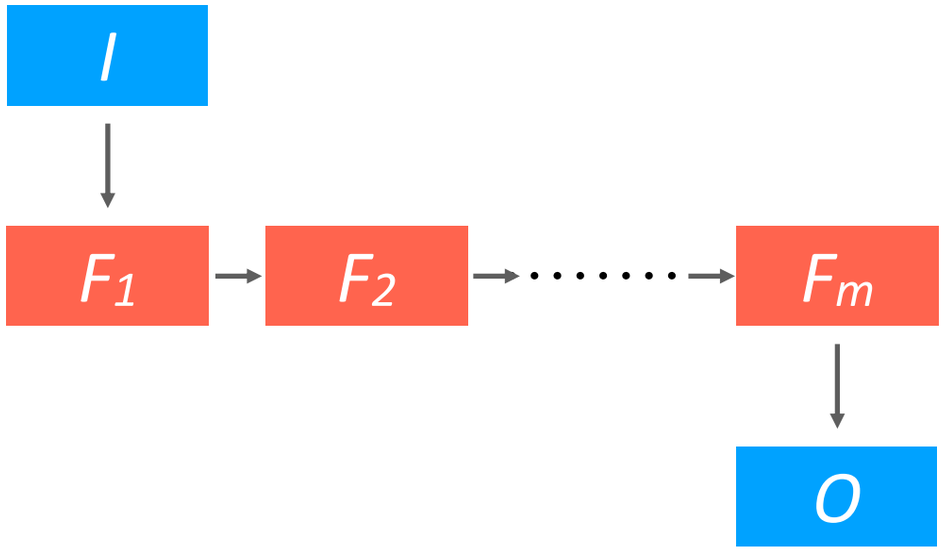
INPUT

PROCESS

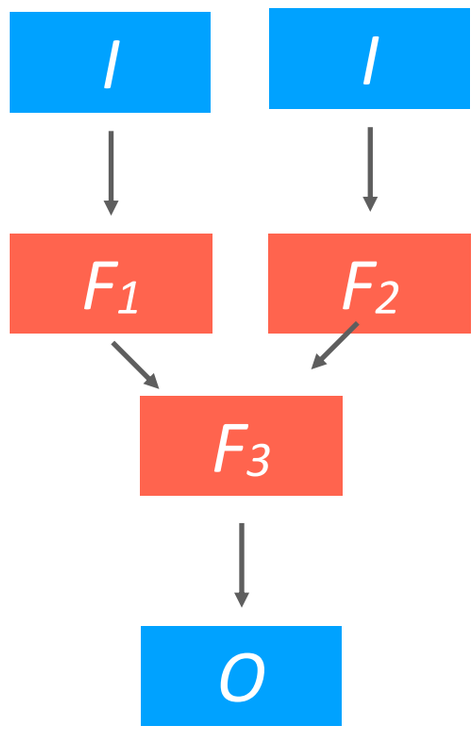
OUTPUT

Task parallelism

- Sequential algorithm
  - A function *F* is applied on a data structure *I* to yield *O*
  - $O = F(I)$



- Task parallelism
  - Partition on control
  - Pipeline
    - $O = F(I) = F_m(F_{m-1}(\ldots F_1(I)))$
  - Task graph
    - $O = F(I) = F_3(F_2(I), F_1(I))$

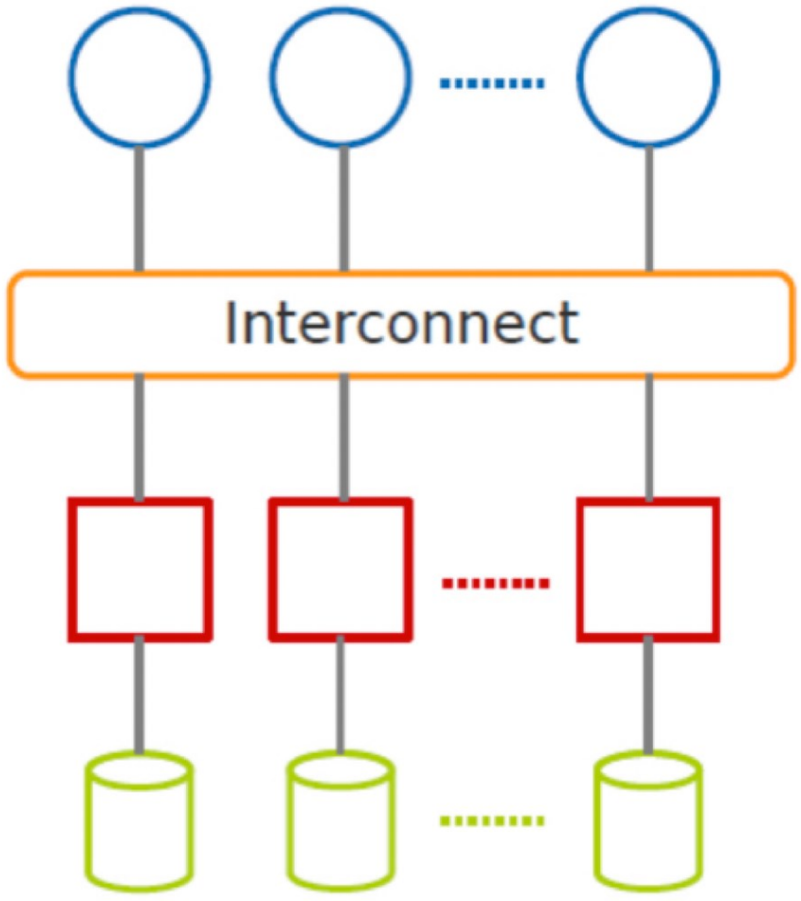


Programmer problems: splitting code

- How to divide code into parallel tasks?
- How to distribute the code?
- How to coordinate the execution?
- How to load the data?
- How to store the data?
- What if more tasks than CPUs?
- What if a CPU crashes?
- What if a CPU is taking too long?
- What if the CPUs are different?

