# Università degli Studi dell'Aquila
## Academic Year 2020/2021

*Course*: Distributed Systems (6 CFU, integrated within the NEDAS curriculum with ''Web Algorithms'' (6 CFU), by Dr. Fabio Persia)

*Instructor*: Prof. Guido Proietti

*Schedule*: Tuesday: 11.50 – 13.20 – Room A1.1 (plus Teams)
Thursday: 14.20 – 15.50 – Room A1.1 (plus Teams)

*Questions?*: Wednesday 16.00 - 18.00 (send an email to guido.proietti@univaq.it)

*Slides plus other infos:*
http://people.disim.univaq.it/guido.proietti/2020.html

# (Computational) Distributed Systems

*In the old days:* a number of workstations over a LAN

## Today

Collaborative Computing Systems
- Military command and control
- Massive computation (e.g., mining a block of the Blockchain)

Distributed Real-time Systems
- Navigation systems
- (Airline) Traffic Monitoring

Mobile Ad hoc Networks
- Rescue Operations
- Robotics

(Wireless) Sensor Networks
- Habitat monitoring
- Intelligent farming

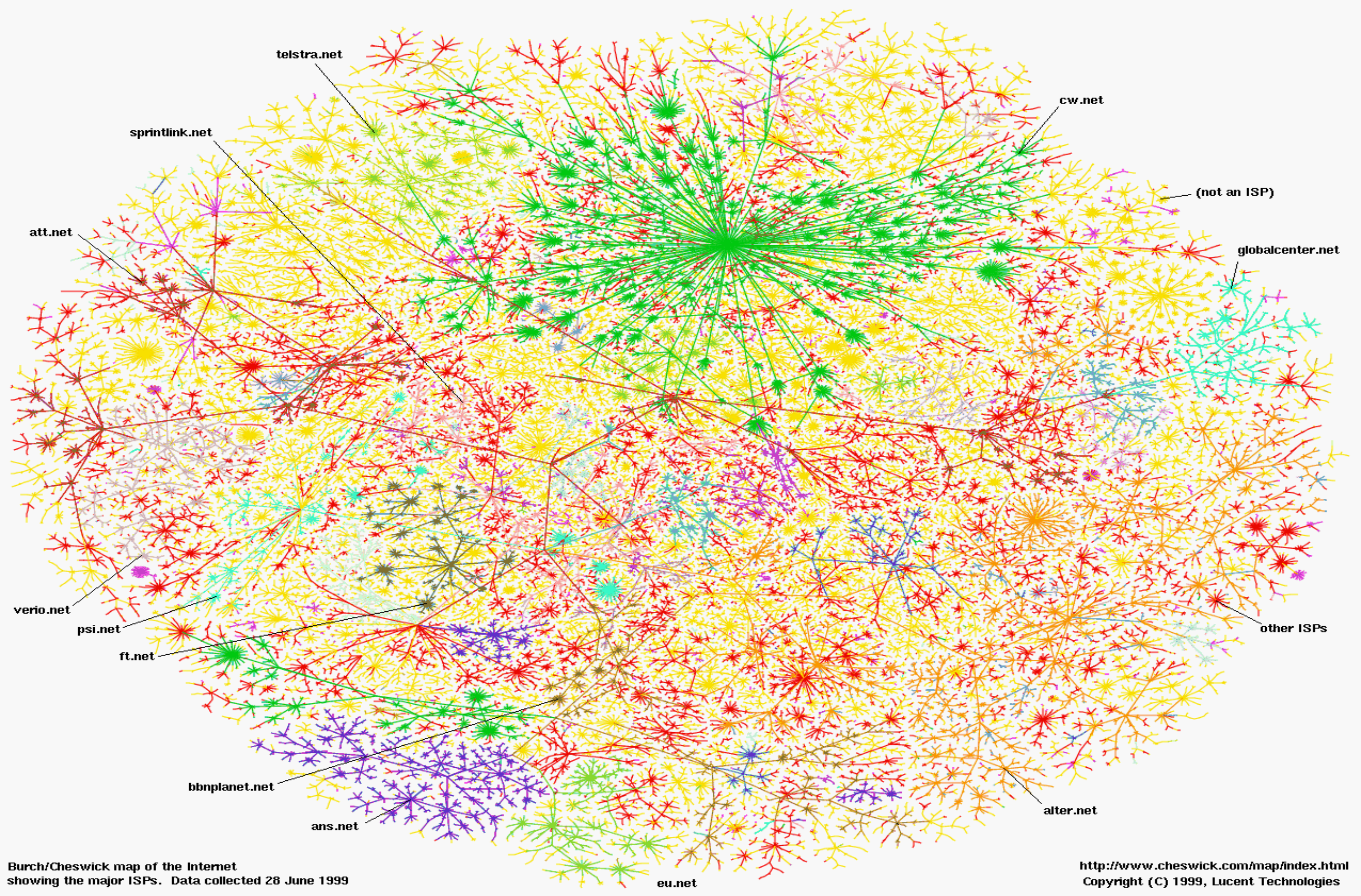
Social Networks

Digital Payment Systems

Grid and Cloud computing

…

# And then, the mother of all DS: the Internet



# Two main ingredients in the course: Distributed Systems + Algorithms

- ***Distributed system (DS)***: *Broadly speaking, this is a set of* ***autonomous computational devices*** *(say,* ***processors****) performing multiple operations/tasks simultaneously, and which influence reciprocally either by* ***taking actions*** *or by* ***exchanging messages*** *(using an underlying wired/wireless* ***communication network****), in order to bring the DS to a* ***final outcome***
- *We will be concerned with the* ***computational aspects*** *of a DS, namely the amount of computational resources needed/spent by its processors in order to compute/reach a certain outcome. As we will see, this will depend on the* ***behaviour*** *of its processors:*
  - *Cooperative: processors are fault-free and they cooperate among them ⇒ Classic field of* ***distributed computing***
