
A guided visit to
DISTRIBUTED COMPUTABILITY

Eli GAFNI[†] **Michel RAYNAL^{*}** **Corentin TRAVERS^{*}**
eli@ucla.edu raynal@irisa.fr ctravers@irisa.fr

[†]**Computer Science Dpt, UCLA, USA**
^{*}**IRISA, Université de Rennes, France**

Content of the talk

- Why study distributed computability?
- Computation model
- Subconsensus tasks
- Fundamental question
- Results
- Conclusion

Consensus tasks

- The consensus problem
- Its universality
- The consensus nb hierarchy
- Example: ranking the power of synch primitives or base objects despite failures (Stack, queue, Compare&Swap, Test&Set, etc.)
- What about subconsensus tasks?

BASE COMPUTATION MODEL and THREE BASIC PROBLEMS (SUB-CONSENSUS TASKS)

Asynchronous process model

- A set Π of n processes p_1, \dots, p_n
- Process p_i : i is its **index**, its **identity** is id_i
- Timing model: **Asynchrony**: No upper bound on the time required to execute a computation step
- Failure model: up to t processes may **crash** ($1 \leq t < n$)
A **Correct process** is a p_i that never crashes
A **Faulty process** is a p_i that crashes
During an execution: f crashes ($0 \leq f \leq t$)
- Any number of processes can crash: **wait-free case**

Base Communication Model

Shared memory =
an array $SM[1..n]$ of **reliable**, **1W*R**, **atomic** registers

- **Reliability** means here that a register never crashes
- **1W*R** means that each register has a single writer (statically defined), but can be read by all the process
- **atomicity**: Each operation appears as being executed instantaneously at some point of the time line between its start event and its end event
- **Snapshot** operation: atomically reads the whole memory (can be implemented from 1W*R, atomic registers)

One-shot Test&Set object

A (n, k) -TS object provides the processes with a single operation, denoted $TS_compete_k()$

One-shot means that, given such an object, each process invokes that operation at most once

- **Termination**: An invocation issued by a correct process terminates
- **Validity**: The value returned by an invocation is 1 (winner), or 0 (loser)
- **Agreement**: $1 \leq \text{nb of winners} \leq k$

Set-Agreement object

A one-shot (n, k) -SA object allows the processes to propose values and decide a value. It provides the processes with the operation $SA_propose_k(v_i)$

- **Termination:** An invocation issued by a correct process terminates.
- **Validity:** A decided value is a value that has been proposed by a process
- **Agreement:** At most k distinct values are decided

Adaptive renaming (1)

- A (n, M) -adaptive renaming object allows a process to obtain a new name in the space $[1..M]$
- **Adaptive** means: the size M of the new name space depends only on the number p ($1 \leq p \leq n$) of processes that do participate in the renaming algorithm $M = f(p)$
- Properties:
 - ★ **Termination**: An invocation issued by a correct process terminates
 - ★ **Validity**: A new name belongs to the set $[1..M]$
 - ★ **Agreement**: $i \neq j \Rightarrow new_name_i \neq new_name_j$
No two invocations return the same new name

Adaptive renaming (2)

- Lower bound in RW systems (wait-free): $M = 2p - 1$
- Here: two types of adaptive renaming objects
 - ★ $M = \boxed{f_k(p) = 2p - \lceil \frac{p}{k} \rceil}$
 - ★ $M = \boxed{g_k(p) = p + k - 1}$
 - ★ Why consider these two values of M ? (see later)

A FUNDAMENTAL QUESTION

A fundamental question

- Given k , how are related the following problems/objects
 - ★ (n, k) -TS
 - ★ (n, k) -SA
 - ★ $(n, 2p - \lceil \frac{p}{k} \rceil)$ -AR denoted (n, f_k) -AR
 - ★ $(n, p + k - 1)$ -AR denoted (n, g_k) -AR
- Is it possible to solve any of them from the other ones?
- Are some of them more difficult to wait-free solve?
- Is it possible to rank them?
- etc.
- **Ranking is** (a part of) **our job!**