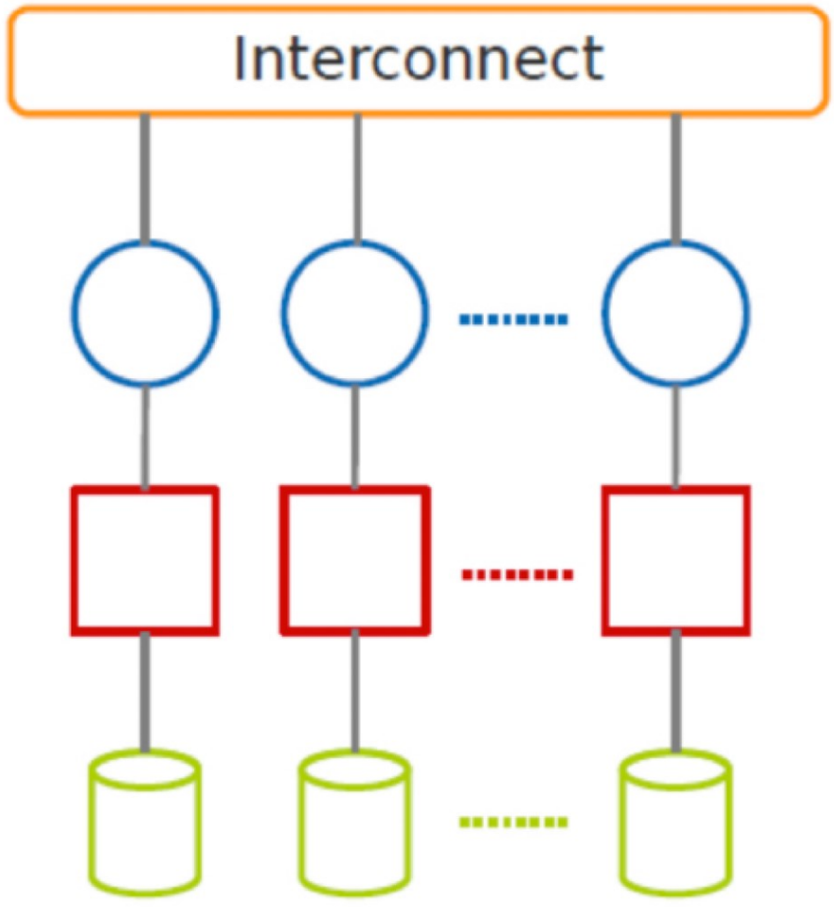
Parallel Architectures

Shared Memory



- Posix Threads
- OpenMP

Message Passing



- Sockets
- MPI - Message Passing Interface

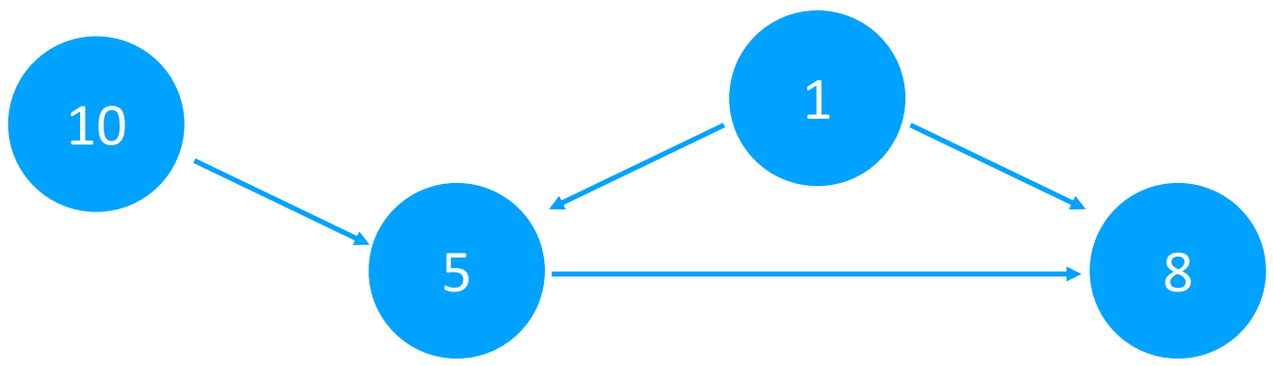
DeWitt and Gray, “Parallel database systems: the future of high performance database systems”, ACM Communications, 1992

Designing Parallel Algorithms

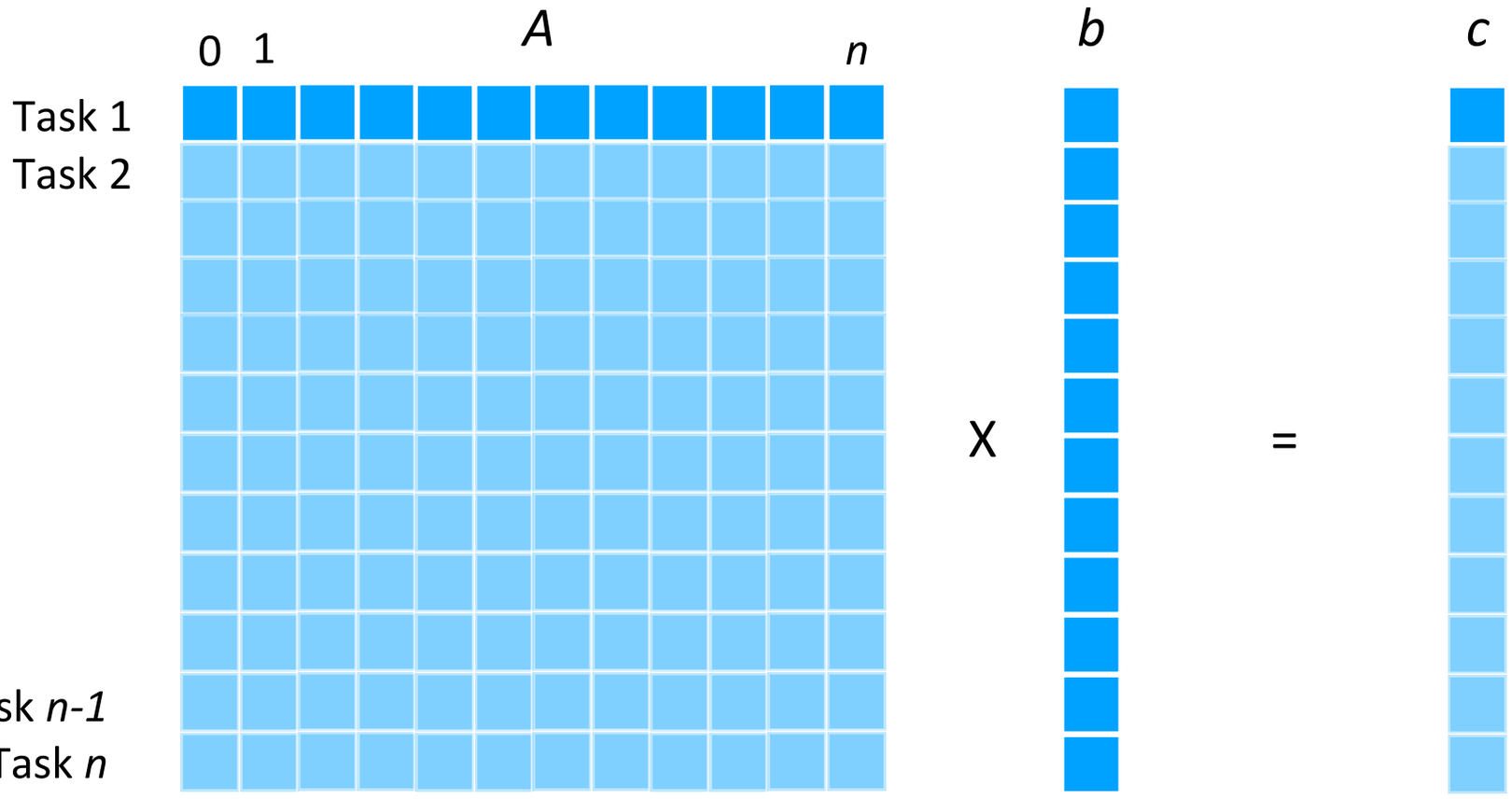
- Typical steps:
  - Identify what pieces of work can be performed concurrently
  - Partition concurrent work onto independent processors
  - Distribute a program's input, output, and intermediate data
  - Coordinate accesses to shared data: avoid conflicts
  - Ensure proper order of work using synchronization
- Some steps can be omitted
  - For shared memory parallel programming model, there is no need to distributed data
  - For message passing parallel programming model, there is no need to coordinate shared data

Tasks Dependency Graph

- The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently
- A given problem may be decomposed into tasks in many different ways.
  - Tasks may be of same, different, or even indeterminate sizes/granularities
- Decomposition modelled/illustrated in the form of Task Dependency Graph (TDG)
  - Directed Acyclic Graph (DAG)
  - Nodes = tasks
  - Directed edges = control dependency among tasks
  - Node labels = computational size / weight of the task



Example



- The computation of each element of the output vector *c* is independent of other rows in *A*. Based on this, a dense matrix-vector product can be decomposed into *n* tasks
- The figure highlights the portion of the matrix and vector accessed by task 1
- While tasks share data (the vector *b*), they do not have any dependencies
- No task needs to wait for the (partial) completion of any other task
- All tasks are of the same size in terms of number of operations.

Example

- Consider the execution of the query:

```
SELECT * FROM T WHERE
Model = "CIVIC" AND Year = "2001" AND
(Color = "GREEN" OR Color = "WHITE")
```

- on the following table T:

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Example

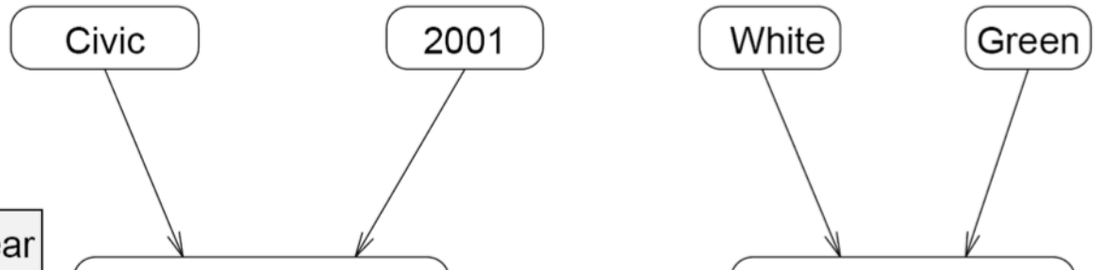
- The execution of the query can be divided into subtasks in various ways
- Each task can be thought of as generating an intermediate table of entries that satisfy a particular clause

ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green



ID#	Model	Year
6734	Civic	2001
4395	Civic	2001

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

ID#	Model	Year	Color
6734	Civic	2001	White

Example

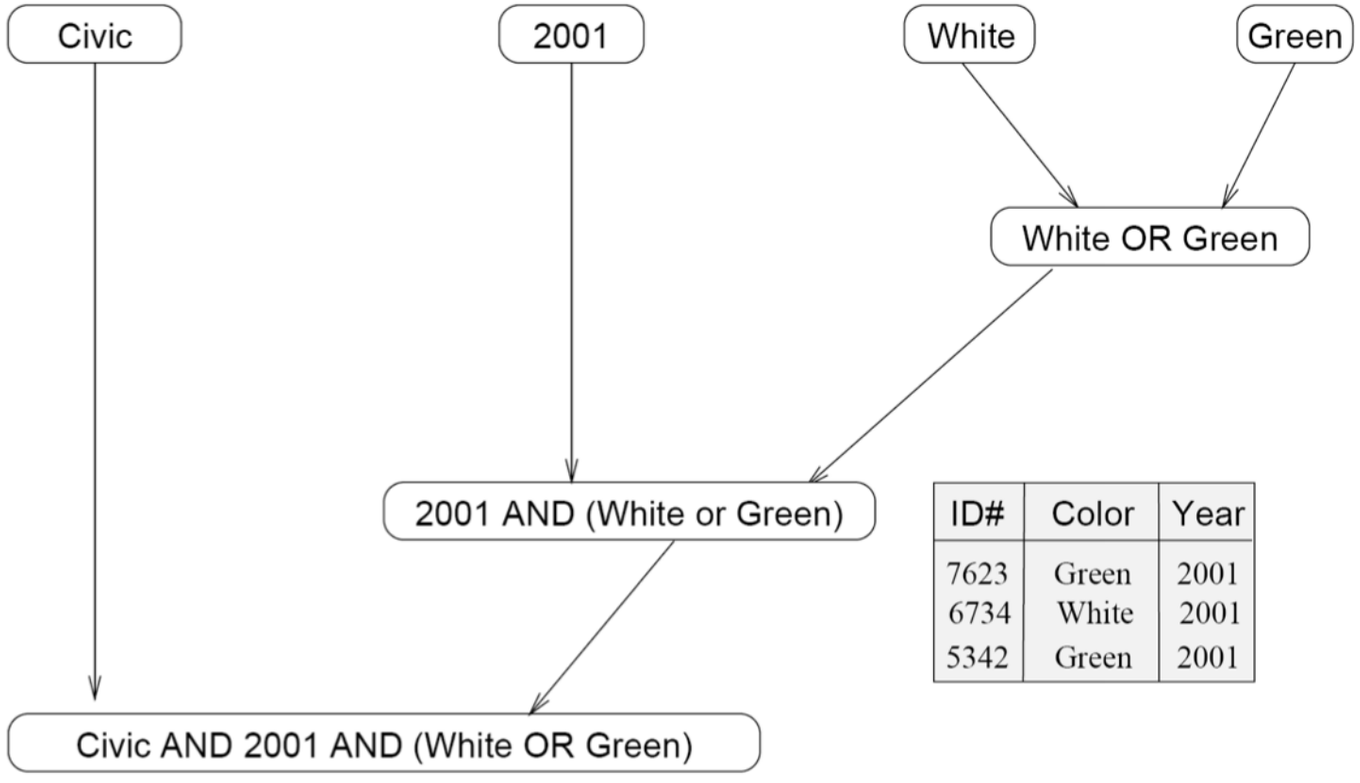
- The same problem can be decomposed into subtasks in other ways as well
- This because the AND operation is associative

ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

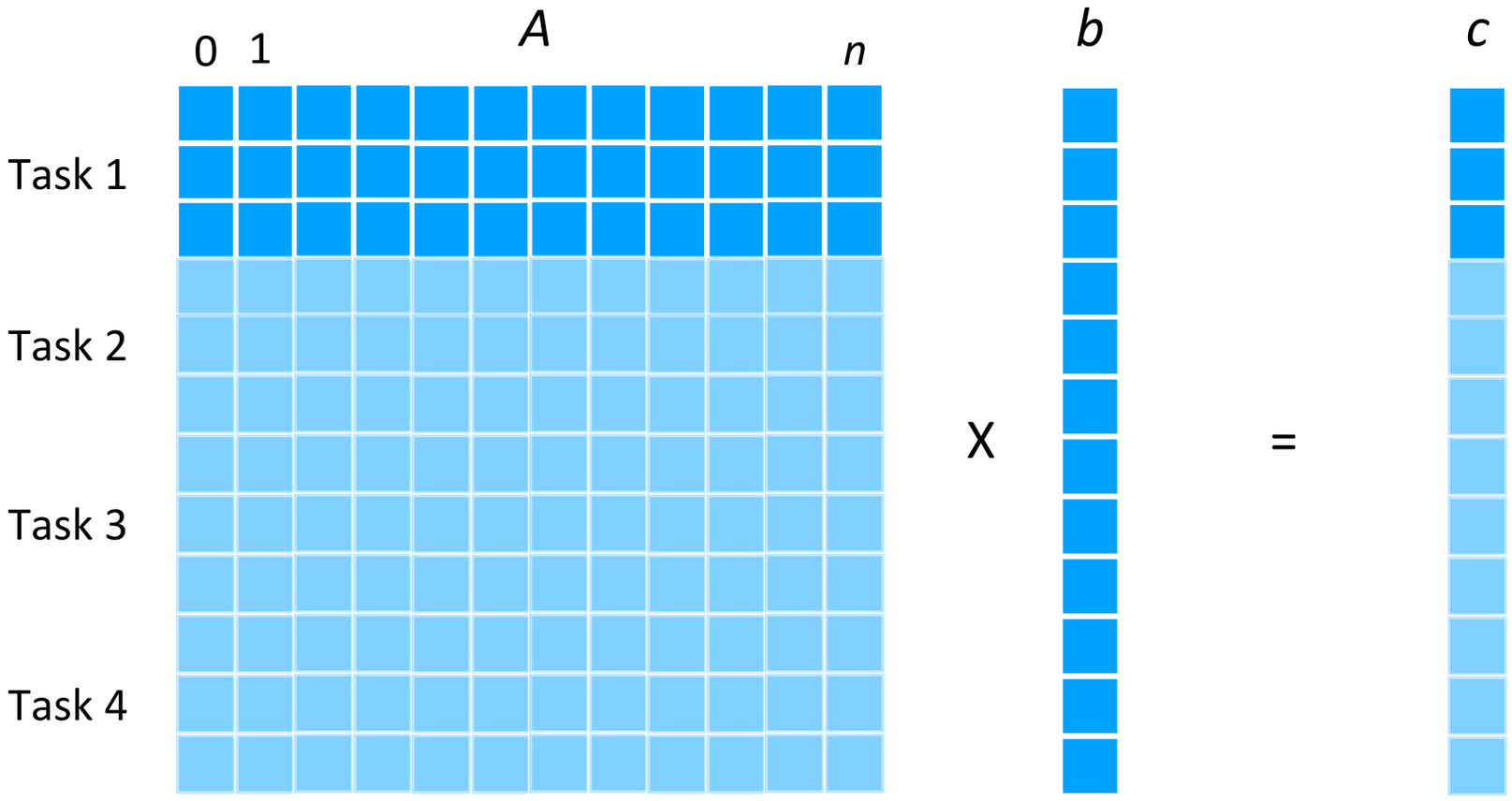


ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

ID#	Color	Year
7623	Green	2001
6734	White	2001
5342	Green	2001

ID#	Model	Year	Color
6734	Civic	2001	White

Task Granularity

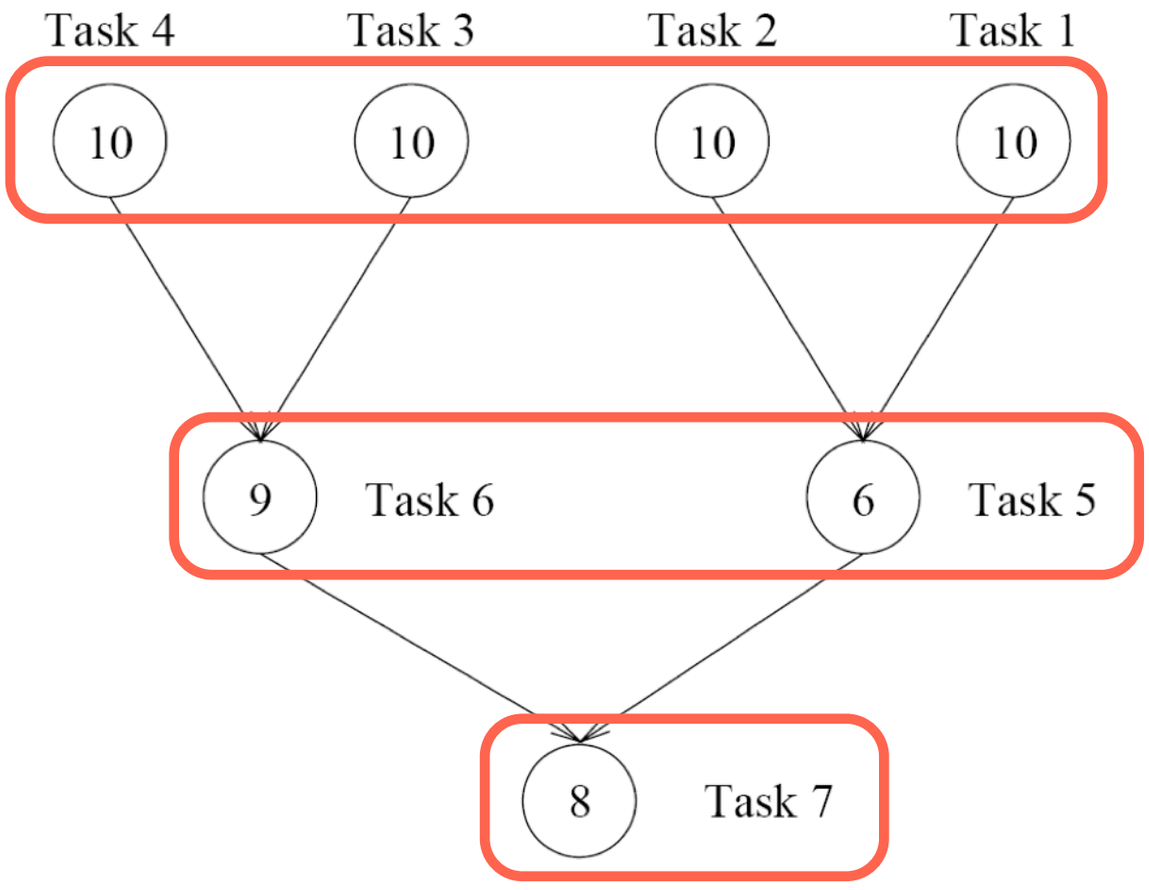


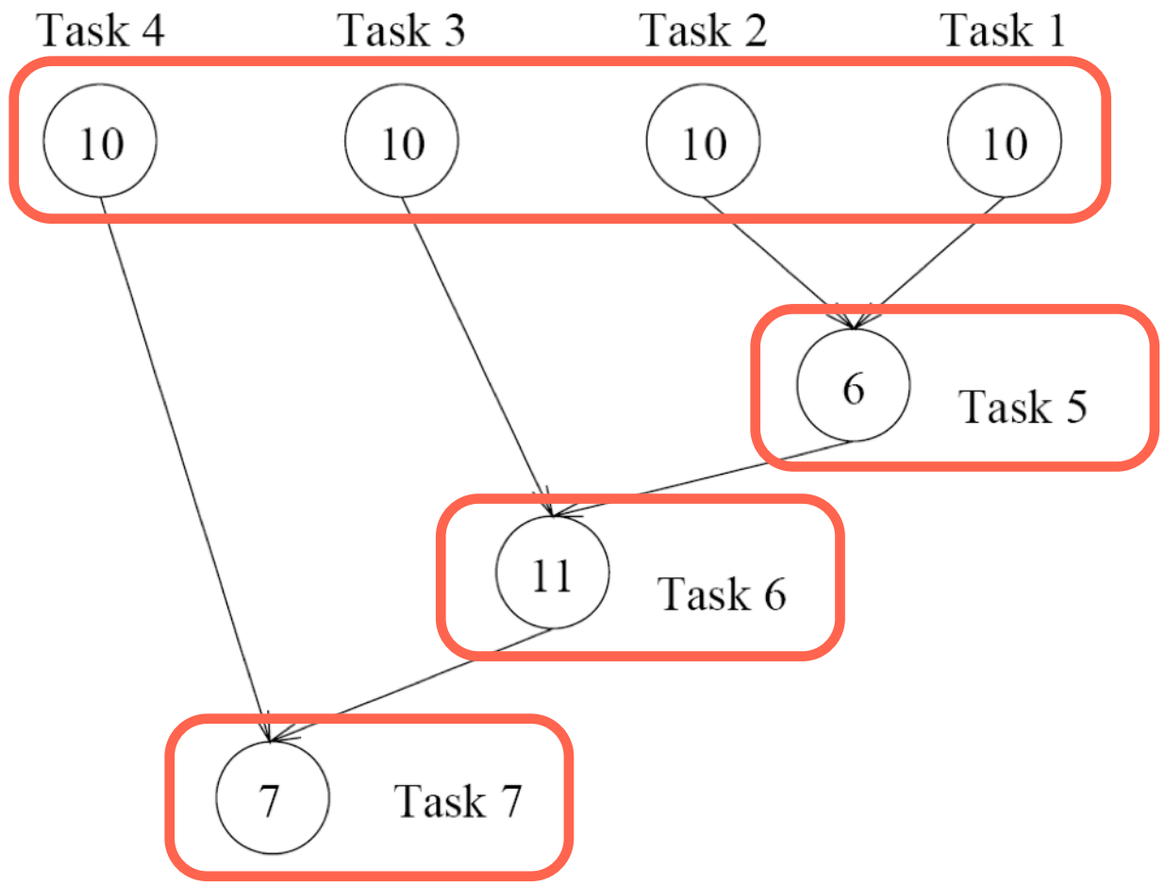
- The number of tasks into which a problem is decomposed determines its granularity
- Decomposition into a large number of tasks results in fine-grained decomposition
- Decomposition into a small number of tasks results in a coarse grained decomposition

Degree of concurrency

- The number of tasks that can be executed in parallel is the degree of concurrency of a decomposition
- Since the number of tasks that can be executed in parallel may change over program execution, the maximum degree of concurrency is the maximum number of such tasks at any time during execution
  - What is the maximum degree of concurrency of the database query examples?
- The average degree of concurrency is the average number of tasks that can be processed in parallel over the execution of the program.
  - Assuming that each task in the database example takes identical processing time, and we have enough processors to execute independent tasks in parallel, what is the average degree of concurrency in each decomposition?
- If the average degree of concurrency is similar to the maximum, the parallel system is used very efficiently
- The degree of concurrency increases as the decomposition becomes finer in granularity and vice versa

