Fault tolerance in dynamic distributed systems

Pierre Sens

Delys Team
LIP6 (Sorbonne Université/CNRS), Inria Paris

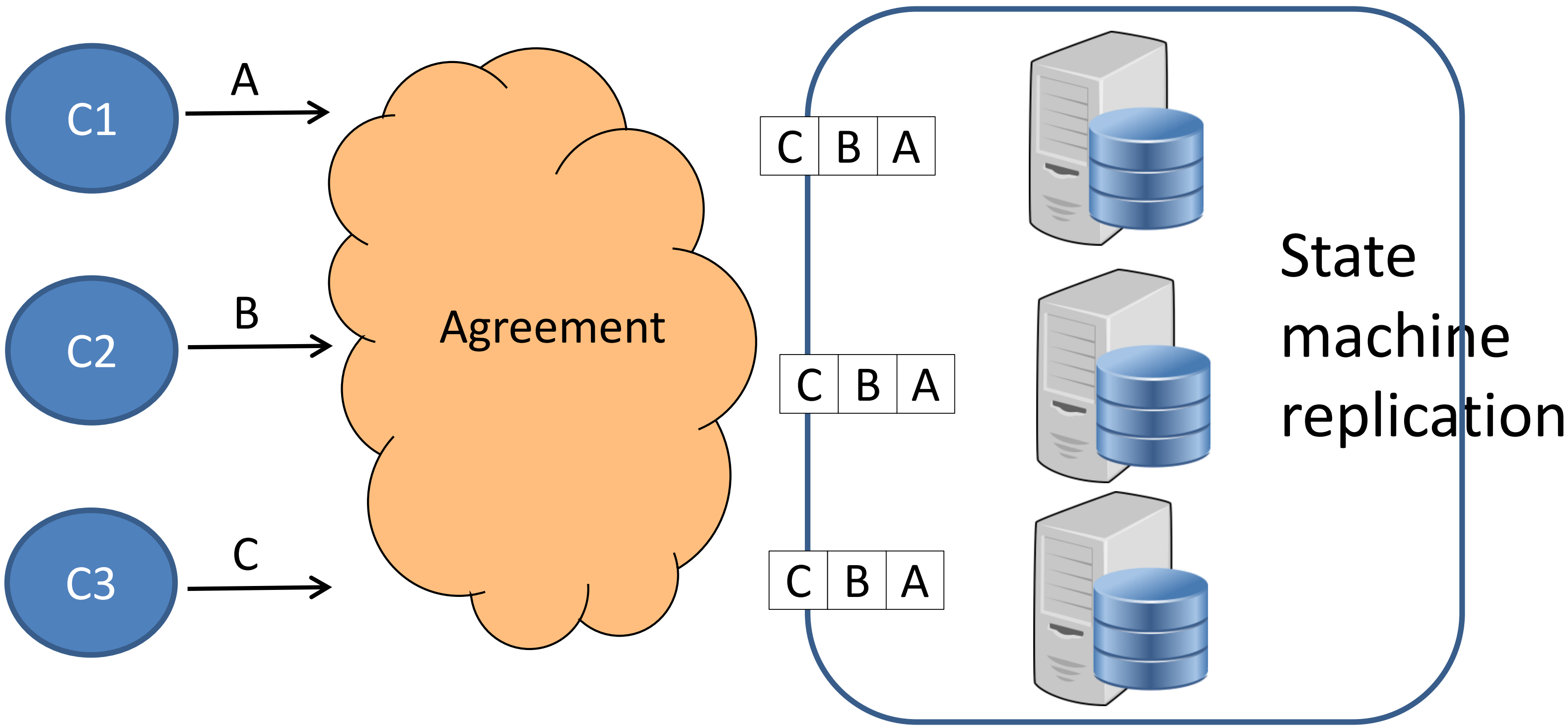
Pierre.Sens@lip6.fr

Outline

- Fundamental abstractions for distributed algorithms
- Modeling dynamic systems
- Fault tolerant algorithms in dynamics systems : some results and open issues

Agreement problems

- Fondamental abstraction to build reliable services



agreement on order of operations

Agreement problems: consensus

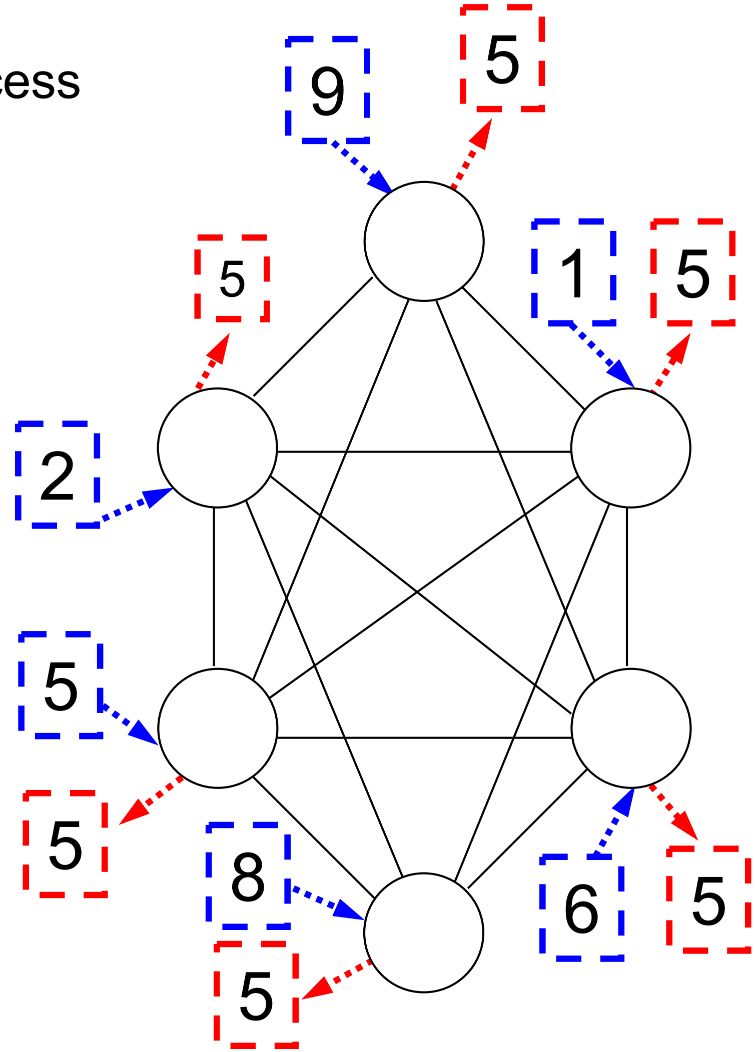
Initially 1 value proposed by each process

Eventually Every correct process decided the same proposed value

Validity: Any value decided is a value proposed

Agreement: No two correct processes decide differently

Termination: Every correct process eventually decides



Other agreement problems

all correct processes try to agree on **some set** of proposed values

- k-set agreement
  - **Agreement**: At most k values are decided.
  - **Validity**: Every value decided must have been proposed.
  - **Termination**: Eventually, every correct process decides.

