Locality and distributed scheduling

Jukka Suomela
Aalto University, Finland

Distributed scheduling

- ***Centralized*** scheduling:
  - **input:** encoded as a string
  - **model of computing:** RAM model, Turing machines
  - **solution:** encoded as a string
- ***Distributed*** scheduling:
  - can mean **two different things**!

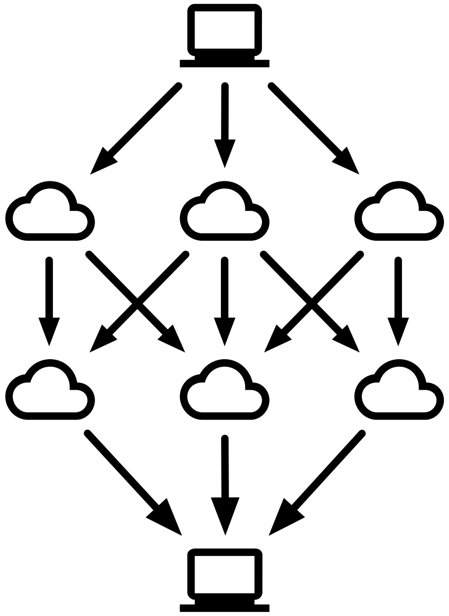
Big data perspective

“***Too large*** for my laptop to solve, I’ll have to resort to Amazon cloud”

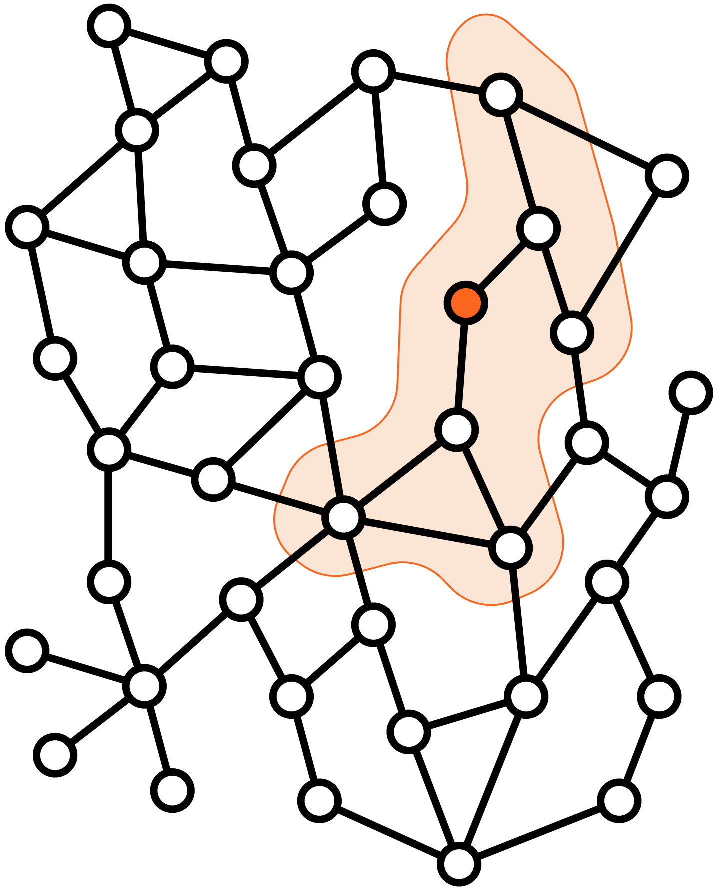
Network algorithms

“How to schedule radio transmissions in a large network ***without centralized control***?”

Big data perspective



Network algorithms



Big data perspective

- Focus: ***computation***
- Distributed perspective **helps** us

Network algorithms

- Focus: ***communication***
- Distributed perspective additional **challenge**

Big data perspective

- Fully centralized control
- ***Global*** perspective
- **Input & output in one place**

Network algorithms

- No centralized control
- ***Local*** perspective
- **Input & output distributed**

Big data perspective

- I know ***everything*** about input
- I need to know ***everything*** about solution

Network algorithms

- Each node knows its ***own part*** of input
  - e.g. local constraints
- Each node needs its ***own part*** of solution
  - e.g. when to switch on?