Degree of concurrency

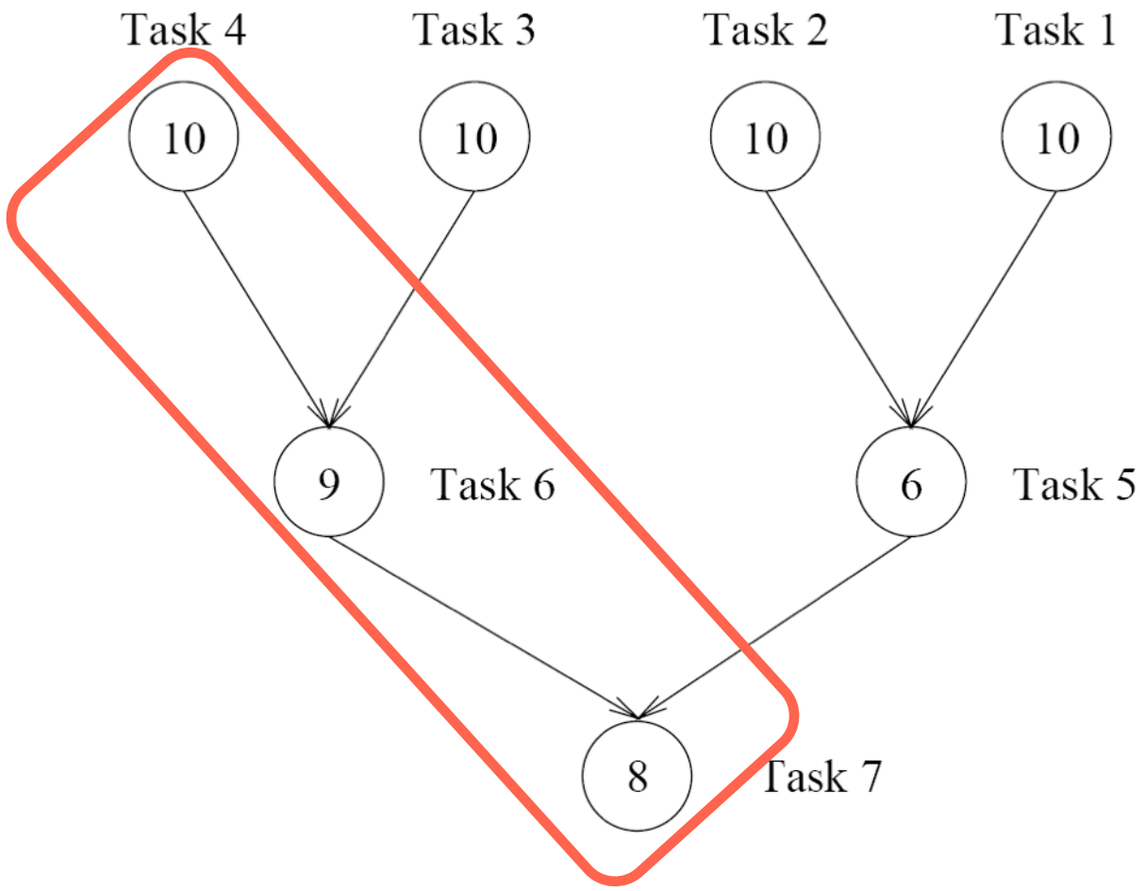


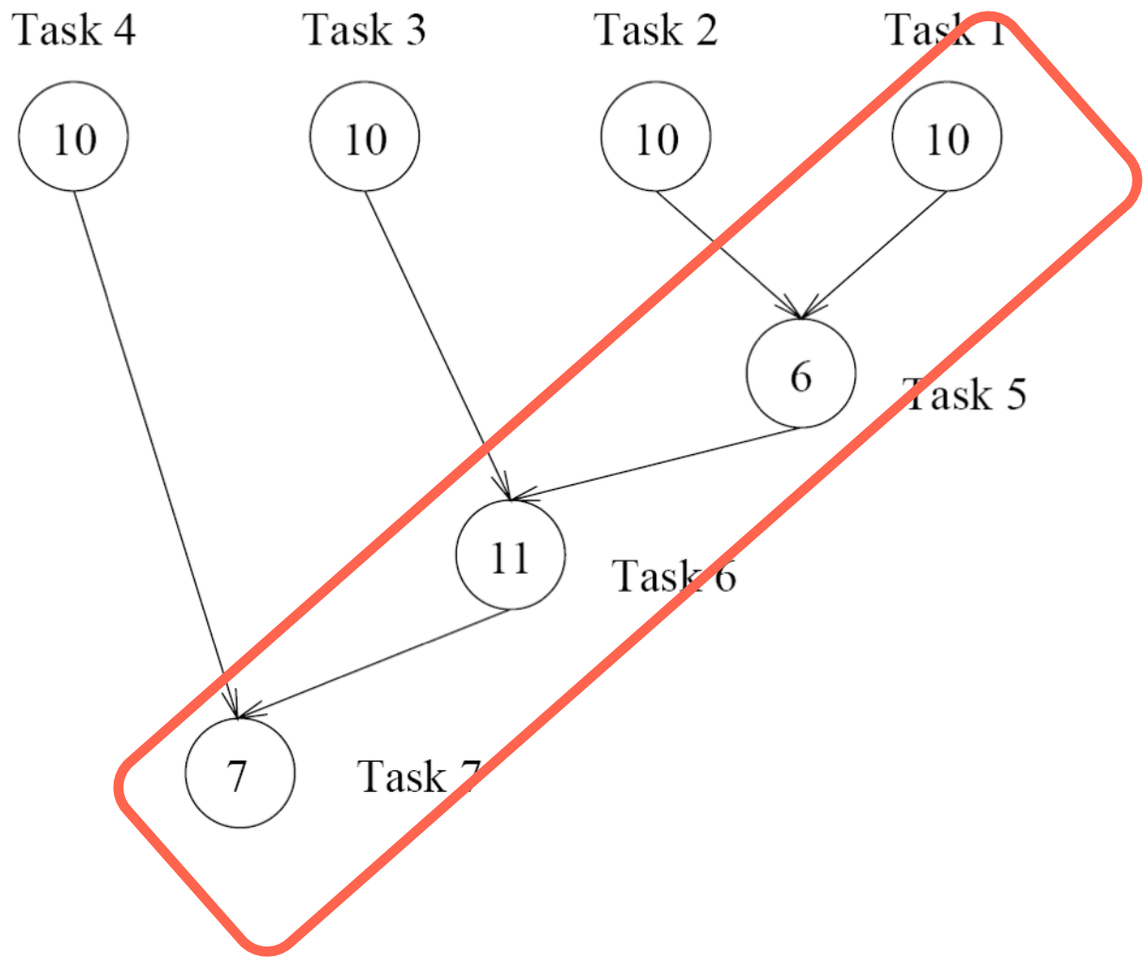


Critical Path

- A directed path in the task dependency graph represents a sequence of tasks that must be processed one after the other
- The longest such path determines the shortest time in which the program can be executed in parallel
  - measured in terms of number of tasks, or sum of the weights of the tasks involved
- The length of the longest path in a task dependency graph is called the critical path length
  - It corresponds to the minimum execution time of a parallel program

Critical Path Length





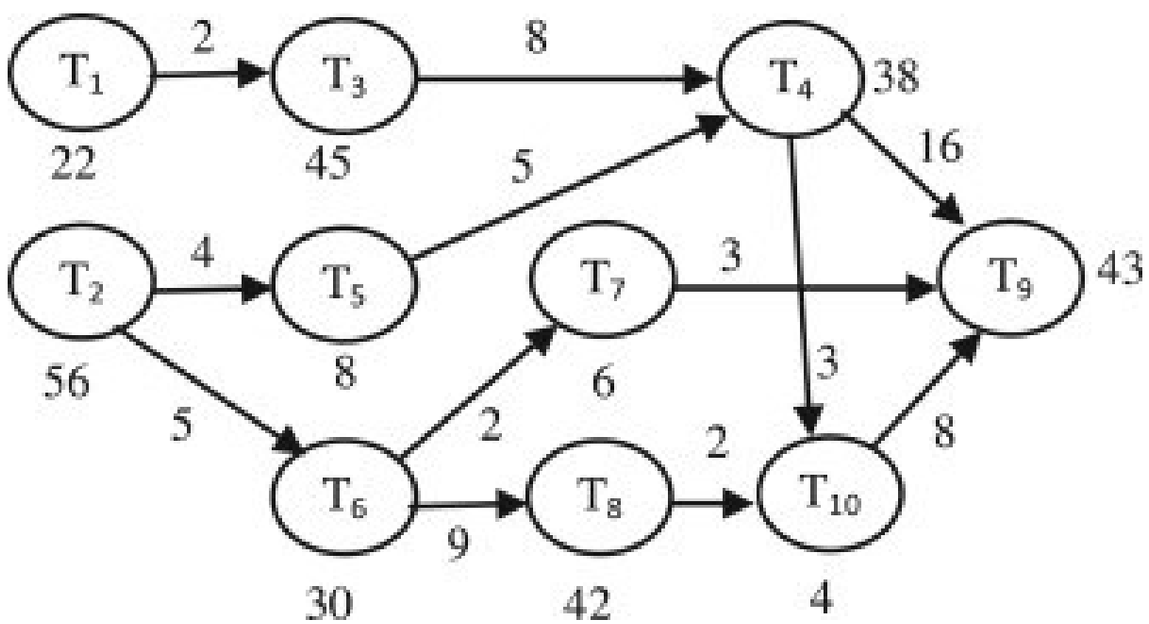
- Task dependency graphs of the two database query
- If each task takes 10 time units, what is the shortest parallel execution time for each decomposition?
- How many processors are needed in each case to achieve this minimum parallel execution time?
- What is the maximum degree of concurrency?

Parallel Performance Limitations

- It would appear that the parallel time can be made arbitrarily small by making the decomposition finer in granularity
- There is an inherent bound on how fine the granularity of a computation can be
  - For example, in the case of multiplying a dense matrix with a vector, there can be no more than (n^2) concurrent tasks
    - That is, all the multiplications performed in parallel
- Concurrent tasks may also have to exchange data with other tasks. This results in communication overhead, which increases along with the parallelism increase (finer granularity)
- There is a tradeoff between the granularity of a decomposition and associated overheads