  - *Concurrent: processors compete among them for a resource ⇒ Classic field of* ***synchronization/concurrent computing***
  - *Unreliable: processors may fail and operate* ***against*** *the system ⇒ Classic field of* ***fault-tolerance*** *in DS*
  - *The emerging field of* ***game-theoretic aspects of DS****, where processors behave strategically in order to maximize her personal benefit will be studied next year in the class of* ***Autonomous Networks***

# Two main ingredients in the course: Distributed Systems + Algorithms (2)

- ***Algorithm (informal definition)***: *effective method, expressed as a finite list of well-defined instructions, for solving a given problem (e.g., calculating a function, implementing a goal, reaching a benefit, etc.)*
- *The actions performed by each processor in a DS are dictated by a* ***local algorithm****, and the global behavior of a DS is given by the ''composition'' (i.e., interaction) of these local algorithms, then named* ***distributed algorithm***

***General assumption:*** *local algorithms are all the same (so-called homogenous setting), otherwise we could force the DS to behave as we want by just mapping different algorithms to different processors, which is unfeasible in reality!*

- *We w* [...] *existence, correctness, finiteness, efficiency (computational complexity), effectiveness, robustness (w.r.t. to a given fault-tolerance concept), etc.*

# Course structure

FIRST PART (12 lectures): Algorithms for COOPERATIVE DS

1. Leader Election
2. Minimum Spanning Tree
3. Maximal Independent Set
4. Minimum Dominating Set

Mid-term **Written** Examination (week 16-20 of November): 10 multiple-choice tests, plus an open-answer question on the above part

SECOND PART (2 lectures): CONCURRENT DS: Mutual exclusion

THIRD PART (8 lectures): Algorithms for UNRELIABLE DS

1. Benign failures: consensus problem
2. Byzantine failures: consensus problem

FOURTH PART (2 lectures): Advanced topics in DS: the Blockchain

Final **Oral** Examination: There will be fixed a total of 6 dates, namely:

