# Overview of WP2

# Tradeoff between information and efficiency

Laurent Feuilloley

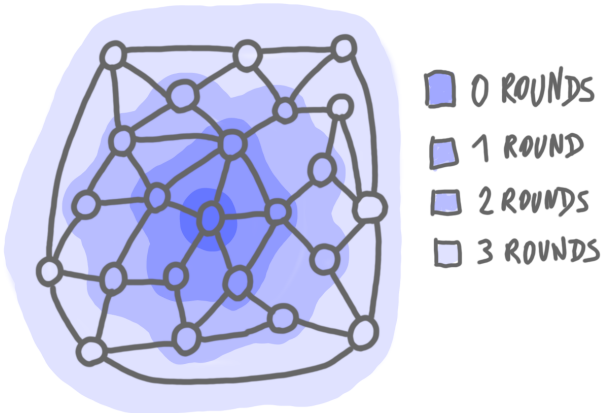
ENEDISC Kick-off meeting · IRIF, Paris · February 2025

→ **Background**

Tasks

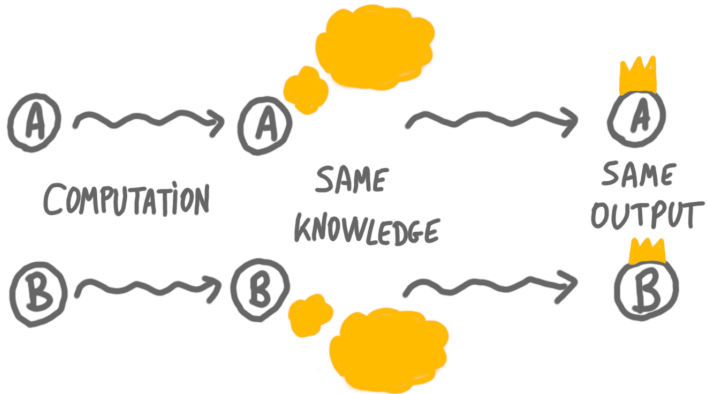
# Information in distributed computing

Often: **distributed computation** $\approx$ **gathering enough information** to output.



# Information in distributed computing

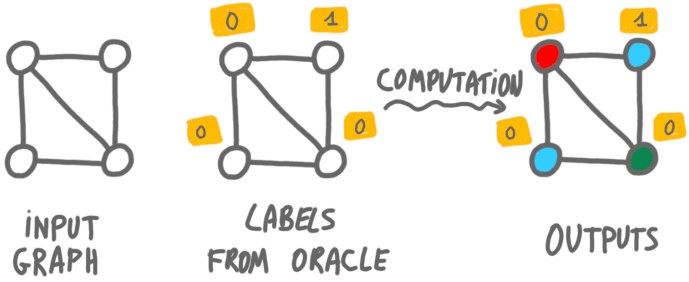
Often: **distributed computation** $\approx$ **gathering enough information** to output.



# Information from an external source

A way to study information itself:

- Give every node **additional information** as a **label** from an **external source**.
- How different is the computation?




This label **does not** allow to completely **solve the problem**: because it is **non-trustable**, or **too small**, or ... Several flavors: advice, certificates, predictions.

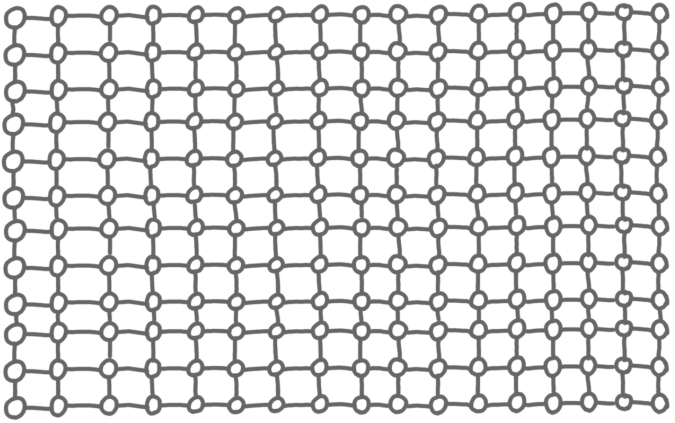
# Flavor 1: Advice

**Advice setting:**

- Originate from complexity and online algorithms.
- Usually for construction problems (not decision problem).
- A piece of advice is **trustable**. (The oracle is your friend)
- The piece of advice is **very small**. Cannot encode the output.
- It usually helps to **speed up computation** or **improve the quality/approximation**.
- Can also be seen as a **compression** of the output or best decision to take.