  Generalization of consensus (k=1)

- set agreement: k=n-1

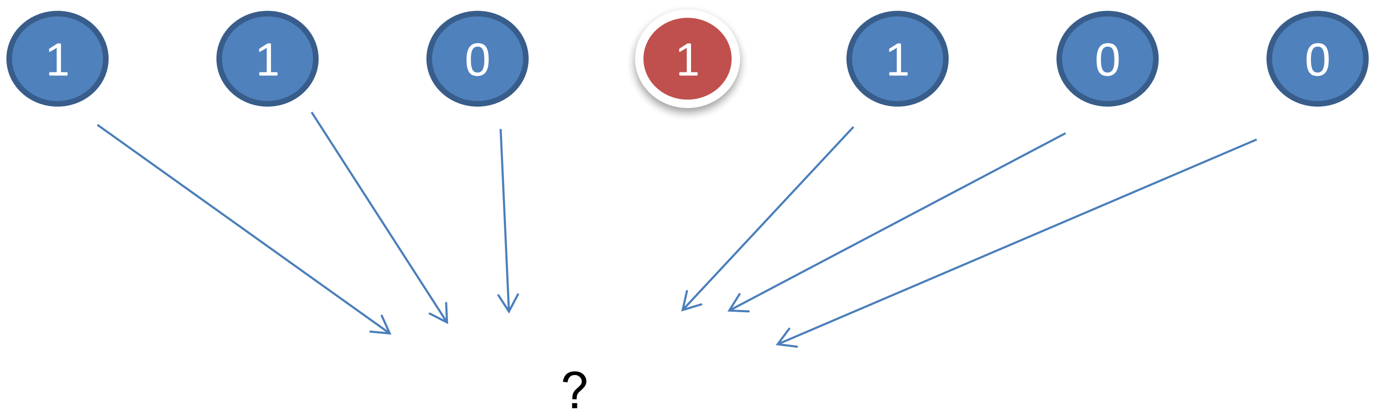
Traditional assumptions

- Connectivity
  - ☼ = {p1,p2, .., pn} **known processes**
  - n processes strongly connected (**no partition**)
- Time
  - Synchronous links (known bound on transmission delays)
  - Asynchronous links (no bound)
- Failures
  - Crash, recovery, Byzantine

A fundamental result

- “Impossibility to solve deterministically the **consensus** in a asynchronous networks with only **1 crash failure**” **[Fischer-Lynch-Paterson 85]**

- *The idea*: impossible to distinguish faulty hosts from slow ones



Circumvent FLP impossibility

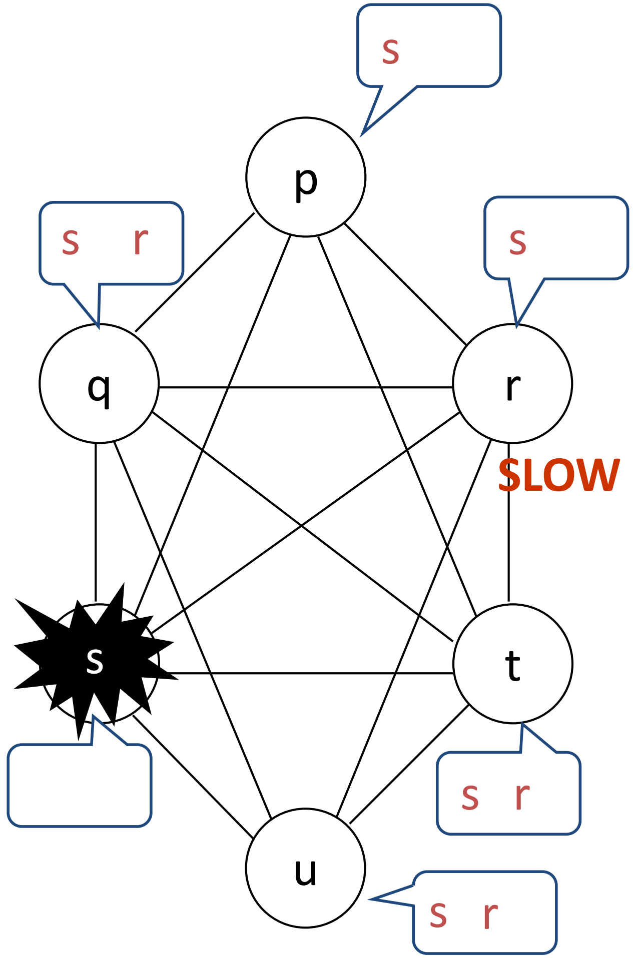
4 approaches:

- Probabilistic (probabilistic consensus, e.g., Ben-Or)
  - Possibly no termination
- k-agreement
  - A relaxed consensus (may output k different values)
- Partial synchrony
  - Add assumptions on the network
  - Eg, There is an unknown bound on the transmission delay
- **Unreliable failure detectors**

Unreliable failure detectors

- Introduced in the beginning of 90's by Chandra and Toueg
- Failure detector = an oracle per node
- Oracles provide lists of hosts *suspected* to have crashed

  => possibly false detections



System model

- n processes $\pi=\{p_1, . . . , p_n\}$
- Processes communicate by **message passing**
- Fully connected **asynchronous network**
- Reliable channels
- Processes may **crash** (processes that do not crash are called correct)
- The system is enhanced with failure detectors

Properties of FD

- **Strong Completeness**:
  - Eventually every process that crashes is permanently suspected by ***every*** correct process
- **Accuracy**:
  - [Eventual] Strong: [There is a time after which] correct processes are not suspected by any correct processes
  - [Eventual] Weak: [There is a time after which] **some** correct processes are not suspected by any correct proc

	Accuracy			
	Strong	Weak	Eventually strong	Eventually Weak
Strong completeness	Perfect P	Strong S	◊ P	◊S