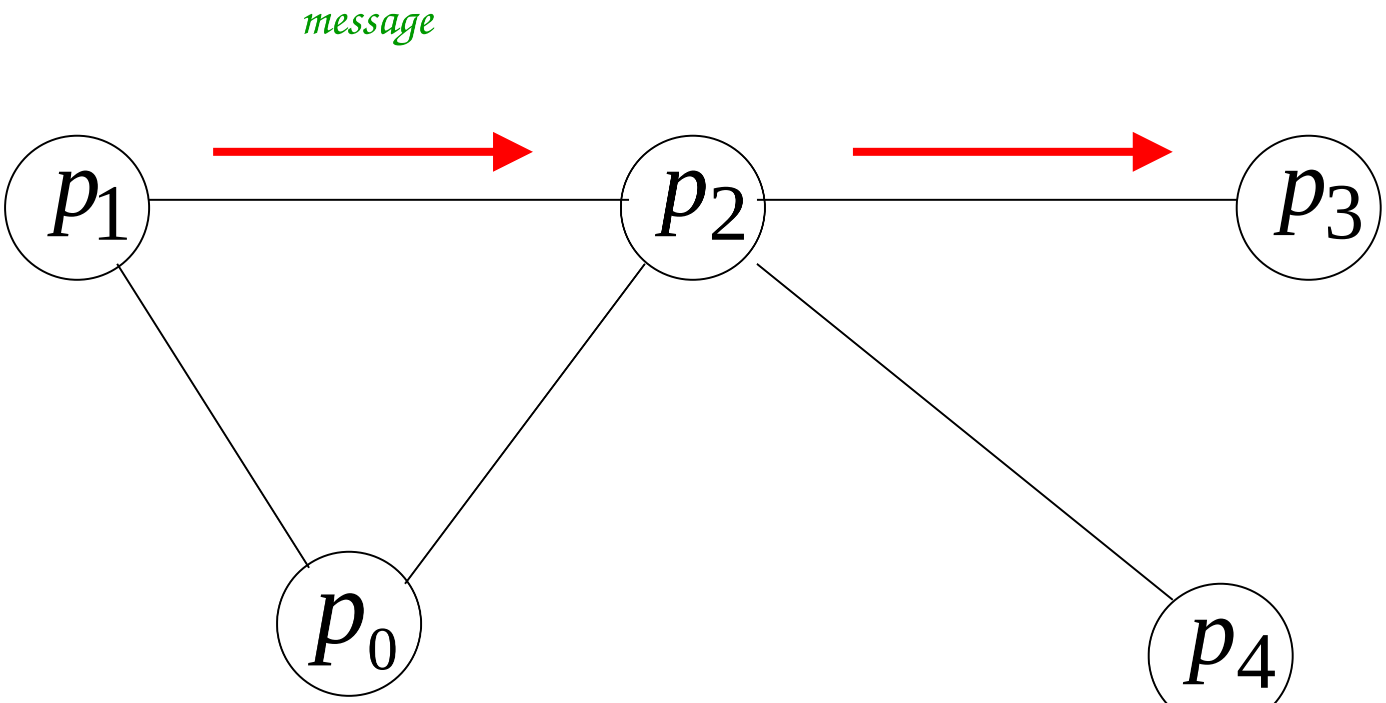
- 3 in January-February
- 2 in June-July
- 1 in September

Students passing the mid-term written examination can take the oral examination **on the second half of the program only** (limited to the January-February session).

# *The 12 CFU course (Distributed Systems & Web Algorithms)*

- *For those enrolled in the NEDAS curriculum, there will be a* ***single*** *final grade as a result of the grades obtained in this course and in the "Web Algorithms" course*
- *The corresponding exams* ***can be done separately****, but they must be sustained within the* ***same calendar year (i.e., 2021)***
- *People must register for the 12 CFU exam, but for the actual dates of the 2 courses they must refer to the calendar for the corresponding 6 CFU exams (this is published on the DISIM site, or it can be asked to me and to Dr. Persia)*