Transformation requirements

- **Index independence:**

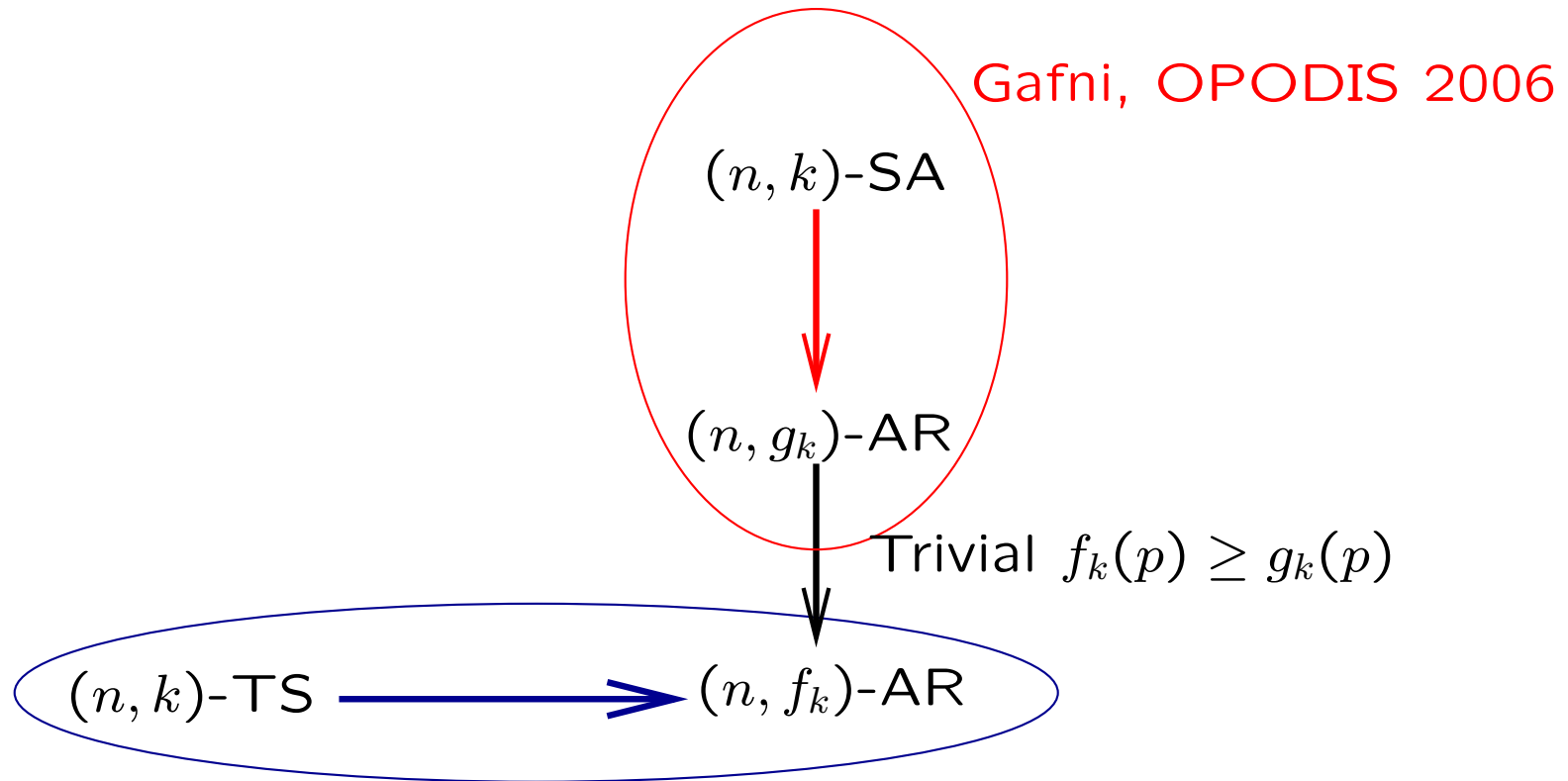
The behavior of a process is independent of its index

- Motivation

- ★ This property states that, if, in a run, a process whose index is i obtains a result v , that process would have obtained the very same result if its index was j
- ★ **operational point of view:** the indexes define an underlying communication infrastructure, namely, an addressing mechanism that can only be used to access entries of shared arrays