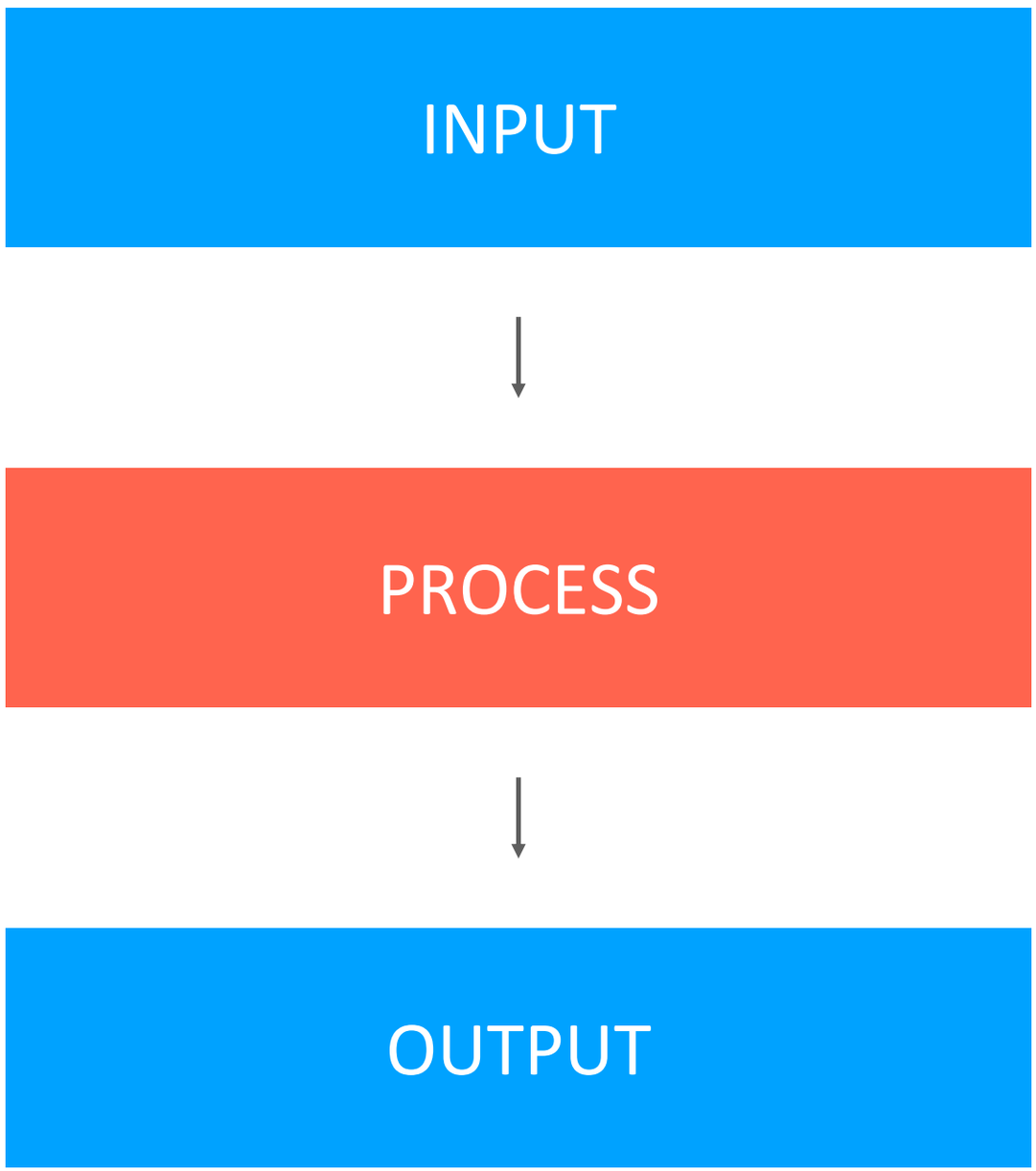
Tasks Interaction Graph

- Subtasks generally exchange data with others in a decomposition
- The graph of tasks (nodes) and their interactions/data exchange (edges) is referred to as a task interaction graph (TIG)
  - Nodes = tasks
  - Edges = interaction or data exchange
  - Node labels = computation weight of tasks
  - Edge labels = amount of data exchanged

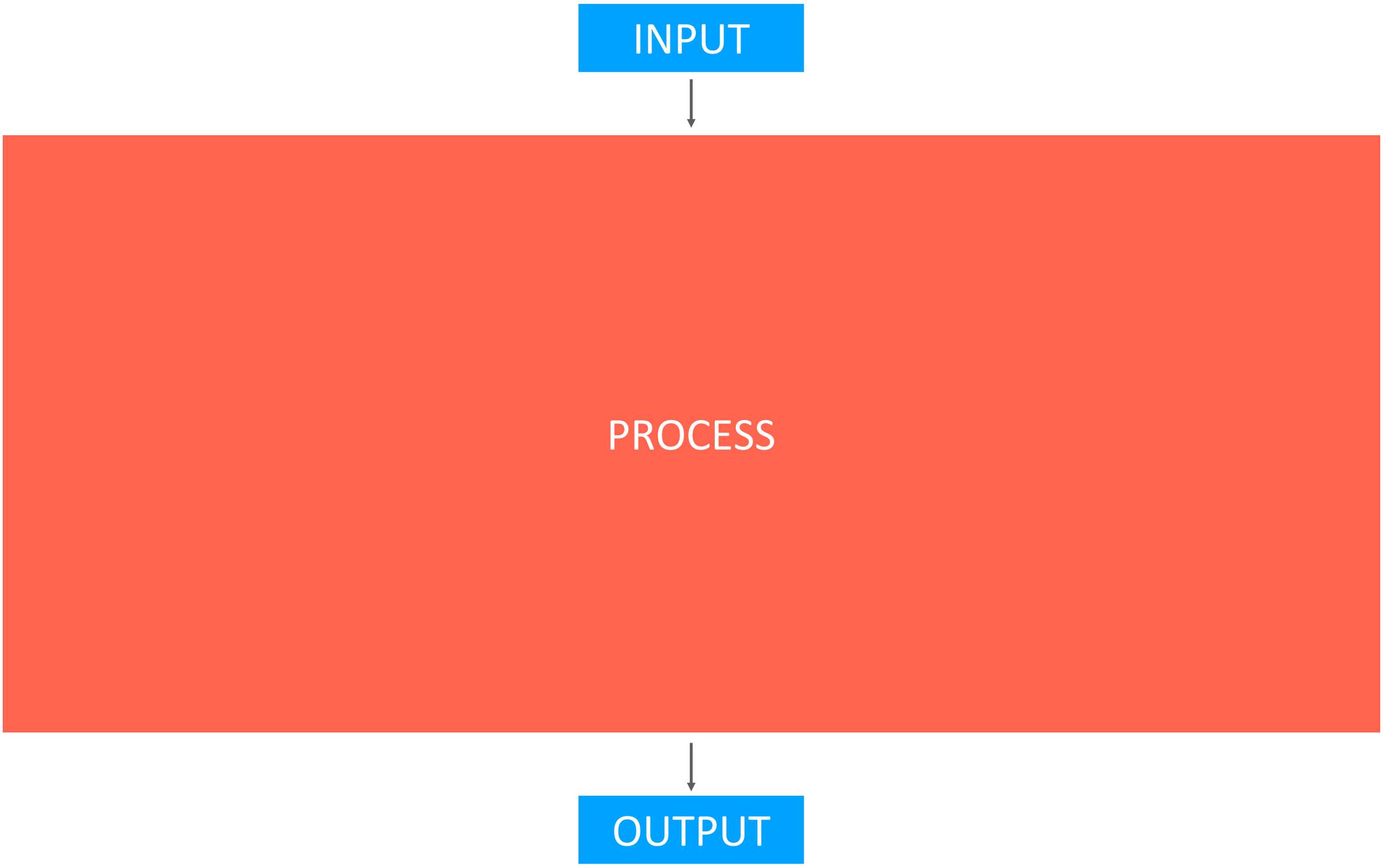


MapReduce

Typical Application

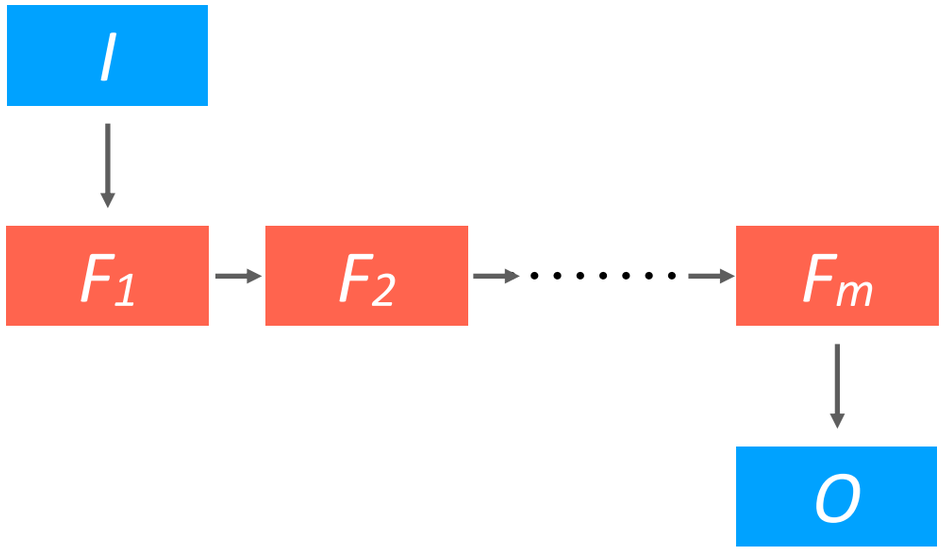


Last time we saw … what if?