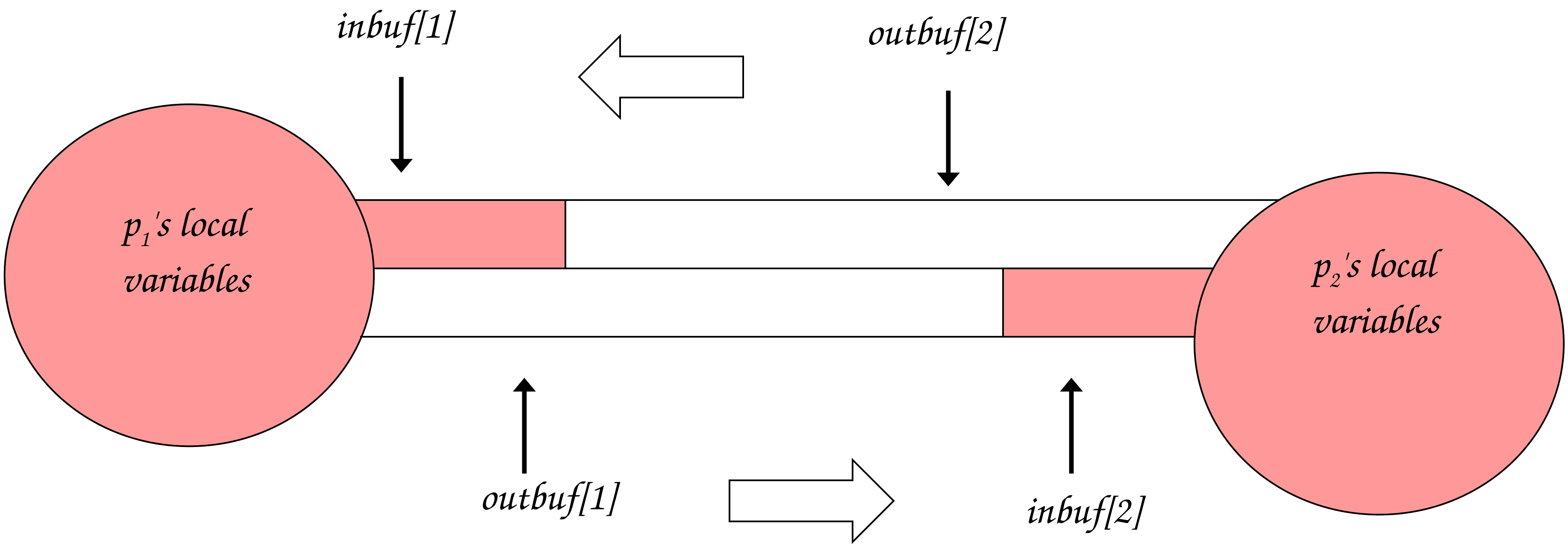
# Cooperative DS:
# Message Passing Model/System



# The model (MPS)

- *n processors $p_0, p_1, \ldots, p_{n-1}$ which communicate by exchanging messages*
- *It can be modelled by a graph $G=(V,E)$: nodes $V$ are the processors, while edges $E$ are the (bidirectional) point-to-point communication channels*
- *Each processor has a consistent knowledge of its neighbors, numbered from 1 to $r$*
- *Depending on the context, a processor (or more precisely, its algorithm) may make use of global information about the network, e.g., the size, the topology, etc.*
- ***Communication** of each processor takes place only through **message exchanges**, using buffers associated with each neighbor, say **outbuf** and **inbuf***
- *$Q_i$: the **state set** for $p_i$, containing a distinguished initial state; each state describes the current internal configuration of the processor and the content of the incoming buffers*

# Modeling Processors and Channels




Pink area (local vars + inbuf) is the current state of a processor

# Configuration and events

- ***System configuration C***: A vector $[q_0, q_1, \ldots, q_{n-1}]$ where $q_i \in Q_i$ is the state of $p_i$, plus the status of all outbuffers

- ***Events***: Computation events (internal computations plus sending of messages), and message delivering (receipt of messages) events

# *Execution*

$C_0\,\phi_1\,C_1\,\phi_2\,C_2\,\phi_3\,\ldots$ *where*

- ✓ $C_0$ : *The initial configuration (all processors are in their initial state and all the buffers are empty)*
- ✓ $\phi_i$ : *An event*
- ✓ $C_i$ : *The configuration generated by* $\phi_i$ *once applied to* $C_{i-1}$

# *Synchronous MPS*

- ✓ *Each processor has a (universal) clock, and computation takes place in **rounds** (ticks of the clock)*
- ✓ *At each round each processor:*
  1. *Reads the incoming messages buffer*
  2. *Makes some internal computations*
  3. *Sends messages which will be read in the next round.*