Variantes : Eventual leader

Ω : Output only **one trusted process**, the eventual leader

The leader is eventually the **same correct** process for every correct process

Weakest failure detectors

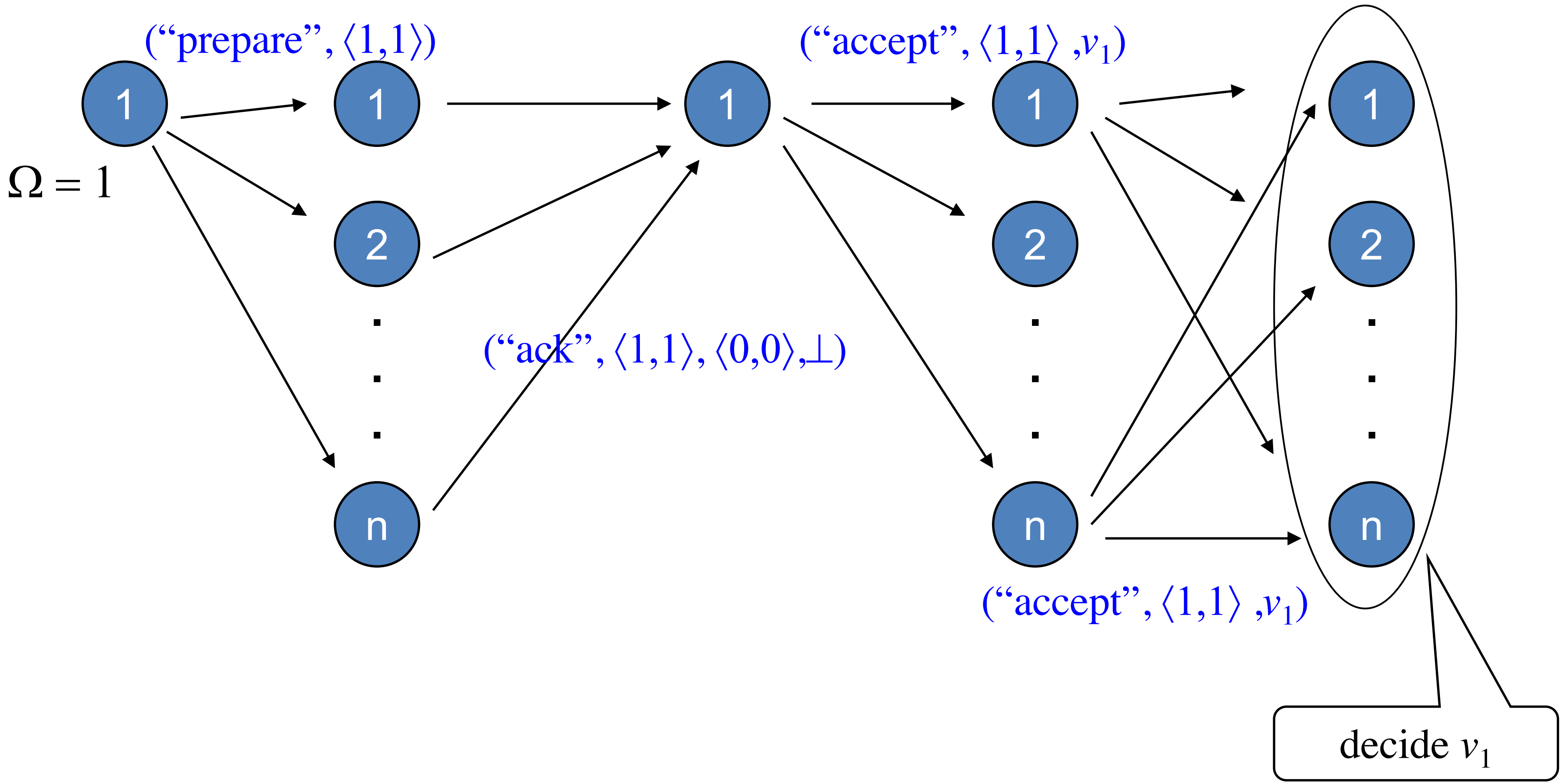
- Introduced by Chandra, Hadzilacos and Toueg
- A weakest failure detector D for a problem P has to be :
  - Sufficient: with D it is possible to solve P
  - Necessary: every other sufficient FD D' is stronger than D (D' can emulate D)

Ω and ◊S are the weakest FD to solve consensus with a majority of correct processes (eg. Paxos)