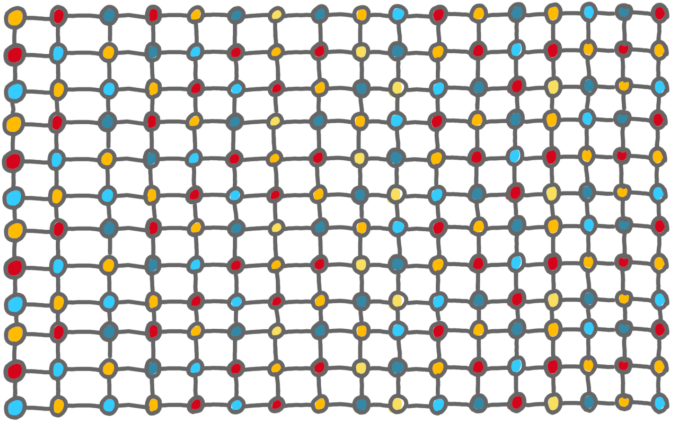
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



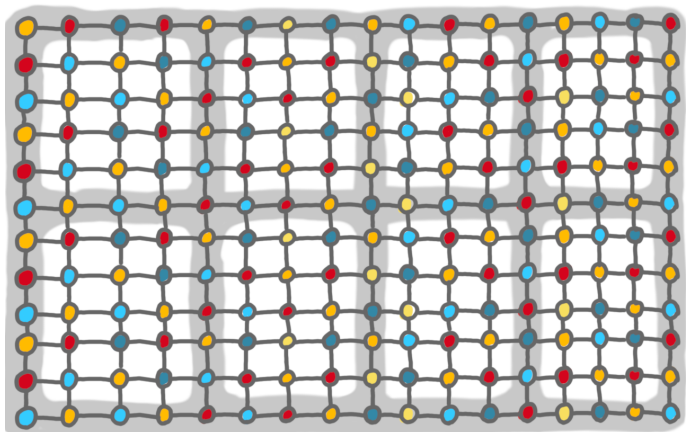
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



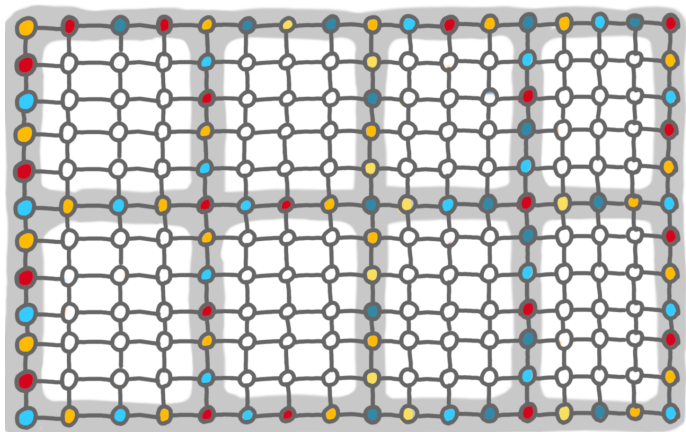
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



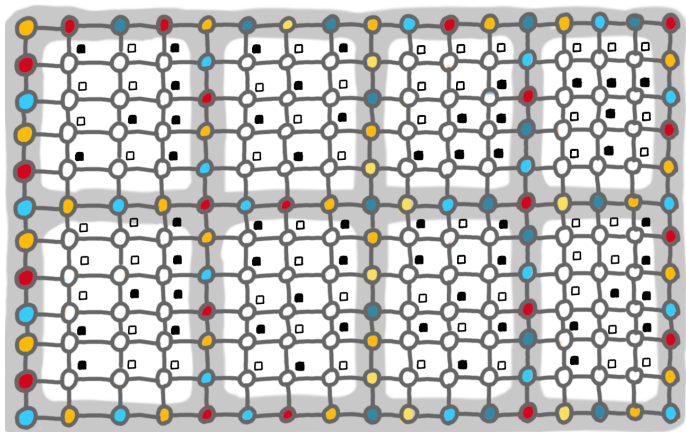
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