# *Asynchronous MPS*

- *No any universal clock: events happen at arbitrary time*
- *No **upper bound** on internal computations and delivering times of messages*
- ***Admissible** asynchronous execution: each message sent is eventually delivered*

# Time Complexity in synch vs asynch

- We will assume that each processor in both models has **unlimited** computational power (quite strange, isn't it?)
- According to this assumption, to establish the time complexity of a synchronous algorithm, we will simply count the **number of rounds** until termination
- Asynchronous systems: the time complexity is not really meaningful, since processors do not have a consistent notion of time

# Message Complexity

- We will assume that each message can be **arbitrarily** long
- According to this assumption, to establish the efficiency of an algorithm, both in the synch and in the asynch case, we will only count the **total number** of messages sent during any admissible execution of the algorithm (in other words, the number of message delivery events in the worst case), regardless of their size

# Different MPS

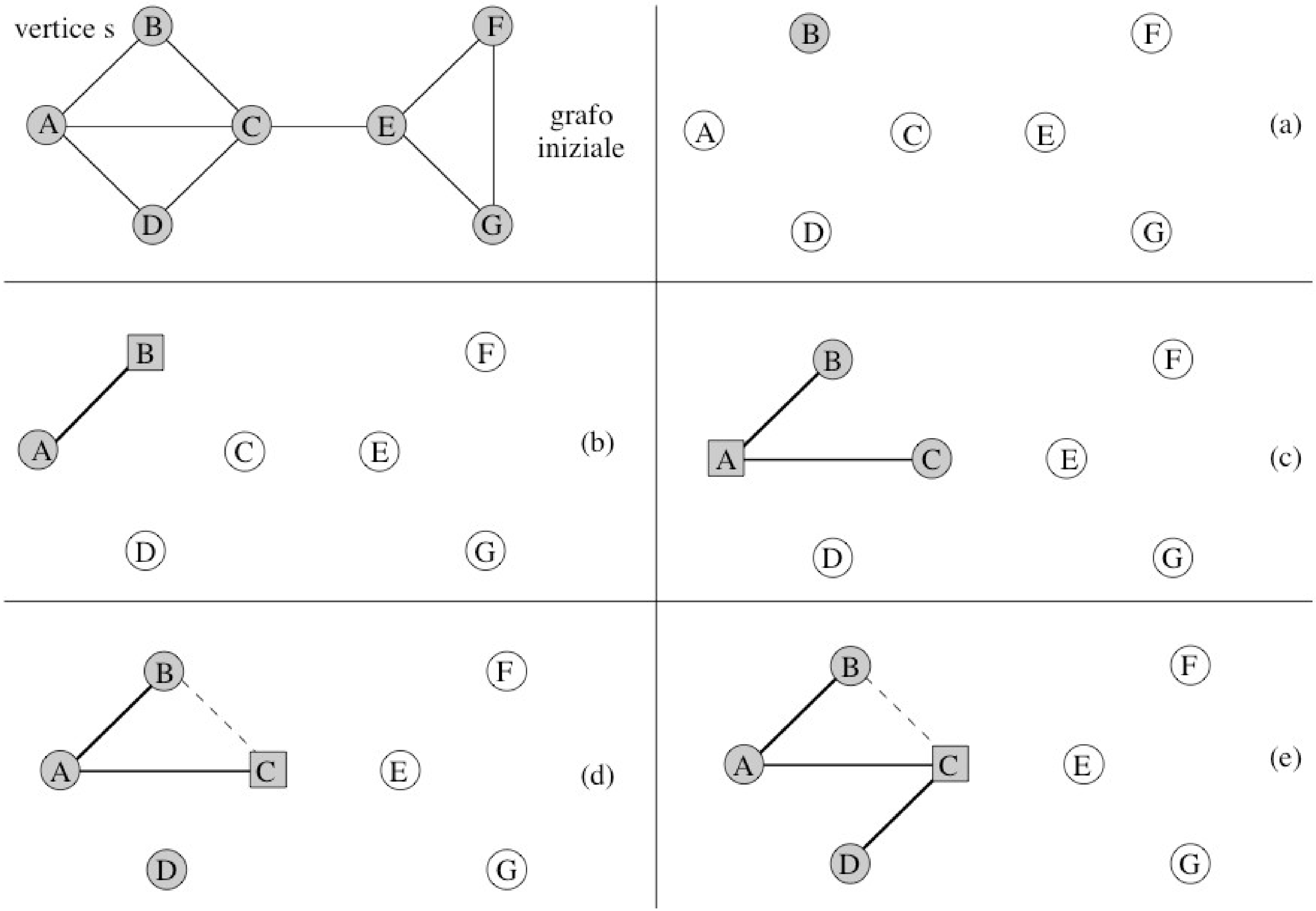
- **Topology** of the network (connected undirected graph representing the MPS): clique, ring, star, etc. A regular topology can sometimes be of help, while some other times it will be a drawback!
- **Synchronicity**: asynchronous versus synchronous (universal clock): synchronous systems are much more powerful than asynchronous ones!
- **Symmetry**: anonymous (processors are indistinguishable) versus non-anonymous: this is a very tricky point, which refers to whether a processor has a distinct ID which can be used during a computation; as we will see, there is a drastic difference in the computational power of a DS, depending on this assumption
- **Uniformity**: uniform (number of processors is unknown) versus non-uniform: knowing the number of processors in the systems can greatly help to solve a problem