=> Ω and ◊S are equivalent

Consensus on weakest FD

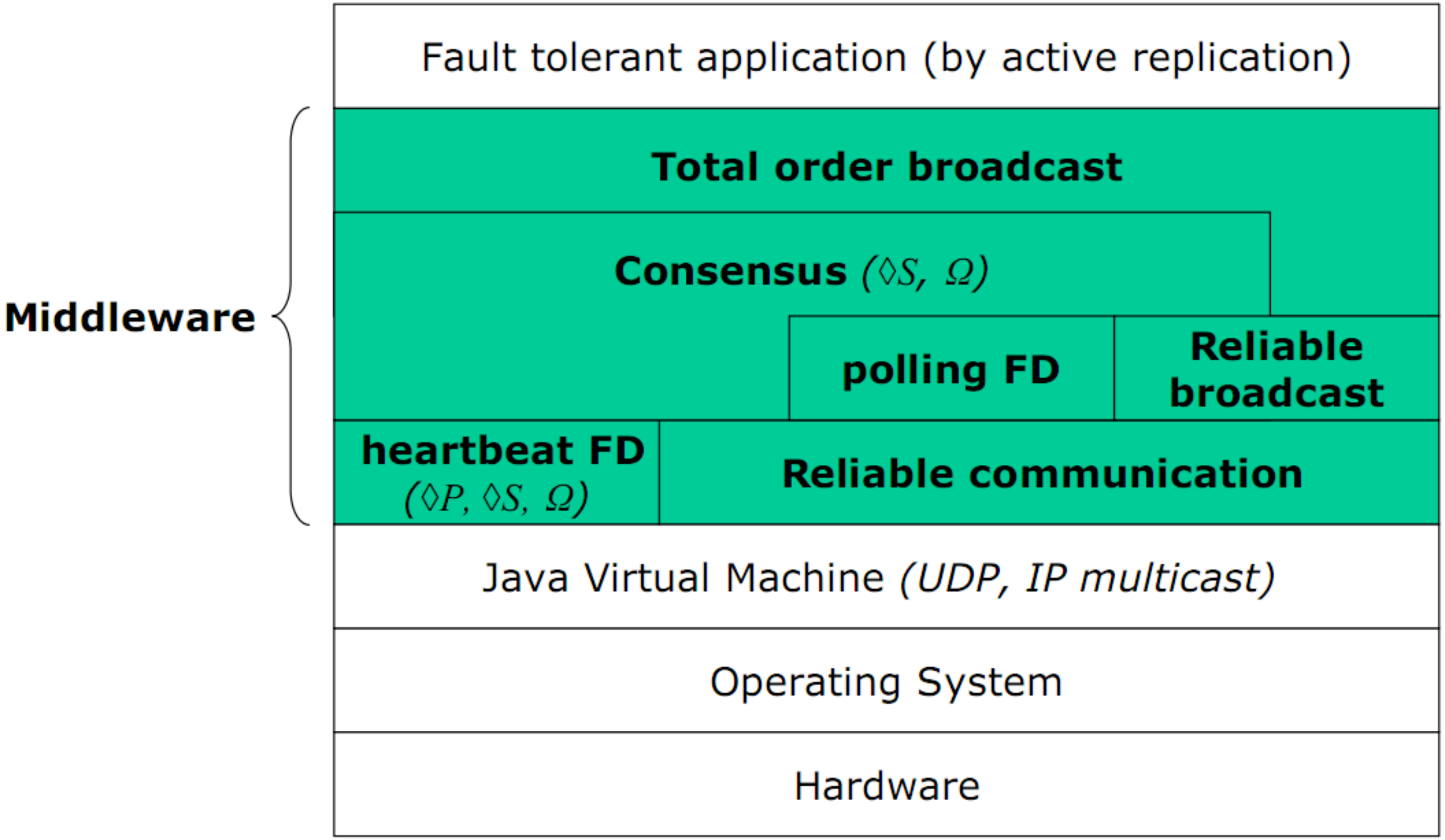
- Paxos



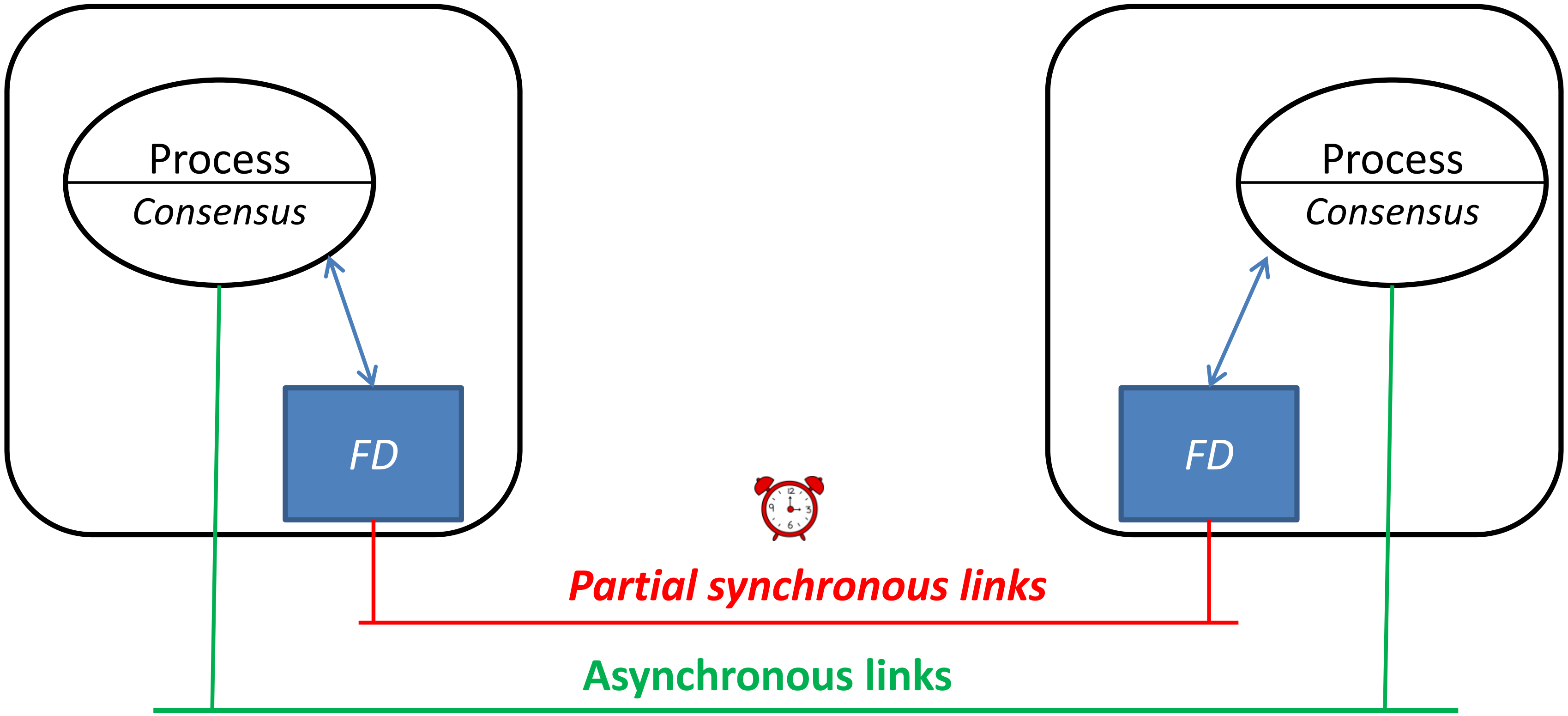
Some weakest FD results

Problems / Models	Consensus	k-set agreement	set agreement	Eventual consistency
Shared memory	Ω [LH94]	k-anti-Ω [GK09]	anti-Ω [Z10]	
Message passing	(Ω,Σ) [DFG10]	?	$\mathcal{L}$ [DFGT08]	Ω [DKGPS15]

Implementation : Fault-tolerant Architecture



Implementation of FDs



Additional assumptions

- Assumptions on transmission delay Δ and relative process speed δ
- Partial synchrony [DLS88] *timer approach*
  1. Either Δ (δ) is known but holds only eventually, or
  2. Δ (δ) exists but is not known.
- Relative speed [MMR03] *timer-free approach*
  – Constraints on the message pattern (message delivery order)
  – e.g., some processes always response among the first ones

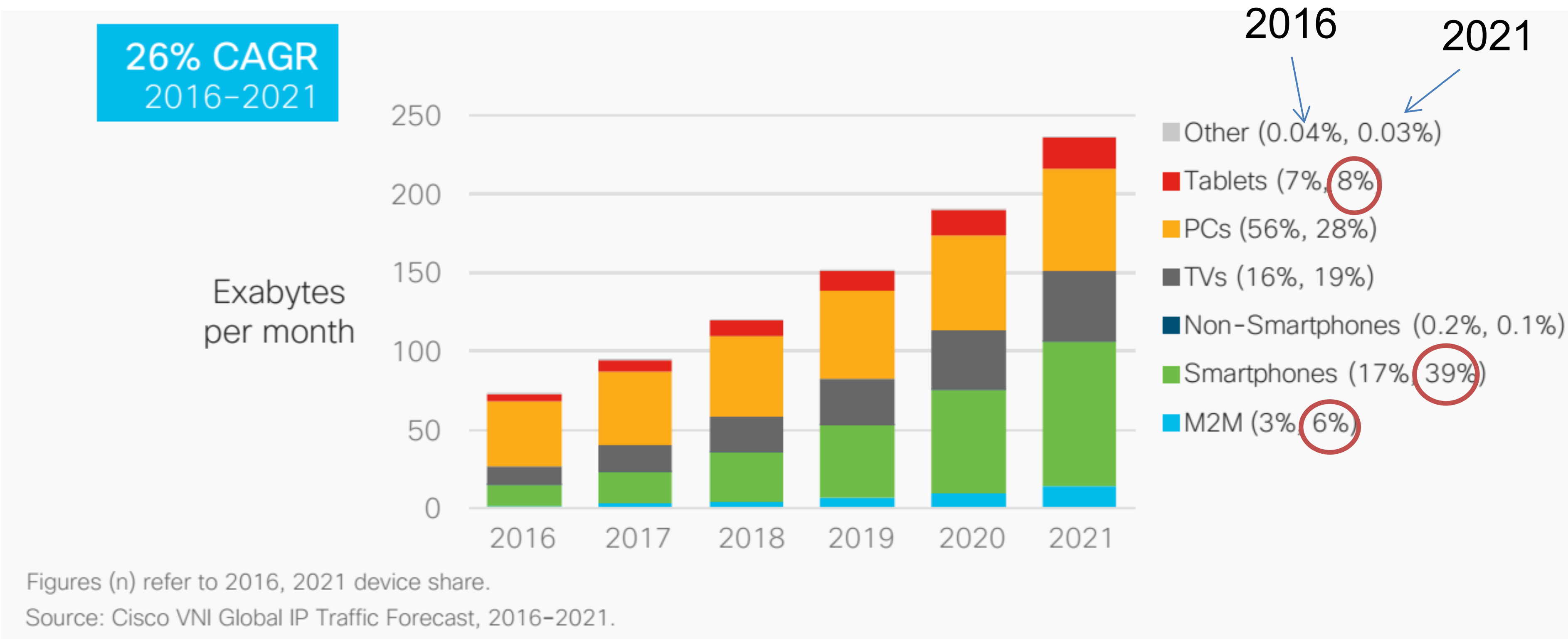
Limits of current implementations

- Many implementations of FD target **static** systems
  - Membership and topology are known