Big data perspective

- Explicit input
  - encoded as a string, stored on my laptop
- Well-known network structure
  - tightly connected cluster computer

Network algorithms

- Implicit input
  - ***input graph = network structure***
- Unknown network structure
  - e.g. entire global Internet right know

Big data perspective

Can we divide problem in **small independent tasks** that can be solved **in parallel**?

Network algorithms

If each node is only aware of its **local neighborhood**, can we nevertheless find a **globally consistent solution**?

Big data perspective

- Closely related to ***parallel algorithms***
  - **independent subtasks** that can be solved in parallel

Network algorithms

- Somewhat related to ***sublinear-time algorithms*** and property testing
  - making decisions **without seeing everything**

Big data perspective

- Computationally intensive problems
- Finding ***optimal*** solutions

Network algorithms

- Computationally easy problems
- Finding ***good*** solutions

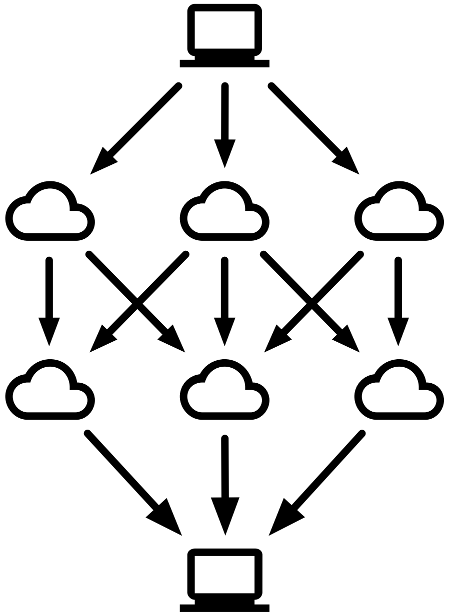
Big data perspective

- Models of computing:
  - **MapReduce**
  - bulk synchronous parallel (**BSP**)