RESULTS

Previous results

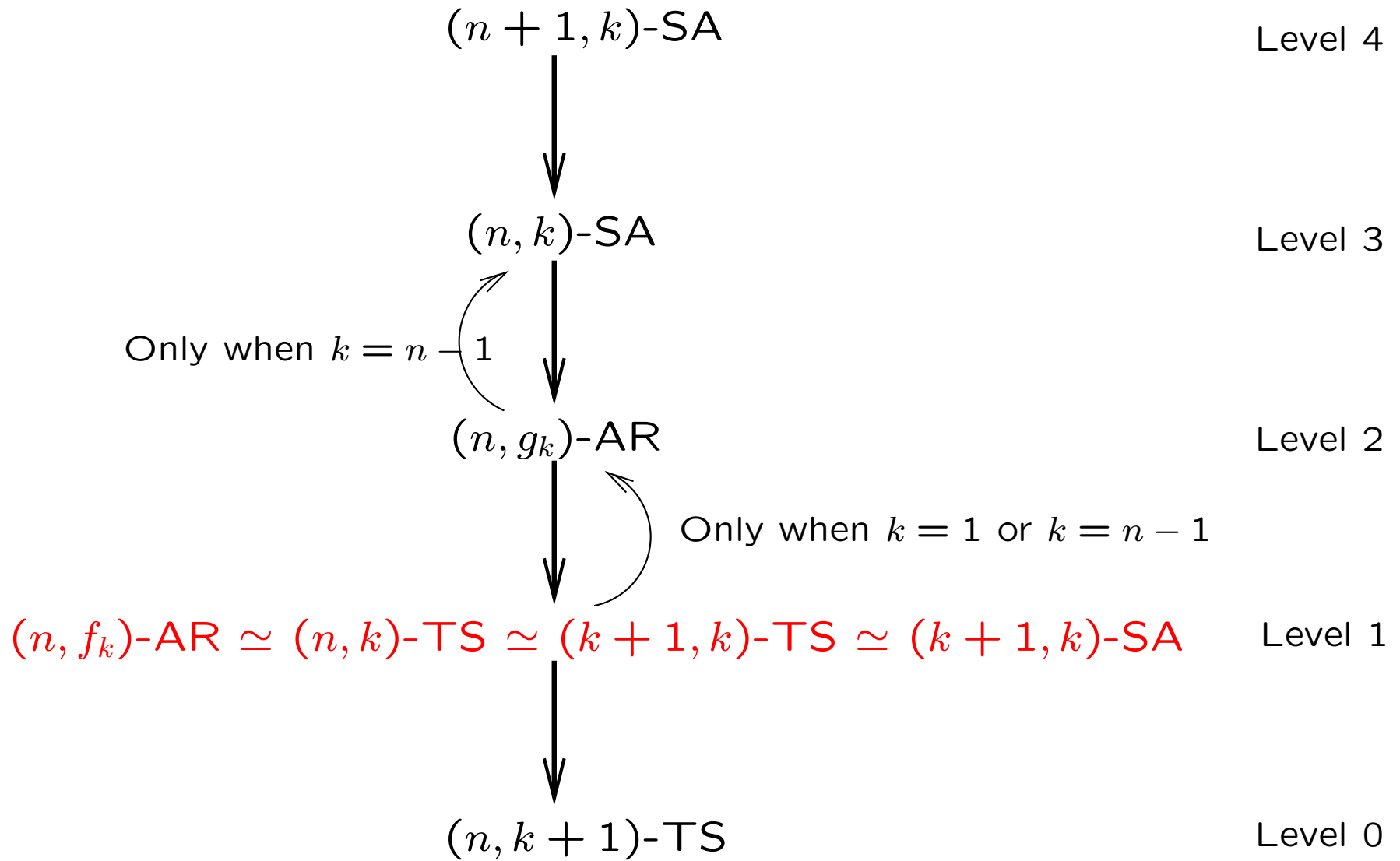


Gafni, OPODIS 2006

Trivial $f_k(p) \geq g_k(p)$

Mostéfaoui, Raynal and Travers, DISC 2006

A Hierarchy



Paper results

$$(n, f_k)\text{-AR} \simeq (n, k)\text{-TS} \simeq (k + 1, k)\text{-TS} \simeq (k + 1, k)\text{-SA}$$

- From $(n, f_k)\text{-AR}$ to $(n, k)\text{-TS}$
- $(n, k)\text{-TS}$ and $(k + 1, k)\text{-TS}$ are equivalent
- $(k + 1, k)\text{-SA}$ and $(k + 1, k)\text{-TS}$ are equivalent

A simple transformation

From an (n, k) _SA object to an (n, k) _TS object

operation $TS_compete_k()$

$REG[i] \leftarrow KS.SA_propose_k(id_i);$

$reg_i[1..n] \leftarrow REG.snapshot();$

if $(\exists x : reg_i[x] = id_i)$

then $res_i \leftarrow 1$ **else** $res_i \leftarrow 0$ **end if;**

return (res_i)

A more sophisticated transformation

From an (n, t) -TS object to an (n, t) -SA object

operation $SA_propose_t(v_i)$:

$REG[i] \leftarrow v_i$;

$COMPETING[i] \leftarrow KTS.TS_competet_t()$;

repeat $competing_i \leftarrow COMPETING.snapshot()$

until $(|\{j : competing_i[j] \neq \perp\}| \geq (n - t))$;

let $winner_i = \{j : competing_i[j] = 1\}$;