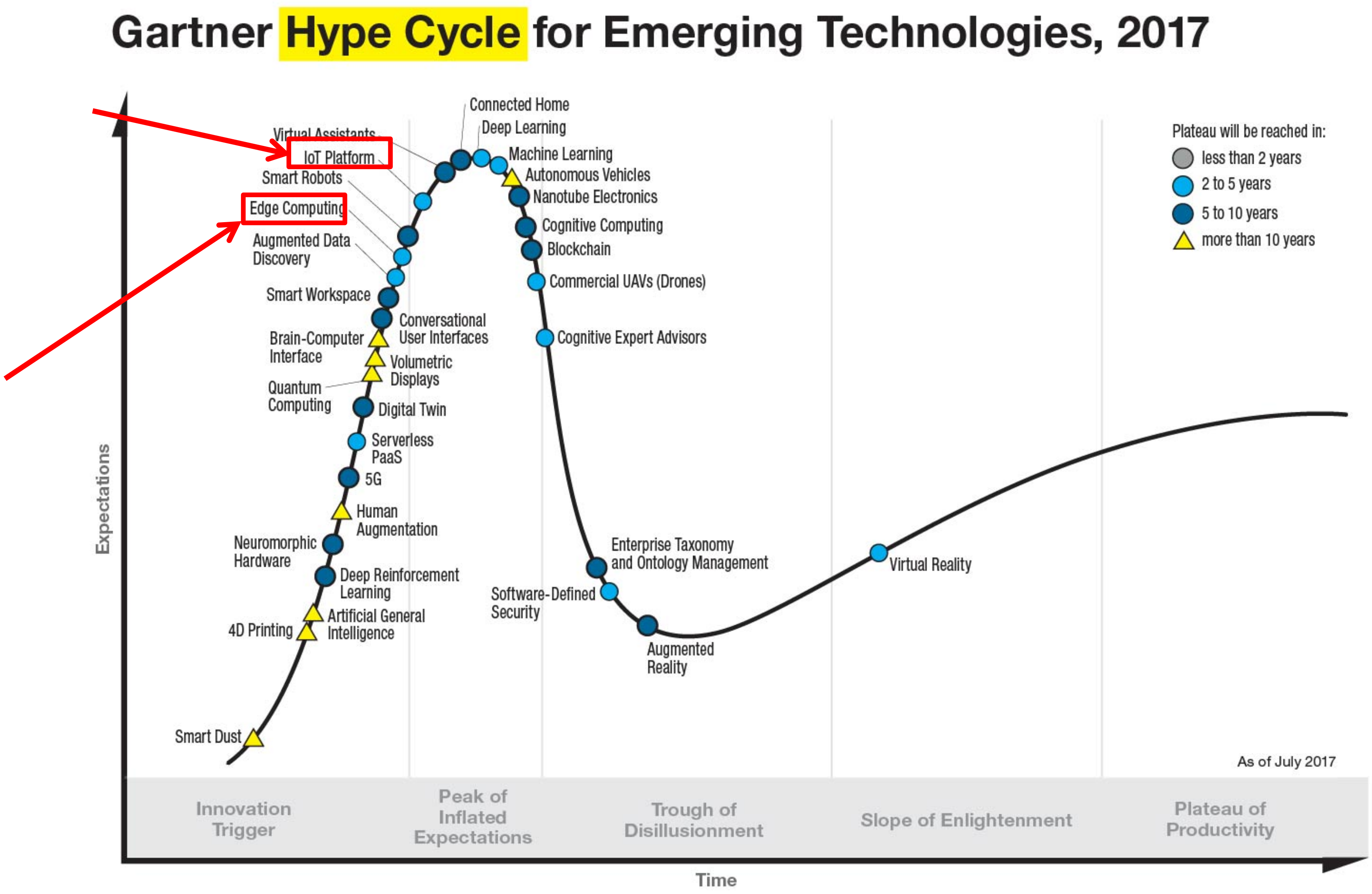
- Scalability

Distributed systems are more and more dynamic

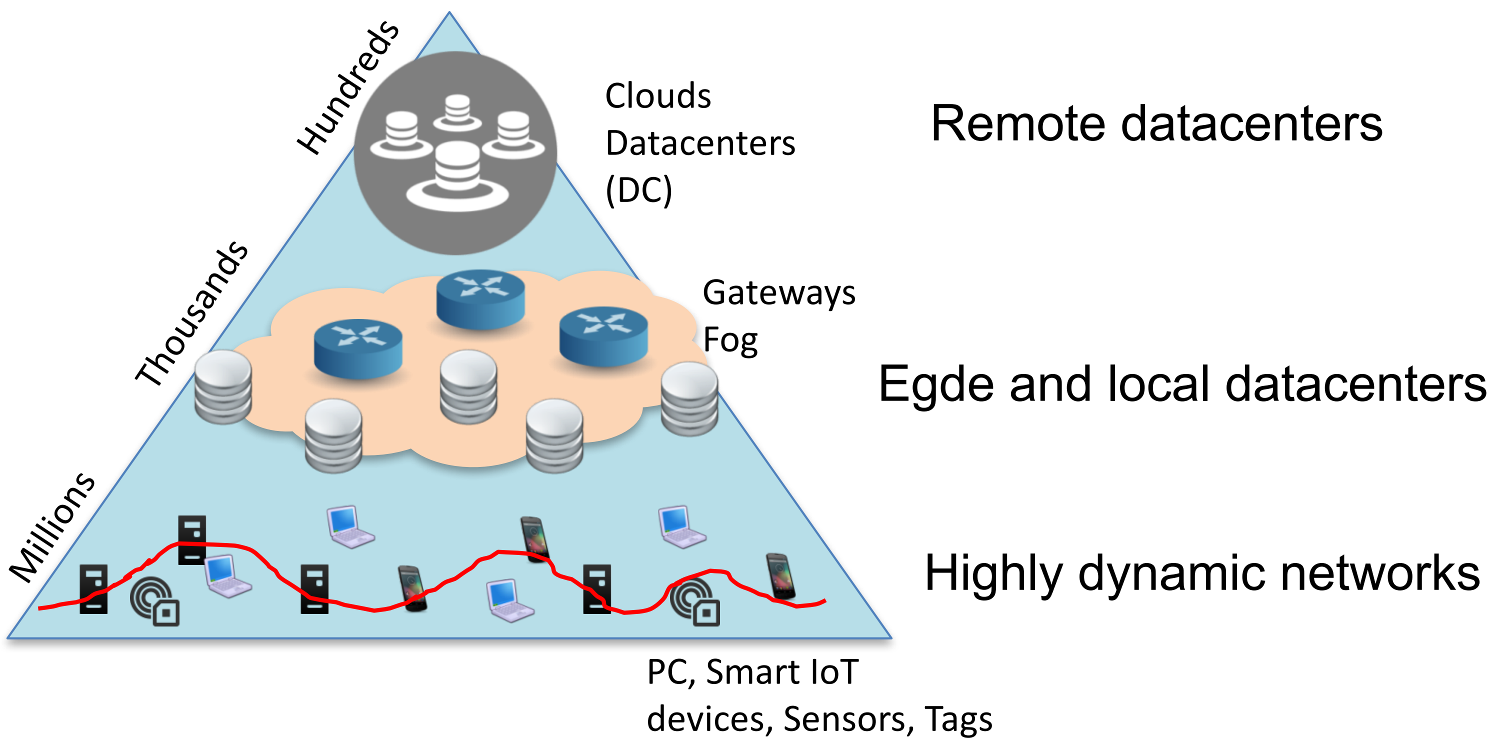
- In 2021, mobile devices will account for a half of global internet traffic



Edge computing and IoT emerging



New distributed architectures



Features of large and dynamic distributed systems

- **Asynchronous** network
  - No bound on transmission delays
- **Huge** number of resources
  - >1M nodes
- **Dynamicity**
  - Churn: Permanent arrival and leave of nodes
  - Mobility: Devices, virtual machines … can move or migrate
  - High failure rate, failure = common event
- “Chaotic” systems with no global state