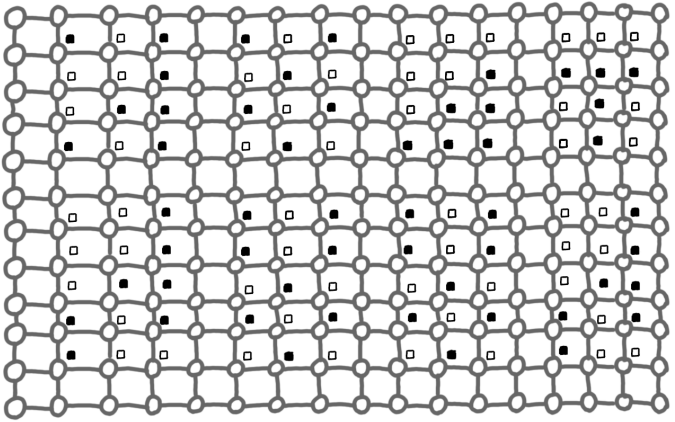
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



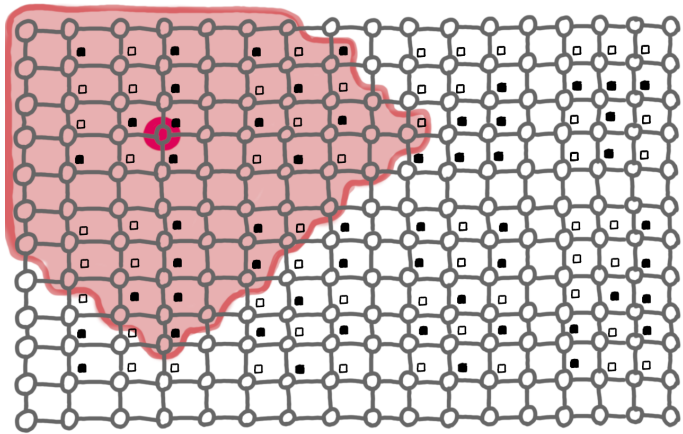
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



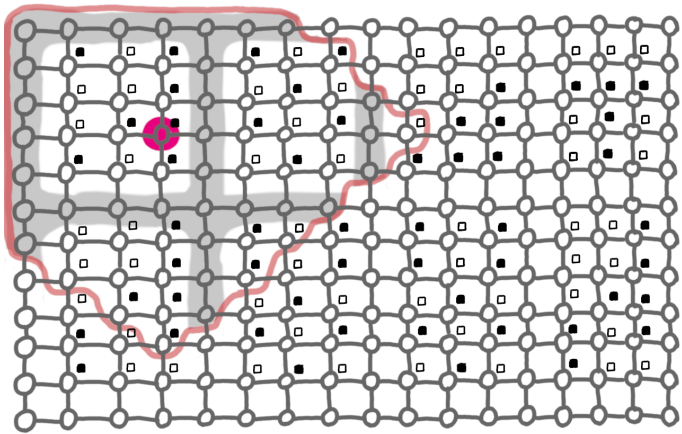
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



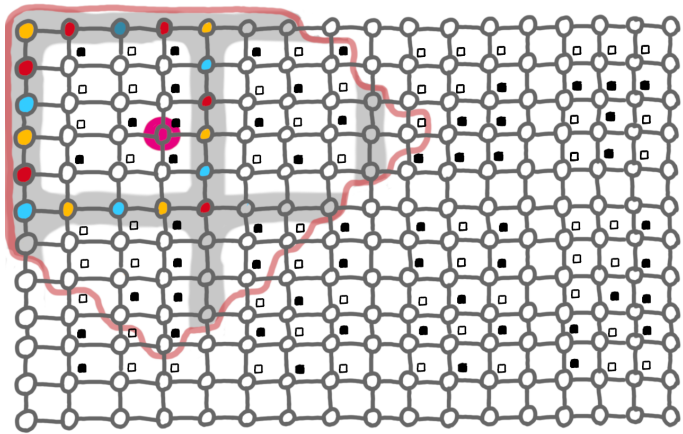
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



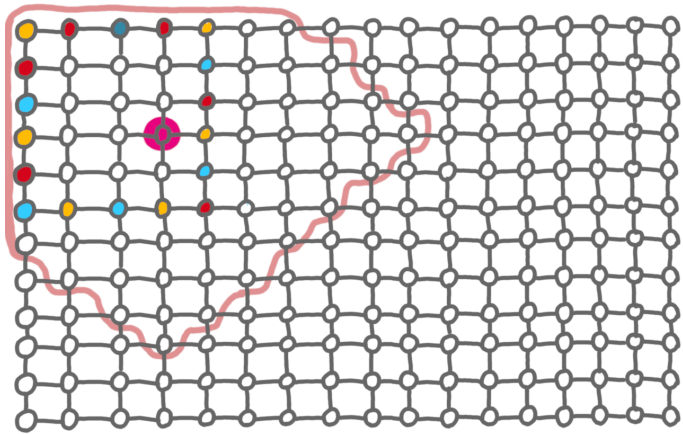
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