- *Example: Distributed Depth-First Search visit of a graph*
  - Visiting a (connected) graph $G=(V,E)$ means to explore **all** the nodes and edges of the graph
  - General overview of a *sequential* algorithm:
    1. Begin at some source vertex, $r_0$
    2. when a vertex $v$ is visited for the first time

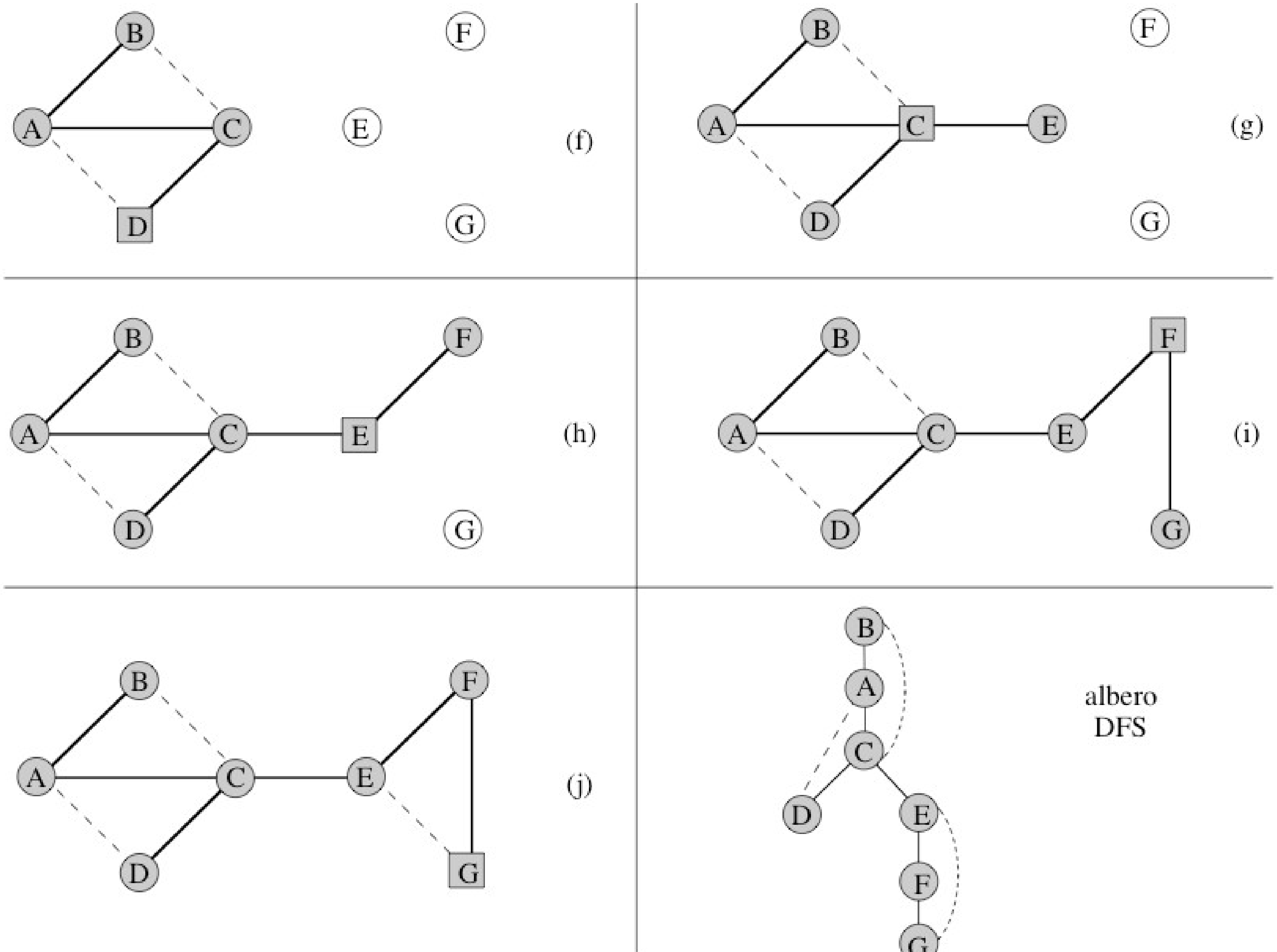
       2.1 if $v$ has an unvisited neighbor, then visit it and proceed further from it