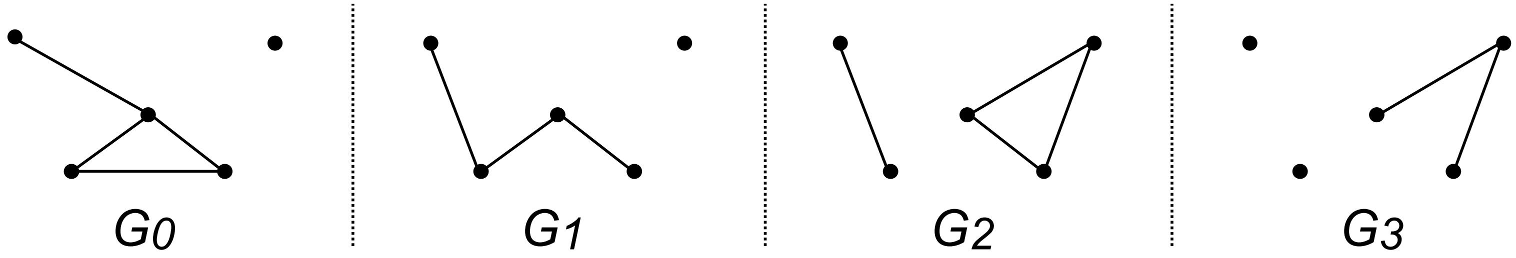
Models for dynamic systems

- Toward more dynamics : Infinite arrival models
  - Processes can be up or down
  - The number of up processes in any interval of time is upperly bounded by a **known constant C**

- Dynamic networks : dynamic graphs

Graph Representation

- Sequence Based [B. Bui-Xuan, A. Ferreira, A. Jarry, JFCS 2003]

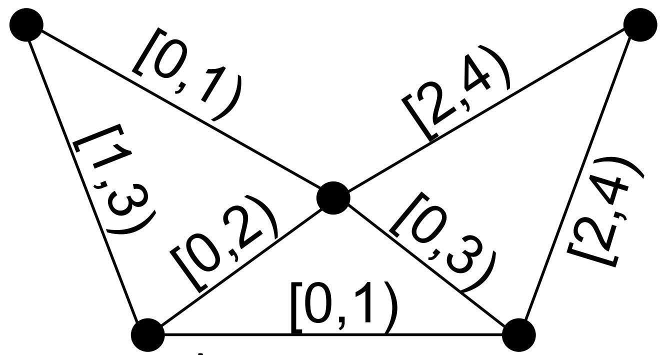


$\mathcal{G} = G_0, G_1, G_2, G_3, \ldots G_i, \ldots, \; i \in \mathbb{N}$

- Time varying graphs (TVG) [A. Casteigts, P. Flocchini, W. Quattrociocchi, N. Santoro, 2012]

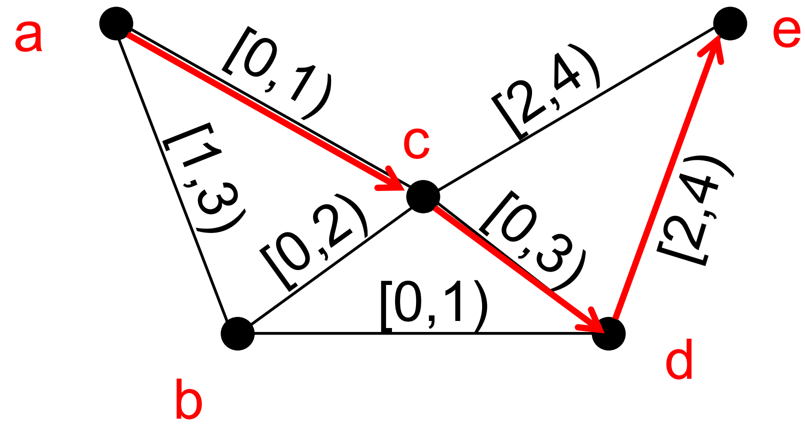
G = (V, E), lifetime $\mathcal{T}$

- *Presence function* $\rho : E \times \mathcal{T} \rightarrow \{0,1\}$
- *+ other functions (latency, node presence, …)*



© A. Casteigts

TVG: Basic Properties



- *Temporal path (a.k.a Journey), e.g.,* $a \leadsto e$

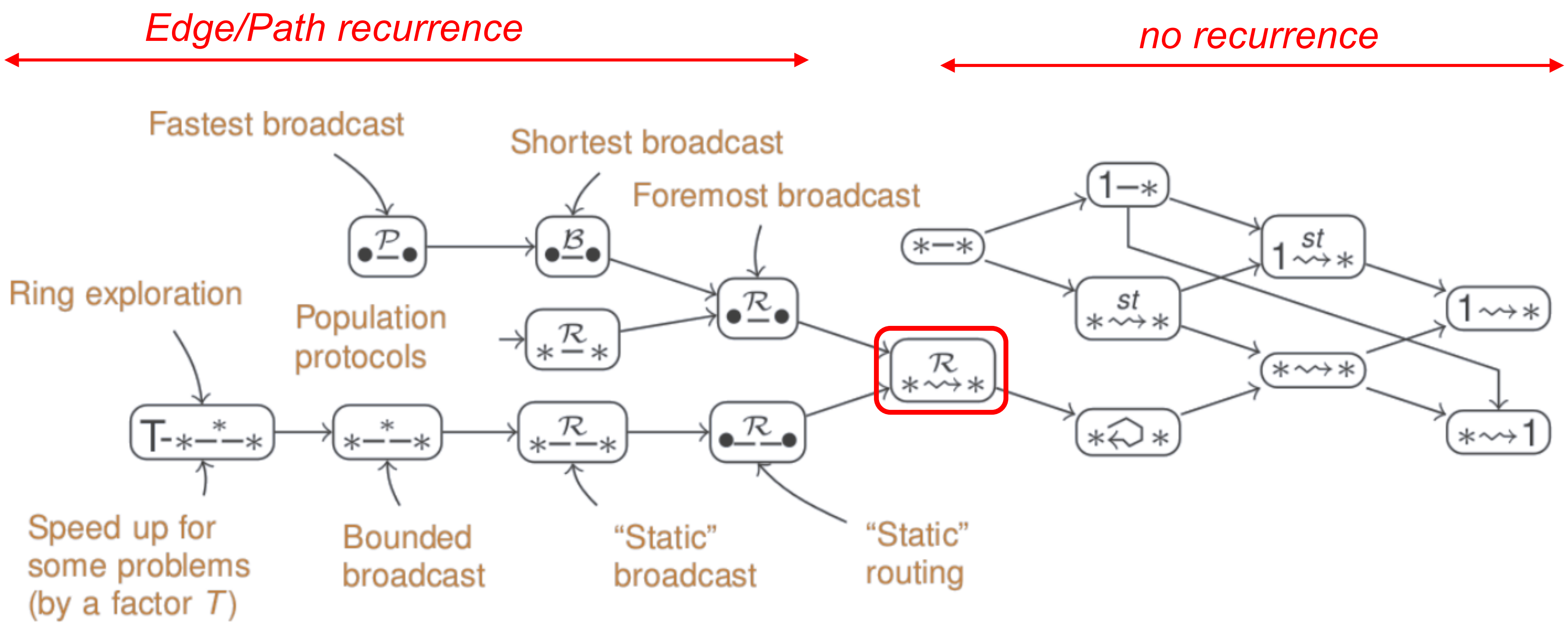
  $a \leadsto *$, $b \leadsto *$, $c \leadsto *$, $d \leadsto *$, except e!