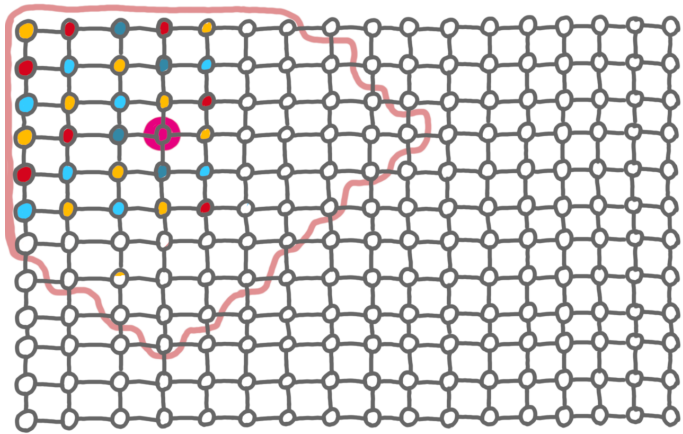
# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



# Advice in distributed computing

**Example**: $(\Delta + 1)$-coloring with $\ll \log(\Delta)$ advice bits (if sublogarithmic growth).
**Oracle:** Computes a coloring, clusters the graph, encodes borders colors.
**Decoder:** Find the borders, decodes the colors, fills in the gaps.



# Flavor 2: Local certification

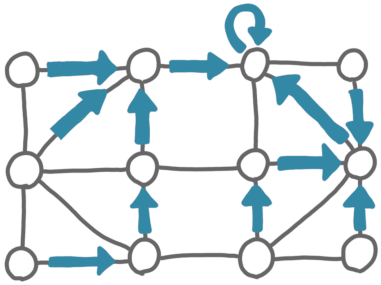
- Usually for **decision problems**: checking a graph property or the output of an algorithm.
- In a **good instance**, **all nodes** should **accept**
  In a **bad instance**, **at least one node** should **reject**
- The labels are **certificates**, that are **not trustable**.
  (The oracle is a *prover*, trying to make all nodes accept.)
- The nodes run a **verifier algorithm** which checks the certificates.

a.k.a proof-labeling schemes, locally checkable proofs, distributed certification.

# Flavor 2: Local certification

**Example: checking a spanning tree**

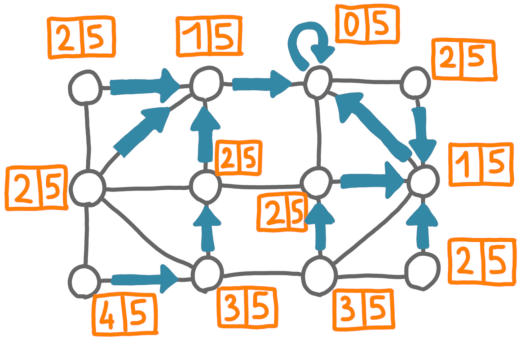
- On good instances, certificates:
  - Distance to root.
  - ID of the root.
- Checking distances consistency
  $\rightarrow$ ensures acyclicity.
- Checking root-ID consistency
  $\rightarrow$ ensures connectivity.



# Flavor 2: Local certification

**Example: checking a spanning tree**

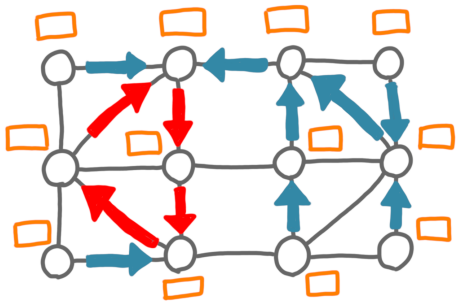
- On good instances, certificates:
  - Distance to root.
  - ID of the root.
- Checking distances consistency
  $\rightarrow$ ensures acyclicity.
- Checking root-ID consistency
  $\rightarrow$ ensures connectivity.