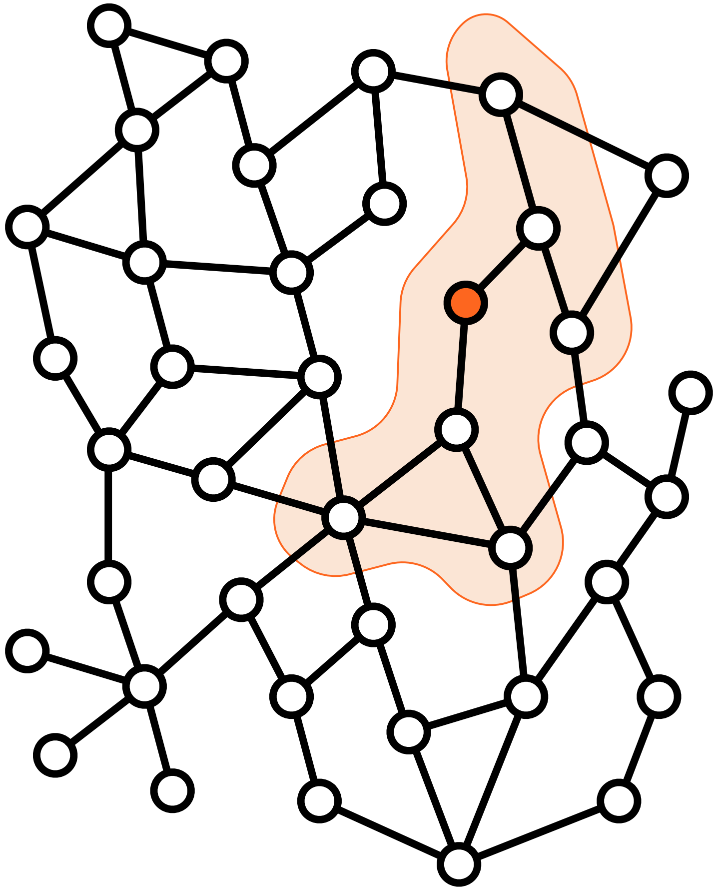
Network algorithms

- Models of computing:
  - **LOCAL**
  - **CONGEST**

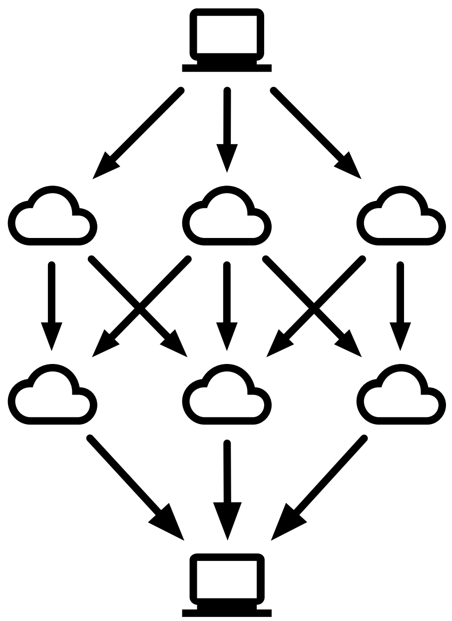
Big data perspective



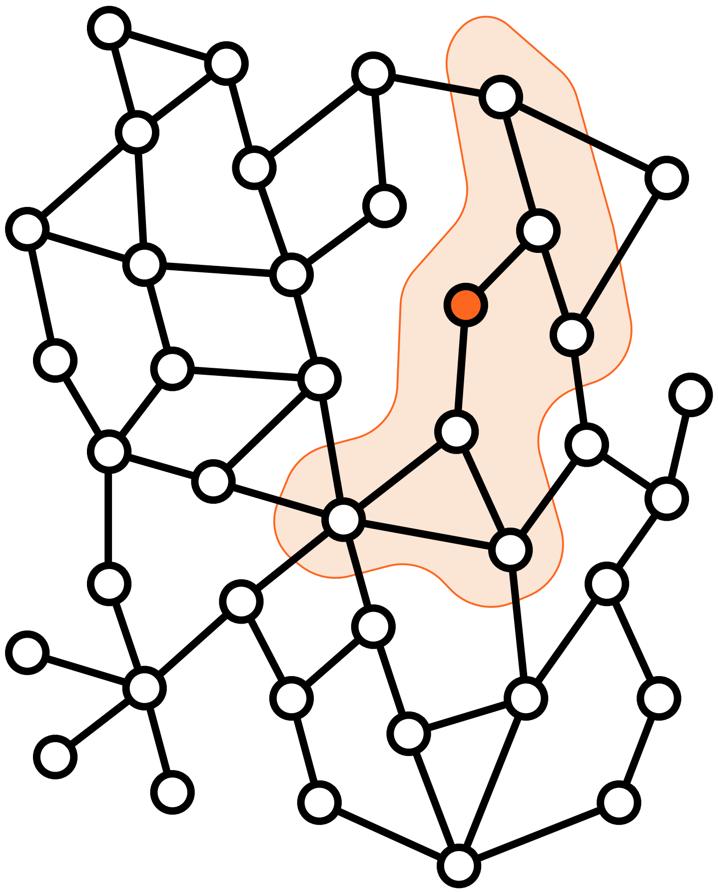
Network algorithms



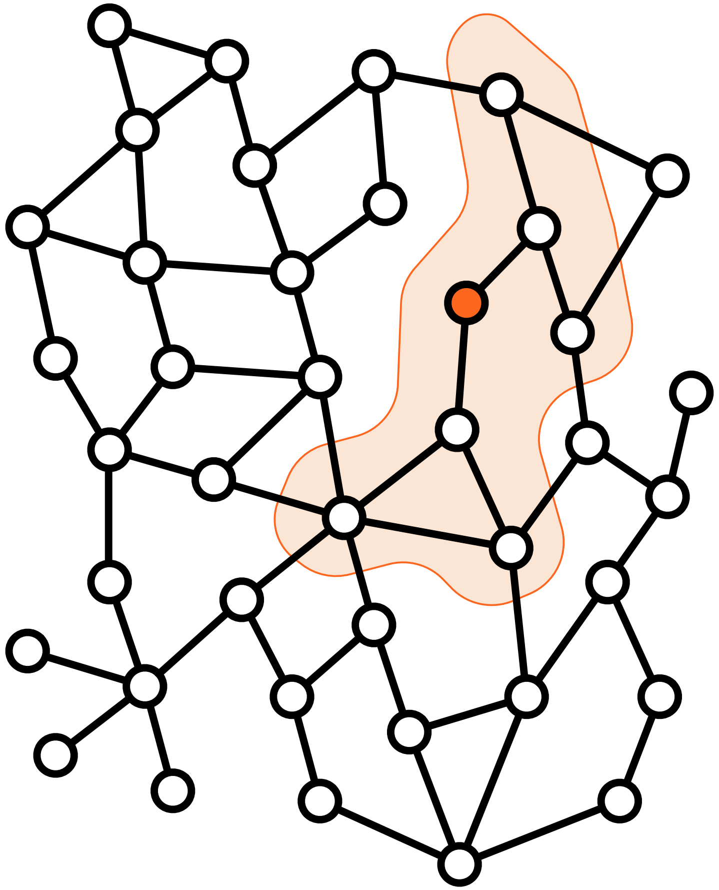
Big data perspective



Network algorithms



LOCAL model

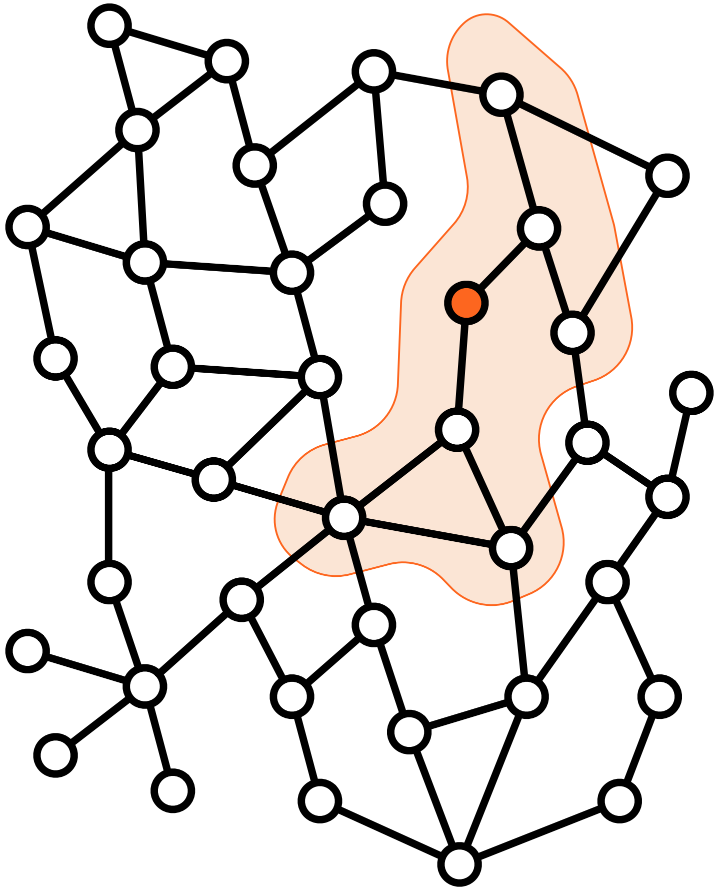


- Initial knowledge:
  - local input, number of neighbors
- Communication round:
  - **send** message to each neighbor
  - **receive** message from each neighbor
  - **update** state
  - possibly: ***announce local output and stop***

LOCAL model

Equivalent:

- "*running time*"
- number of synchronous **communication rounds**
- ***how far*** do we need to look in the graph



Fast algorithm ↔ highly "localized" solution

Scheduling & network algorithms

What are relevant and interesting scheduling problems to study here?