- $1 \leadsto *$ $\exists u \in V, \forall v \in V, u \leadsto v$
- $* \leadsto 1$ $\forall u \in V, \exists v \in V, u \leadsto v$
- $* \leadsto *$ $\forall u, v \in V, u \leadsto v$

TVG: Classes

- $u \overset{P}{\rightsquigarrow} v$ - Periodic journey
- $u \overset{B}{\rightsquigarrow} v$ - Bounded journey
- $u \overset{R}{\rightsquigarrow} v$ - Recurrent journey

What assumption for what problems ? © Casteigts



Eventual Leader Election in Dynamic Environments

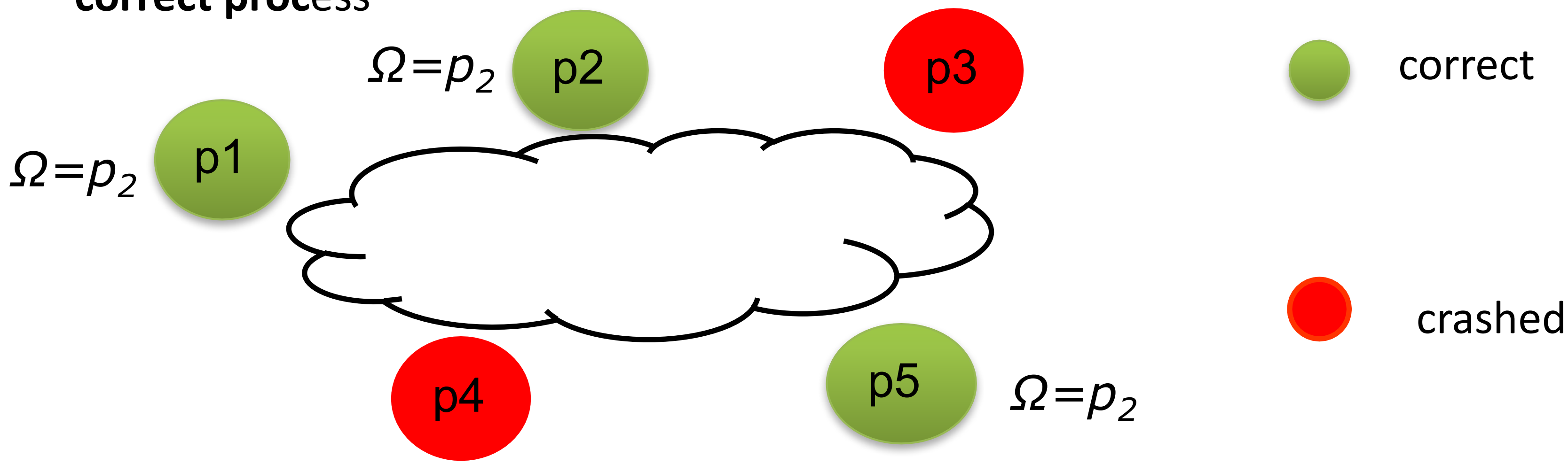
Luciana Arantes[1], Fabiola Greve[2], Véronique Simon[1], and Pierre Sens[1]

LIP6, Inria, France[1]

Federal University of Bahia (UFBA), Brazil [2]

Eventual leader election (Ω : omega failure detector)

- The Ω failure detector satisfies ("eventual leader election"):
  - there is a time after which **every correct process** always trusts **the same correct proc**ess



Context

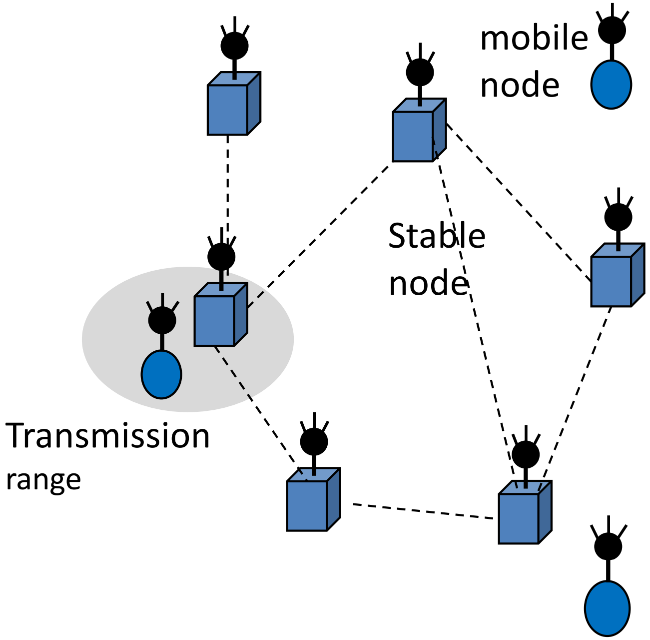
- Dynamic self-organized systems
  - Multi-hop networks (e.g. wireless ad-hoc networks)
    - broadcast /receive messages to/from neighbors within transmission range
- Communication
  - Channels are **fair-lossy**
  - there is no message duplication, modification or creation
- The system is **asynchronous**
  - There are no assumptions on the relative speed of processes nor on message transfer delays.
- Failure model : **crashes**
- The membership is **unknown**
  - A node is not aware about the set of nodes nor the number of them.
- Nodes have partial view of the network

Dynamics of the network

- Dynamic changing topology
  - join/leave of nodes,
  - mobility of nodes, failure of nodes (crash)
  - Finite arrival model
    - The network is dynamically composed of infinite mobile nodes, but each run consist of a finite set of *n* nodes.

Processes status and network connectivity

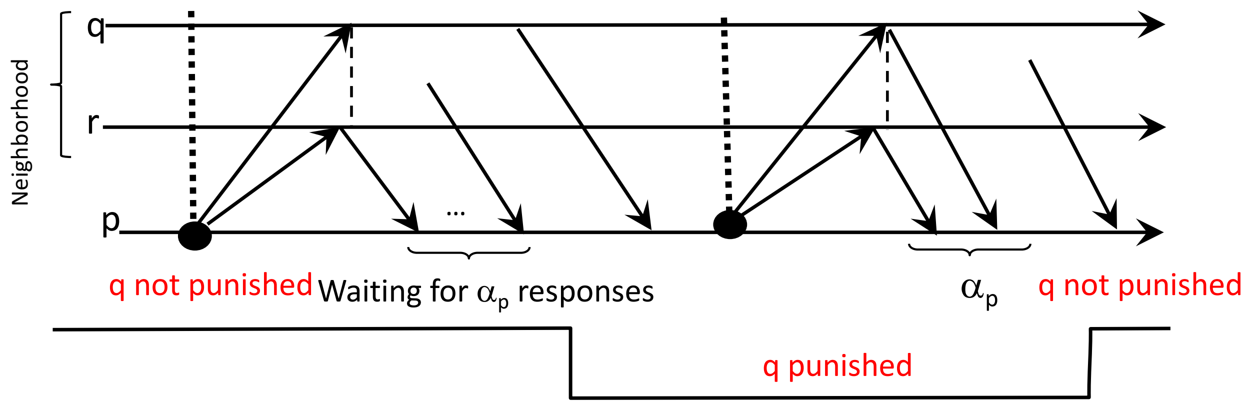
- Two sets of nodes:
  - STABLE (correct): nodes eventually and permanently correct
  - FAULTY: nodes which crash