# Flavor 2: Local certification

**Example: checking a spanning tree**

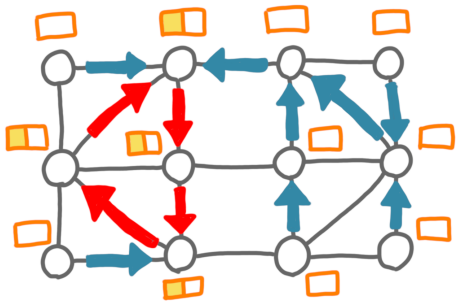
- On good instances, certificates:
  - Distance to root.
  - ID of the root.
- Checking distances consistency
  $\rightarrow$ ensures acyclicity.
- Checking root-ID consistency
  $\rightarrow$ ensures connectivity.



# Flavor 2: Local certification

**Example: checking a spanning tree**

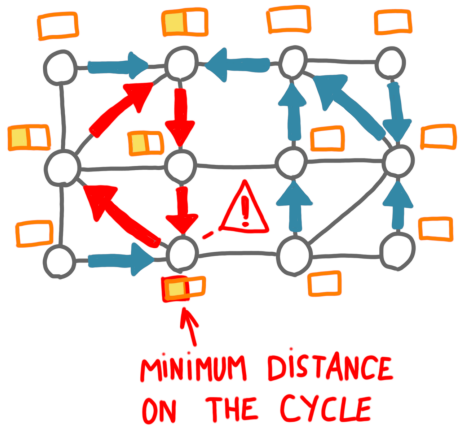
- On good instances, certificates:
  - Distance to root.
  - ID of the root.
- Checking distances consistency
  $\rightarrow$ ensures acyclicity.
- Checking root-ID consistency
  $\rightarrow$ ensures connectivity.



# Flavor 2: Local certification

**Example: checking a spanning tree**

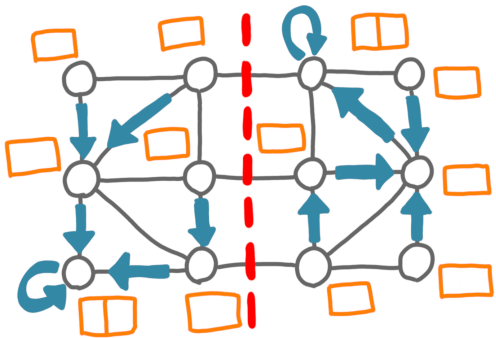
- On good instances, certificates:
  - Distance to root.
  - ID of the root.
- Checking distances consistency
  $\rightarrow$ ensures acyclicity.
- Checking root-ID consistency
  $\rightarrow$ ensures connectivity.



# Flavor 2: Local certification

**Example: checking a spanning tree**

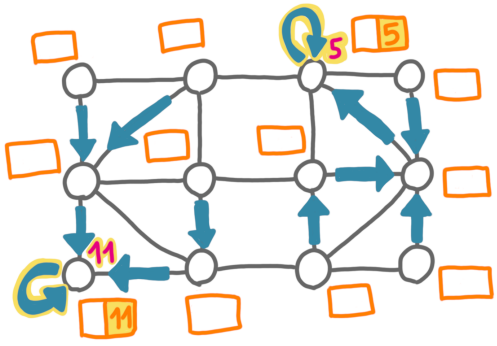
- On good instances, certificates:
  - Distance to root.
  - ID of the root.
- Checking distances consistency
  $\rightarrow$ ensures acyclicity.
- Checking root-ID consistency
  $\rightarrow$ ensures connectivity.



# Flavor 2: Local certification

**Example: checking a spanning tree**

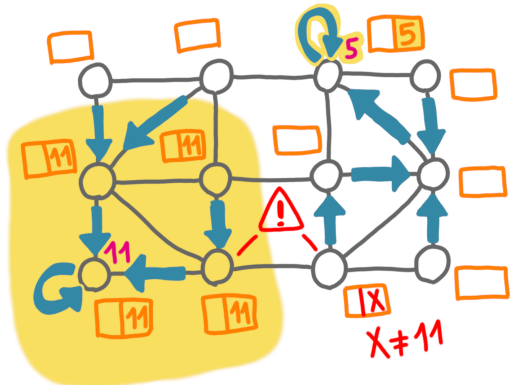
- On good instances, certificates:
  - Distance to root.
  - ID of the root.
- Checking distances consistency
  $\rightarrow$ ensures acyclicity.
- Checking root-ID consistency
  $\rightarrow$ ensures connectivity.