1. What kind of scheduling ***is needed*** in networks?
2. What kind of scheduling problems ***can be solved*** (efficiently) in networks?

Scheduling & network algorithms

Not necessarily intersection:

1. We can ask ***what if*** we could solve this
   - e.g. what is the power of **scheduling oracles**
2. We can ***explore limits*** of solvability, without specific applications in mind
   - cf. "**canonical hard problems**" in centralized setting

Scheduling & network algorithms

- Interesting scheduling problems are usually ***graph problems***
  - nodes need to take actions, and scheduling constraints can be represented as (labelled) edges
- Prime example: **(fractional) graph coloring**

Fractional graph coloring

- **Constraint graph *H***
  - edge $\{u, v\}$ = nodes u and v cannot be active simultaneously
- Each node has **1 unit of work** to do
  - can be generalized to weighted graphs
- Schedule activities, **minimize makespan**

Fractional graph coloring

- **Constraint graph *H***
  - edge $\{u, v\}$ = nodes u and v cannot be active simultaneously
- Set of active nodes = **independent set**
  - ***global view:*** list of independent sets + time spans
  - ***local view:*** each node knows its own schedule

[Fractional] graph coloring

- **Fractional graph coloring:**
  1 unit of work can be divided arbitrarily
  - i.e. ***with preemption***
- **Graph coloring:** atomic jobs
  - i.e. ***without preemption***
  - w.l.o.g. jobs may start at times 0, 1, … only
  - "color" of a node = time slot

[Fractional] graph coloring

- **Fractional graph coloring:**
  - ***"external" applications***: e.g. scheduling radio transmissions in a non-interfering manner
- **Graph coloring:**
  - ***"internal" applications***: coordinating activities of nodes in a distributed algorithm
  - e.g.: constructing a maximal independent set

Graph coloring & network algorithms

- **Constraint graph *H*:**
  - edge {u, v}: nodes u and v interfere with each other
- **Network graph *G*:**
  - edge {u, v}: nodes u and v can talk to each other
- Interesting case: ***H* = *G***

Graph coloring & network algorithms

- **Constraint graph H = network graph G**
  - typical: **conflict → nodes close to each other**
  - worst case: conflict ↔ nodes close to each other
  - often not literally true if G = physical network
  - but we can interpret H as a **virtual network**, and efficiently **simulate** any communication in H by message-passing in G (with constant overhead)

Graph coloring & network algorithms

- Toy example: ***G* = cycle, 3 colors**
  - you are a node in the middle of a long cycle
  - you can talk to your neighbors
  - eventually you need to announce
    "**I am now done, I pick color *x* and stop**"
  - how many (parallel) ***rounds of communication***
    are needed?

Graph coloring & network algorithms

- Toy example: **G = cycle, 3 colors**
- Simple randomized algorithm

Graph coloring & network algorithms

- Toy example: **G = cycle, 3 colors**
- Simple randomized algorithm:
  - everybody picks a random color from {1, 2, 3}
  - check with your neighbors, stop if good for you
  - $O(\log n)$ rounds until everybody stops w.h.p.

Graph coloring & network algorithms

- Toy example: **G = cycle, 3 colors**
- Simple randomized algorithm: $O(\log n)$
- No deterministic algorithm: why?