       2.2 otherwise, return to parent($v$), and if parent($v$) $\neq$ NULL proceed the visit further from it, otherwise terminate since $v = r_0$
  - DFS defines a tree, with $r_0$ as the root, which spans all vertices in the graph
    - sequential time complexity = $\Theta(|E|+|V|)$ (we use **Θ** notation because *every* execution of the algorithm costs exactly $|E|+|V|$, in an asymptotic sense)

# DFS: an example (1/2)



# DFS: an example (2/2)



# Distributed DFS: an asynchronous algorithm

- **Distributed version** (token-based): the (virtual) token traverses the graph in a depth-first manner using the algorithm described above, but with the following modifications
  1. Start exploration (visit) at a waking-up node (root) $r$ (who wakes-up $r$? Good question, we will see…)
  2. When $v$ is visited for the first time (i.e., it gets the token, namely it receives a message from a neighbor parent node):
     - 2.1 Inform (i.e., send a message to) all of its neighbors that it has been visited
     - 2.2 Wait for an acknowledgment (i.e., receive a message) from all neighbors

       (we will see steps 2.1 and 2.2 are useful in the synchronous case)
     - 2.3 Select an unvisited neighbor node and pass the token to it; if no unvisited neighbor node exists, then pass the token back to the parent node (if no parent node exists, stop)
  3. When $v$ gets the token from a child node, repeat step 2.3.
- Message complexity is $\Theta(|E|)$, since it is easy to see that onto each edge a constant nember of messages is passed; this is optimal, because of the trivial lower bound of $\Omega(|E|)$ induced by the fact that every node must know the status of each of its neighbors, and this requires at least one message for each graph edge

# *Time complexity analysis for the synchronous case*

- *The synch version is just as you expect: steps 2.1-2.3 are performed in rounds*
- *Observe that through steps 2.1 and 2.2, we ensure that vertices visited for the first time know which of their neighbors have been visited; this way, each node knows which of its neighbors is still unexplored*
- *Number of rounds for steps 2.1-2.3:*

*Round #1: Node v informs all its O(|V|) neighbors it has been visited;*

*Round #2: Neighbors receive notification and send an Ack message*

*Round #3: v receives all the Ack messages and either pass the token to an unvisited neighbor or back to its parent*

- *Number of rounds for each token-back case (Step 3): 1*

⇒ *3 rounds for each new discovered node, plus 1 round for each token-back case ⇒ time complexity = 3|V|+(|V|-1) rounds= Θ(|V|)*

***Homework**: What does it happen to the algorithm's complexity if we do not inform the neighbors (i.e., we remove 2.1 and 2.2) about having been visited?*