# Flavor 2: Local certification

**Example: checking a spanning tree**

- On good instances, certificates:
  - Distance to root.
  - ID of the root.
- Checking distances consistency
  $\rightarrow$ ensures acyclicity.
- Checking root-ID consistency
  $\rightarrow$ ensures connectivity.



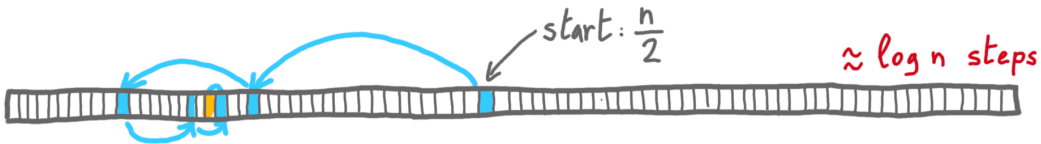
# Flavor 3: Predictions

- Originates from online algorithms, now also centralized and distributed.
- For **construction problems**.
- A prediction setting comes with a **quality measure**.
- In an algorithm with predictions should:
  - If the **prediction is perfect** then the **algorithm is very good**.
  - If the **prediction is garbage**, the **algorithm is ok**.
  - If possible, **degrades smoothly**.
- Typical example: **warm-start** a local search.

Related to another ANR projects (PREDICTIONS PI: Spyros Angelopoulos)