Graph coloring & network algorithms

- Toy example: **G = cycle, 3 colors**
- Simple randomized algorithm: $O(\log n)$
- No deterministic algorithm:

  everyone has the **same initial state**
  → everyone sends the **same messages**
  → everyone receives the **same messages**
  → everyone has the **same new state**

Graph coloring & network algorithms

- Toy example: ***G*** **= cycle, 3 colors**
- Simple randomized algorithm: $O(\log n)$
- No deterministic algorithm — unless some **symmetry-breaking** information is provided
- Standard assumption: **unique identifiers**

Graph coloring & network algorithms

- Toy example: ***G* = cycle, 3 colors**
- Assume each node has a **unique identifier** from {1, 2, …, poly(n)}
  - e.g. IP address, MAC address, …
  - we will assume a **worst-case assignment**
  - note: random identifiers are unique w.h.p.

Graph coloring & network algorithms

- Toy example: **G = cycle, 3 colors**
- We have now a **color reduction problem**:
  - input: coloring with **poly(n) colors** (unique IDs)
  - output: coloring with **3 colors**

Graph coloring & network algorithms

- Toy example: ***G* = cycle, 3 colors**
- We have now a **color reduction problem**:
  - input: coloring with ***k* colors**
  - output: coloring with ***c* colors**

Graph coloring & network algorithms

- Toy example: **G = cycle, 3 colors**
- We can **iterate** color reduction steps:
  - 1 round: **10^{100}** colors → **12** colors
  - 1 round: **12** colors → **4** colors
  - 1 round: **4** colors → **3** colors
- Approx. **½ log* k** rounds: k → 3 colors

Graph coloring & network algorithms

- **G = cycle, 3 colors**
  - distributed complexity $\Theta(\log^* n)$ rounds
  - upper bound: Cole & Vishkin (1986)
  - lower bound: Linial (1992)
- **G = cycle, 2 colors**
  - even if we promise that the cycle is even, we will need $\Theta(n)$ rounds

Graph coloring & network algorithms

- **Graph coloring in cycles:**
  - 2 colors: $\Theta(n)$ rounds
  - 3 colors: $\Theta(\log^* n)$ rounds
  - 4 colors: $\Theta(\log^* n)$ rounds …
- **Fractional graph coloring in cycles:**
  - $3+\varepsilon$ time units: $O(1)$ rounds [not practical]

Graph coloring & network algorithms

- **Graph coloring in 2D grids:**
  - 3 colors: $\Theta(n)$ rounds
  - 4 colors: $\Theta(\log^* n)$ rounds [surprise!]
  - 5 colors: $\Theta(\log^* n)$ rounds …
- **Fractional graph coloring in 2D grids:**
  - 5+ε time units: $O(1)$ rounds [not practical]

Graph coloring & network algorithms

- **Graph coloring, max degree ≤ Δ:**
  - **Δ** colors: **polylog(*n*)** rounds [assuming Δ ≥ 3]
  - **Δ+1** colors: **Θ(log* *n*)** rounds [assuming Δ = *O*(1)]
- **Fractional graph coloring:**
  - **Δ+1+ε** time units: ***O*(1)** rounds [not practical]

Examples of scheduling problems

Scheduling & network algorithms

- Graph coloring
  - ***non-preemptive scheduling***
  - vertex coloring with $\Delta + 1$ colors, Δ colors
  - edge coloring with $2\Delta - 1$ colors, $(1 + \varepsilon)\Delta$ colors
  - coloring trees with 3 colors
  - “defective” and “weak” colorings
  - large cuts …

Scheduling & network algorithms

- Graph coloring
  - note that we do not try to find e.g. optimal colorings
  - we are usually happy with a ***suboptimal coloring*** that ***can be found quickly***
  - typically coloring is used as a subroutine
  - overall running time = ***f***(time to find coloring, number of colors)

Scheduling & network algorithms

- Graph coloring
- Fractional coloring
  - *preemptive scheduling*
  - finding a schedule of length **$\Delta+1+\varepsilon$**

Scheduling & network algorithms

- Graph coloring
- Fractional coloring
- List coloring
  - ***scheduling with node-specific time constraints***
  - coloring with lists of length **Δ+1**