if $winner_i \neq \emptyset$

then $l_i \leftarrow$ any value $\in winner_i$

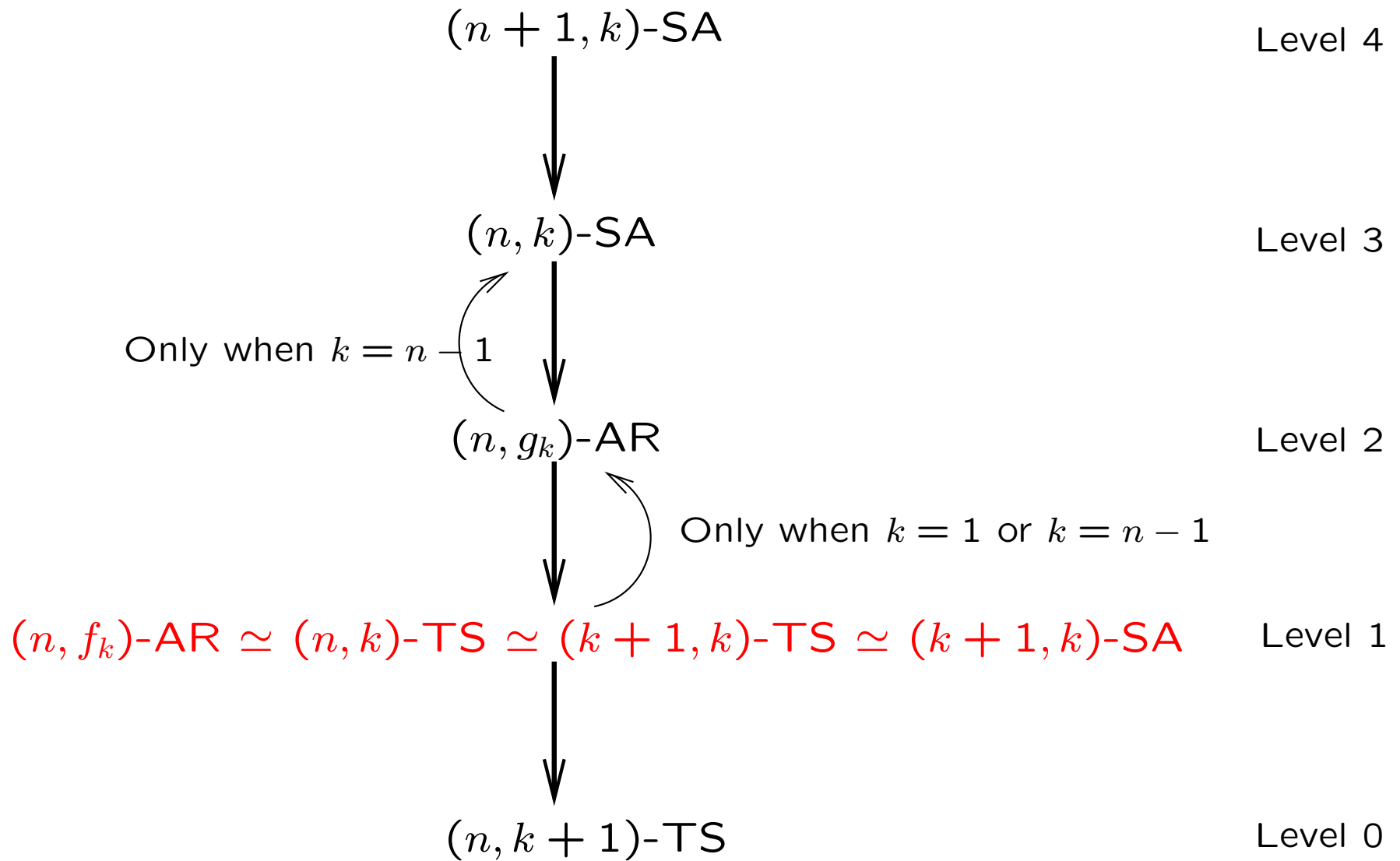
else let $set_i = \{j : REG[j] \neq \perp \wedge competing_i[j] = \perp\}$;

$l_i \leftarrow$ any value $\in set_i$

end if;

return $(REG[l_i])$

A Hierarchy



More generally

- Which problems can be wait-free solved in systems enriched with base objects richer than RW registers?
- Is there a hierarchy for subconsensus problems?
- Is it possible to define a measure (similar to consensus number) for these problems?

Conclusion

- Visit three important problems of fault-tolerant asynchronous computing
- Aim: better understand the fundamental difficulty created by the net effect of asynchrony and failures
- Elements for establishing a hierarchy
- How to generalize

$$(n_1, k_1)\text{-SA} + (n_2, k_2)\text{-SA} \rightarrow (n_1 + n_2, k_1 + k_2)\text{-SA}$$

Look for more general and non-trivial “additions”

$$(n, k)\text{-SA versus } (n - 2, k - 1)\text{-SA ??}$$

Conclusion cont'd

- Which is (are) the most basic fundamental problem(s)?
- In our quest, we have to be guided by computability and complexity (NP-completeness) in sequential computing
- Our job is to understand and to state problems (and their solutions if any) in a way **as simple as possible**