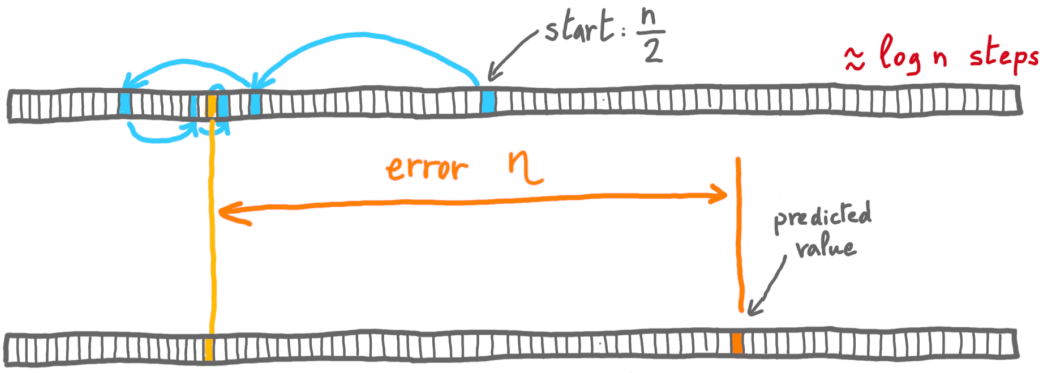
# Flavor 3: Predictions

**Example: Search with predictions**



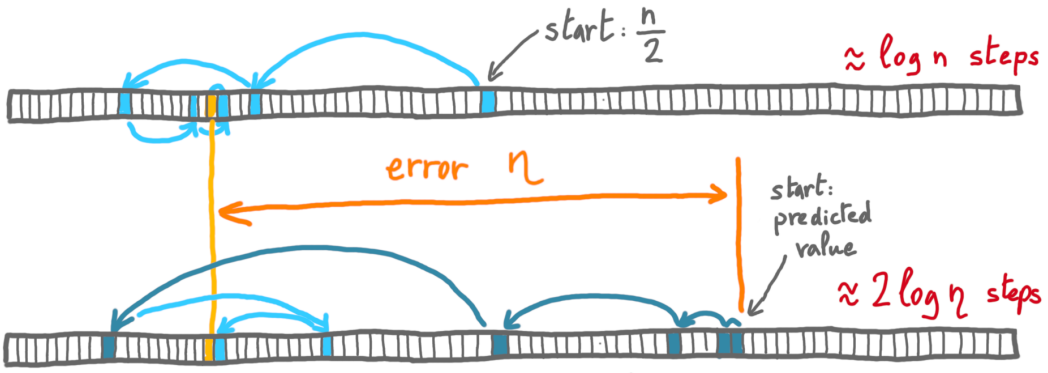
# Flavor 3: Predictions

**Example: Search with predictions**



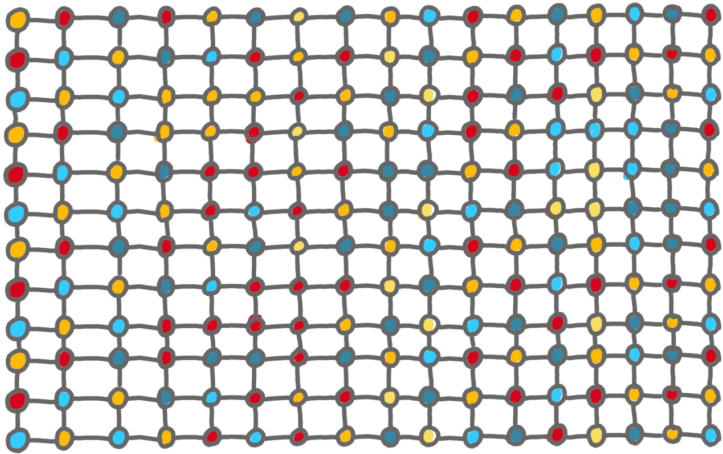
# Flavor 3: Predictions

**Example: Search with predictions**



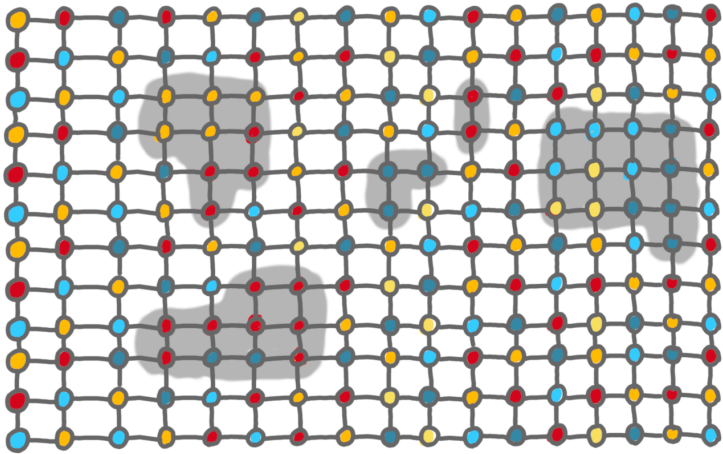
# Flavor 3: Predictions

**Example: distributed coloring**



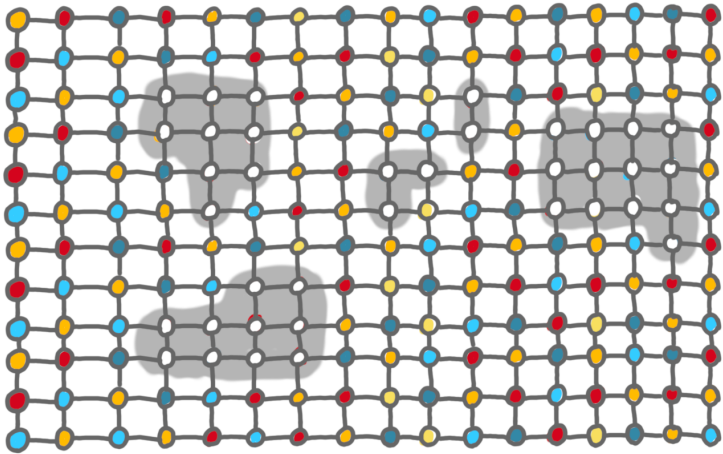
# Flavor 3: Predictions

**Example: distributed coloring**



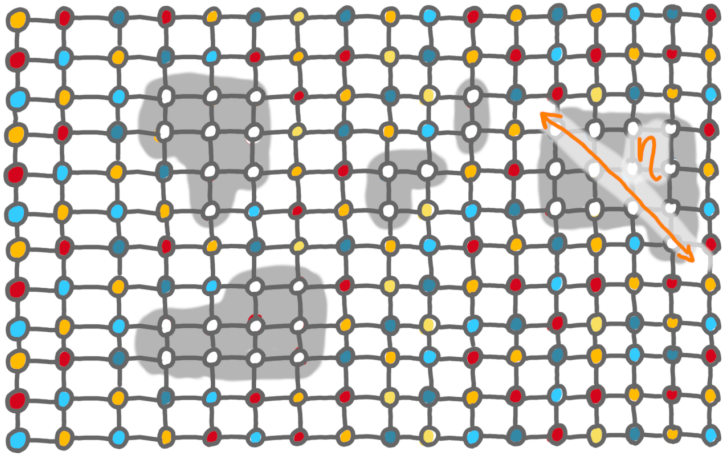
# Flavor 3: Predictions

**Example: distributed coloring**



# Flavor 3: Predictions

**Example: distributed coloring**



Background

→ **Tasks**

# Task 2.1: Awakening schedules

Typical shape of energy-efficient algorithms:

1. **Scheduling phase**: compute a good awakening schedule
2. **Solving phase**: use the schedule to solve the task.