Scheduling & network algorithms

- [Fractional] domatic partition
  - schedule = list of dominating sets + time spans
  - nodes can also “cover” their neighbors
  - each node has to be “covered” all the time
  - each node can be active for only 1 time unit in total
  - e.g. battery-powered sensors

Scheduling & network algorithms

- [Fractional] domatic partition
  - schedule = list of **dominating sets** + time spans
  - **minimum degree**: δ
  - optimal schedule length $\leq$ $\delta + 1$
  - can find solutions of length $\dfrac{\delta + 1}{O(\log \delta + 1)}$

Scheduling & network algorithms

- Reconfiguration problems
  - **input:** “configurations” *A* and *B*
  - **output:** schedule for “smoothly” switching from *A* to *B* without interfering with the network operation
  - example: recoloring problems

Recoloring problems

- **Input:** ***k*-colorings** *A* and *B*
- **Output:** schedule that tells how to turn coloring *A* into coloring *B*
  - at each time step, only non-adjacent nodes can change their colors
  - each intermediate step has to be a ***k*-coloring**

Recoloring problems

- **Input:** ***k*-colorings** *A* and *B*
- **Output:** schedule that tells how to turn coloring *A* into coloring *B*
- Typically hard, global problems
  - relax the constraints slightly…

Recoloring problems

- **Input:** ***k*-colorings** *A* and *B*
- **Output:** schedule that tells how to turn coloring *A* into coloring *B*
  - at each time step, only non-adjacent nodes can change their colors
  - ***c* extra colors**
  - each intermediate step has to be a **(*k*+*c*)-coloring**

Recoloring problems

- **Input:** ***k*-colorings** A and B
- **Output:** schedule that tells how to turn coloring A into coloring B with c extra colors
- How ***fast*** can we do it (number of rounds)?
- What is the ***length of the schedule***?

Recoloring problems: trees

Input colors	Extra colors	Schedule length	Time (rounds)
2	0	—	
2	1	$O(1)$	$\Theta(n)$
3	0	$\Theta(n)$	$\Theta(n)$
3	1	$O(1)$	$O(\log n)$
3	2	$O(1)$	0
4	0	$\Theta(\log n)$	$\Theta(\log n)$

Examples of some recent work

Introducing a little bit of heavy machinery...

Two stories of how to find the same result, without resorting to actual thinking

Some basic definitions needed first

LCL problems

- Assumption throughout this part:
  - **bounded-degree graphs** ($\Delta = O(1)$)
- ***LCL = locally checkable labeling:***
  - $O(1)$ input labels, $O(1)$ output labels
  - feasibility checkable locally: solution is globally good if it looks good in all $O(1)$-radius neighborhoods
  - Naor & Stockmeyer (1995)