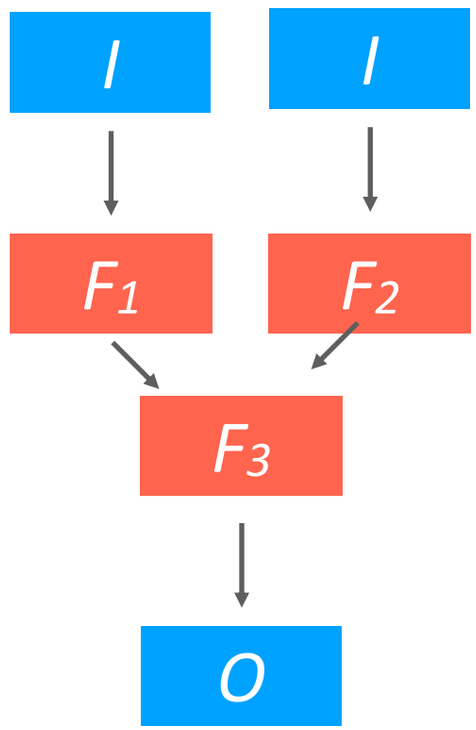


Revising Task parallelism

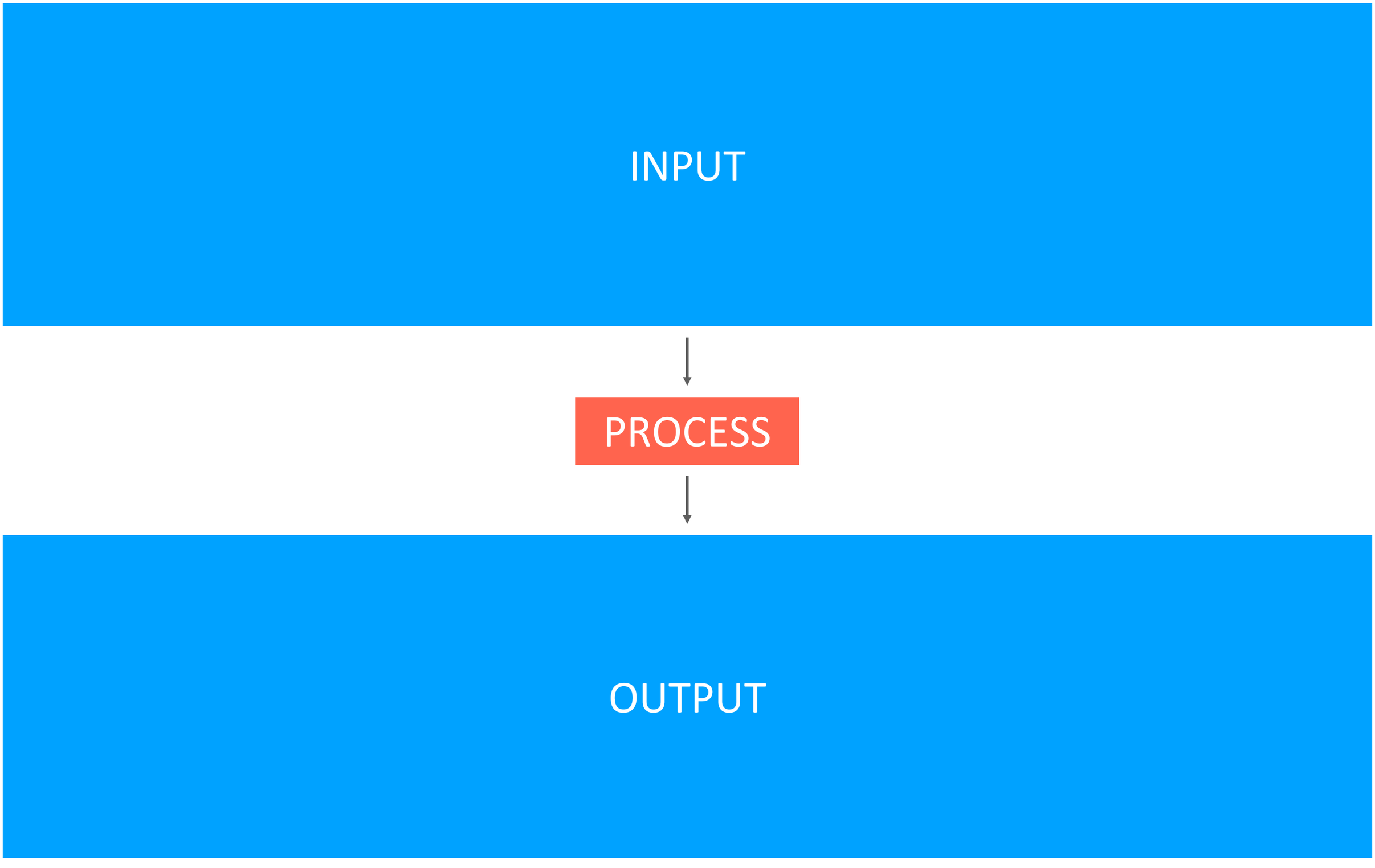
- Sequential algorithm
  - A function *F* is applied on a data structure *I* to yield *O*
  - *O = F(I)*



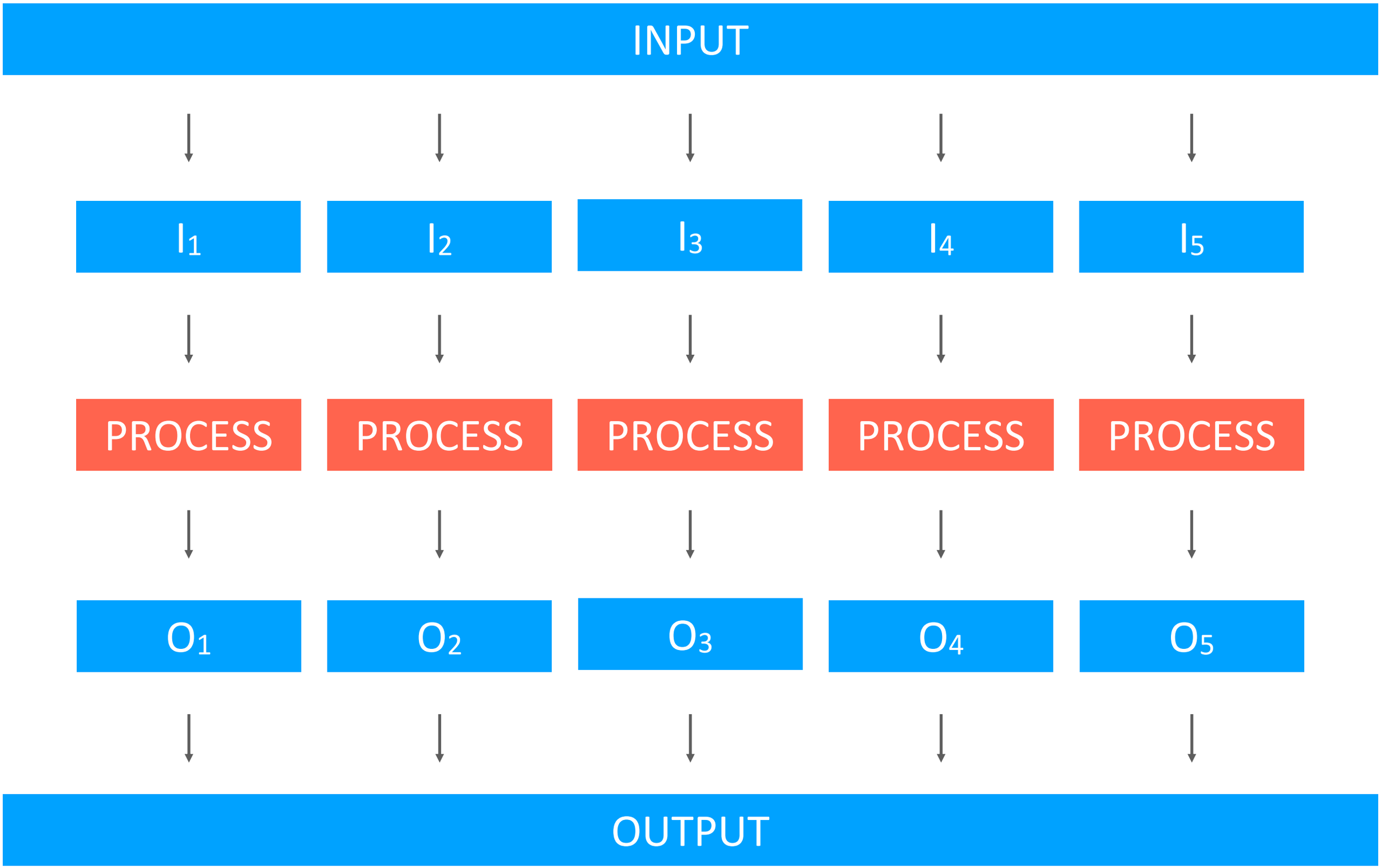
- Task parallelism
  - Partition on control
  - Pipeline
    - $O = F(I) = F_m(F_{m-1}(\ldots F_1(I)))$
  - Task graph
    - $O = F(I) = F_3(F_2(I), F_1(I))$



Today we see … What if?