**Goal:** We want to better **understand awakening schedules**.

# Task 2.1: Awakening schedules

An corner case to keep in mind.

**Problem:** $k$-coloring.

**Scheduling phase**:
Compute a $k$-coloring, and give to every node the schedule: "index-of-color".

**Solving phase**:
Every node wakes up at round index-of-color and takes color index-of-color.

# Task 2.1: Awakening schedules

Different **qualities of an awakening schedule**:

**can be computed fast** | generic | robust

Research directions:

- The natural approach: **balance scheduling complexity and solving complexity** for a specific problem.
- If **reusing the schedule**: we can **allow more resources for scheduling** phase, e.g. computation in CONGEST.
- Isolating the scheduling phase: **fix some constraint, find fastest algorithm** producing a schedule satisfying these constraints.

# Task 2.1: Awakening schedules

Different **qualities of an awakening schedule**:

can be computed fast | **generic** | robust

Research directions:

- If we **target one problem**, we can **tailor the schedule** to that problem.
- What about **more generic schedule?** For example for a **class of problem** defined by a logic or the locality. (DLT is one example.)
- Is there a **trade-off** between **generality** and **efficiency**?

# Task 2.1: Awakening schedules

Different **qualities of an awakening schedule**:

can be computed fast | generic | **robust**

Research directions:

- In classic sleeping algorithms, if the **schedule is not followed exactly**, the **output is completely broken**.
- In other words, **sleeping algorithms** are **brittle**.
- Can we make them more **time-shift tolerant**? Design **generic error-correcting techniques**?
- Can we design algorithms such that the **performance in solving phase degrades smoothly** with the number of "clock issues"?

# Task 2.2: Informative-labeling schemes

**Informative-labeling schemes** are yet another **model of labeling**.

**Example: distance labeling** $\rightarrow$ labels such that from $\ell_u$ and $\ell_v$, we can compute distance between $u$ and $v$.

**Focus: size of the encoding**

Research directions:

- **Computing** such **labels** in the **sleeping model**.
- **Minimize** size of the **schedule encoding**.
- Study the **dependency in the network topology**.

# Task 2.3: From algorithms to certification, and back

A **generic way** to design **local certification for data structures** (e.g. spanning trees) from a distributed algorithm building the it.

**For the prover:**

1. Run virtually the algorithm on the graph. (A run producing the given output.)
2. Give to every node the **full transcript** of the algorithm at that node: **for every round all the messages sent**.

**For every node:**

- **Get the messages sent by neighbors** in their certificates.
- Virtually **check the algorithm run**, round by round.
- Check that the given **output is consistent** with the run.

# Task 2.3: From algorithms to certification, and back

In general, the certificate size is only bounded by:

**Number of rounds** $\times$ **maximum degree** $\times$ **maximum message size**

$\rightarrow$ Much larger than the optimal in general. Can sometimes be compressed in an ad hoc manner, e.g. spanning tree.

We can get better bounds if:

- We **bound message size**: restrict to $O(\log n)$ or even $O(1)$.
- We consider a **broadcast model** (same message for all neighbors): then a node only needs to store one message per round.
- We **reduce the number of interesting rounds** $\rightarrow$ SLEEPING!

# Task 2.3: From algorithms to certification, and back

**Wannabe Theorem:** Efficient broadcast sleeping algorithm $\Rightarrow$ Small local certification.

**Contrapositive:** Lower bound for local certification $\Rightarrow$ Lower bound for broadcast sleeping algorithm.

Intuitively:

- Local certification captures the core information needed to verify a solution.
- The sleeping model removes some of the waste of information from LOCAL, when it comes to computation.

# Wrap-up

- One way to extract the notion of information is via labelings.
- Various types: advice, predictions, certificates, informative labelings.
- Task 2.1: Understand sleeping schedules better. Computation / universality / robustness.
- Task 2.2: Focus on encoding size. In particular encoding of schedules.
- Task 2.3: Transfer results between certification and sleeping algorithms.