- Network connectivity
  - Eventually, the *TVG* is connected over the time
    - There exists a journey between all stable nodes at any time
    - Network recurrent connectivity (**class** $* \overset{R}{\rightsquigarrow} *$ **)**

An Eventual Leader Election Algorithm

- Principle
  - Election of a leader process based on <u>**punishment**</u>
    - Round counter to control the freshness of the information
  - Periodic local **query-response** exchange
    - Wait for α responses
      - If q is <u>**locally known**</u> by p, <u>**has not moved**</u>, and <u>**does not respond**</u> to a query of p among α_p first responses, q is punished by p.



$$\alpha_i = |N_i^t| - f_i + 1$$

Implementation of Ω on dynamic networks

- Each node maintains 3 sets:
  - local_known: the current knowledge about its neighborhood
  - global_known: the current knowledge about the membership of the system
  - punish: a set of tuples <punish counter, node id>

  leader: the process with the smallest counter in punish set

- Diffusion of information over the network by p:
  - *p*'s current round counter
  - set of processes punished by *p*
  - current knowledge of *p* about the membership of the system

Additional properties

- *Stable Termination Property (SatP)*:
  - Each *QUERY* must be received by at least one stable and known node

    Necessary for the diffusion of the information

- *Stabilized Responsiveness Property (SRP):*
  - There exists a time *t* after which all nodes of *p* 's neighborhood receive, to every of their queries, a response from *p* which is always among the first responses

    ***SRP* should hold for at least one *stable* known node (the eventual leader)**

Leader Election: Sending of Query

```
Task T1: [Punishment]
Repeat forever
  | Wait until |recvfrom_i| > α_i
  | If ∀p_j : ⟨−, p_j⟩ ∈ local_known_i ∧ p_j ∉ recvfrom_i ∧ MaxKnown(p_j)   *
then
  |  | If ⟨0, p_j⟩ ∈ punish_i then
  |  |  | c_min ← min c : ⟨c, p⟩ ∈ punish_i, p ≠ p_j
  |  |  | replace in punish_i ⟨0, p_j⟩ by ⟨c_min + 1, p_j⟩
  |  | Else
  |  |  | replace in punish_i ⟨v, p_j⟩ by ⟨v + 1, p_j⟩
  | recvfrom_i ← ∅
  | mid_i ← mid_i + 1
  | broadcast QUERY(mid_i, punish_i, global_known_i)
End repeat
```

punishment

* - p_j is a neighbor of p_i,

- p_j does not answer to p_i,
- p_j is not suspected to have moved

Reception of Query and Response; Invocation of the Leader

Task T2: [Response]
upon reception of RESPONSE $(mid_j, punish_j, global\_known_j)$ from p_j

$UpdateState(mid_j, punish_j, global\_known_j, p_j)$*
$recvfrom_i \leftarrow recvfrom_i \cup \{p_j\}$

Task T3 [Query]
upon reception of QUERY $(mid_j, punish_j, global\_known_j)$ from p_j

$UpdateState(mid_j, punish_j, global\_known_j, p_j)$*
send RESPONSE $(mid_i, punish_i, global\_known_i)$ to p_j

Task T4 [Leader Election]
upon the invocation of *leader()*

return l such that $\langle c, l \rangle = Min(punish_i)$*

***update of p_i's state about punishment, membership, and p_i's neighborhood with more recent information : keeps the tuples with the greatest counter.**

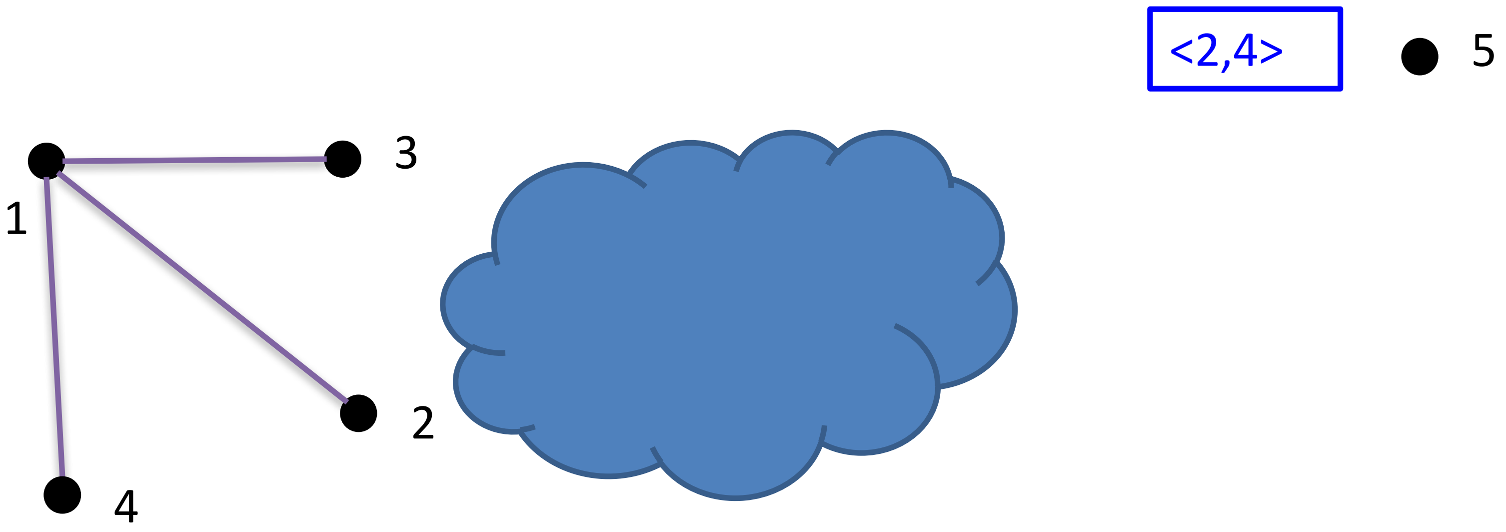
***process with the smallest counter**

Exemple: Mobility of nodes

global_known$_1$ <1,2>,<1,3>,**<2,4>**

punished$_1$ <0,1>,<0,2>,<0,3>,<**3**,4>

local_known$_1$ <1,1>,<1,2>,<1,3>,<1,4>



x:<x,4> in local_known$_1$ < y:<y,4> in global_known$_1$

1 stops punishing 4

Open issues : models

- Minimal condition in terms of time / connectivity / dynamicity to solve agreement problems
- Unified realistic model for distributed systems
  - Dynamicity, heterogeneity of nodes
- Adversary models (omission, byzantine failures)

Open issues: distributed algorithms

- Non deterministic algorithms

- Probabilistic algorithms / Indulgent algorithms

    - Ensure safety properties (eg. agreement)
    - Relax liveness properties (termination)

Open issues: experiments

- Need of testbeds to validate algorithms (Silecs initiative)
- Realistic mobility patterns
- Reproducible experiments

Concluding remarks

Distributed systems are **dynamic**

Failure detection a key component to build reliable application

Unreliable FDs

- A clear extension of asynchronous model
- A tool to build services in asynchronous network