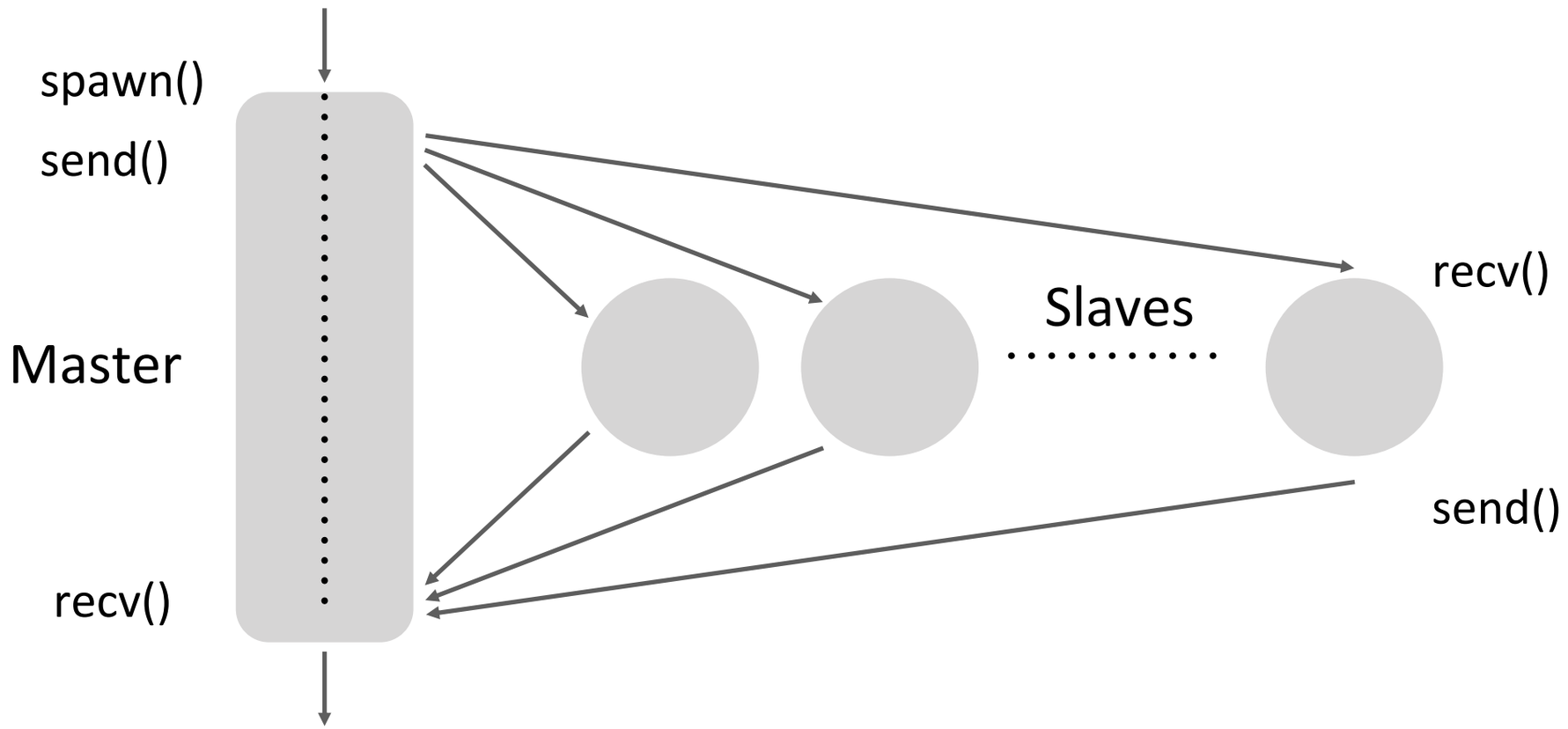


Divide and Conquer



Master-Slave Implementation

- Static partitioning of the input data
- *N* slaves are created dynamically
- Each slave is assigned a partition of the input



- This pattern does not perform well if the assigned tasks are unbalanced

Typical Big Data Application

1. Iterate over a large number of records
2. Extract something of interest from each
3. Shuffle and sort intermediate results
4. Aggregate intermediate results
5. Generate final output

Programmer problems: splitting data

- How to split the data?
- How to distribute the data?
- How to collect the data?
- How to merge the data?
- How to coordinate the access to the data?
- What if more data splits than tasks?
- What if tasks need to share data splits?
- What if a data split becomes unavailable?

Design Ideas

Tackling large-data problems requires a distinct approach:

- Right level of abstraction
  - Hide implementation details from applications development
  - Write very few lines of code
  - Drawback: everything needs to fit into the abstraction
- Scale “out”, not “up”
  - Avoid supercomputer, too costly
  - Use commodity machines, low costs
  - Drawback: many failures
- Move processing to the data
  - Same code, runs everywhere
  - Reduce data over the network
  - Drawback: code must be portable

MapReduce is built on well-known principles from Functional Programming

1. Purity
2. Immutability
3. Higher-order functions

Purity

Will always produce the same output, given the same input parameters (*idempotency*)

Pure Function

```
int pure(int a, int b)
{
    return a + b;
}
```

Not Pure Function

```
const notpure(const moonLandingTime)
{
     return Date.now - moonLandingTime;
}
```

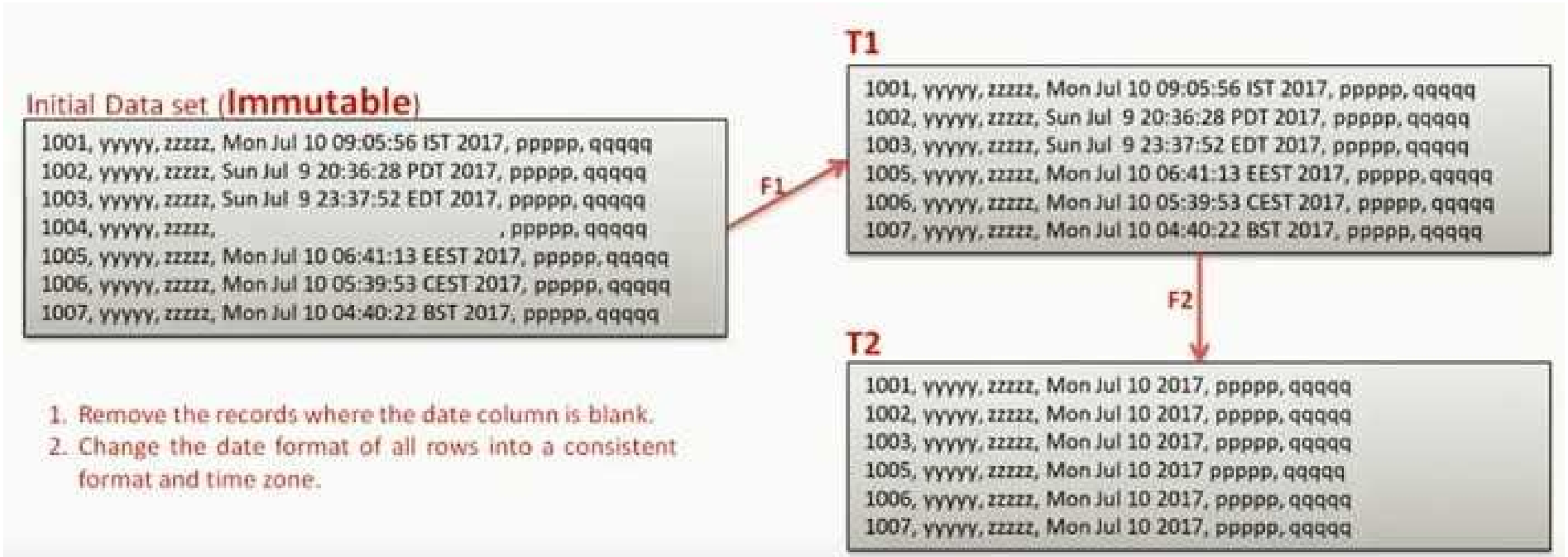
Immutability

1. There are no “variables” in functional programming
2. All “variables” should be considered as constants
3. If needed, create a new data structure with the updated state.
4. Only in few cases, we mutate variables, e.g.,
   - Short living “local” variables
5. Advantages
   - A process does not need to bother about other processes when dealing with a data structure, as nobody can modify it
   - History is not wiped out

Immutability

Example:

1. Remove the records where the date column is blank.
2. Change the date format of all rows into a consistent format and time zone.



What if you delete out your records and days after you find out a bug in your code???

Higher-order functions

1. Higher-order functions are functions that operate on other functions either by:
   - Taking them as arguments (e.g., callback function)
   - Returing them as output
2. Higher-order functions:
   - Make the code easier to understand and debug
   - Allow to perform operations on functions as you would do on other elements