LCL problems

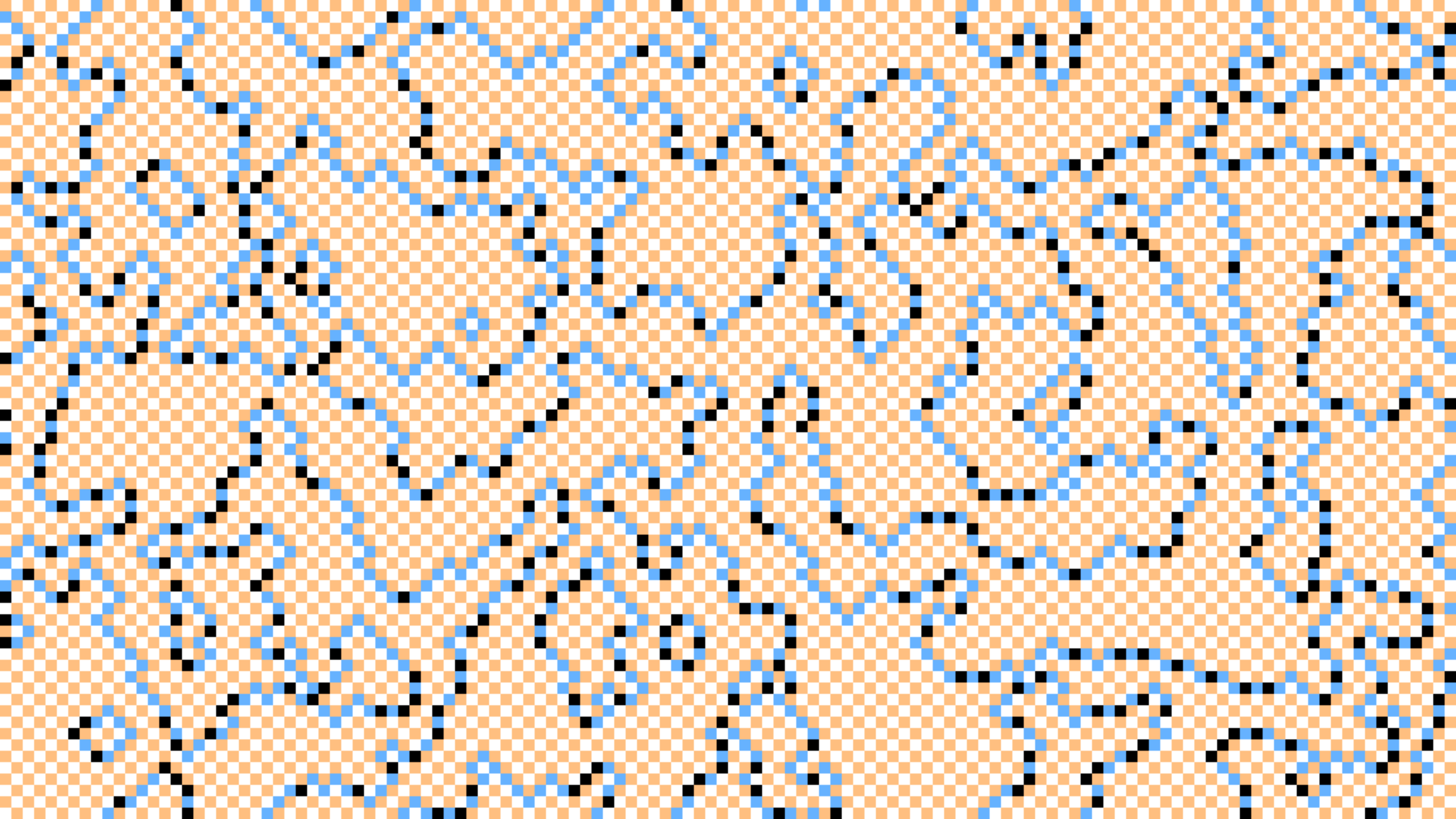
- Examples of ***LCL problems***:
  - graph coloring with 5 colors
  - recoloring in at most 100 steps
- These are ***not*** LCL problems:
  - optimal graph coloring
  - fractional graph coloring
  - recoloring in general

LCL problems

- Examples of ***LCL problems***:
  - graph coloring with 5 colors
  - recoloring in at most 100 steps
- These are ***not*** LCL problems:
  - optimal graph coloring: how to verify locally?
  - fractional graph coloring: unbounded output size
  - recoloring in general: unbounded output size

LCL problems

- Rich theory of LCL problems, lots of recent progress
- Let’s see how it helps with the following problem: ***4-coloring 2D grids***
  - clearly an **LCL problem**
  - highly **nontrivial** problem — try to design an efficient algorithm in the LOCAL model!



Approach 1: gap theorems

- **Theorem:** In ***2D grids***, time complexity of any LCL problem is $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$

(Brandt et al. 2017)

Approach 1: gap theorems

- **Theorem:** In ***2D grids***, time complexity of any LCL problem is $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$
- **Theorem:** In bounded-degree graphs, ***Δ-coloring*** is possible in $\mathrm{polylog}(n)$ time

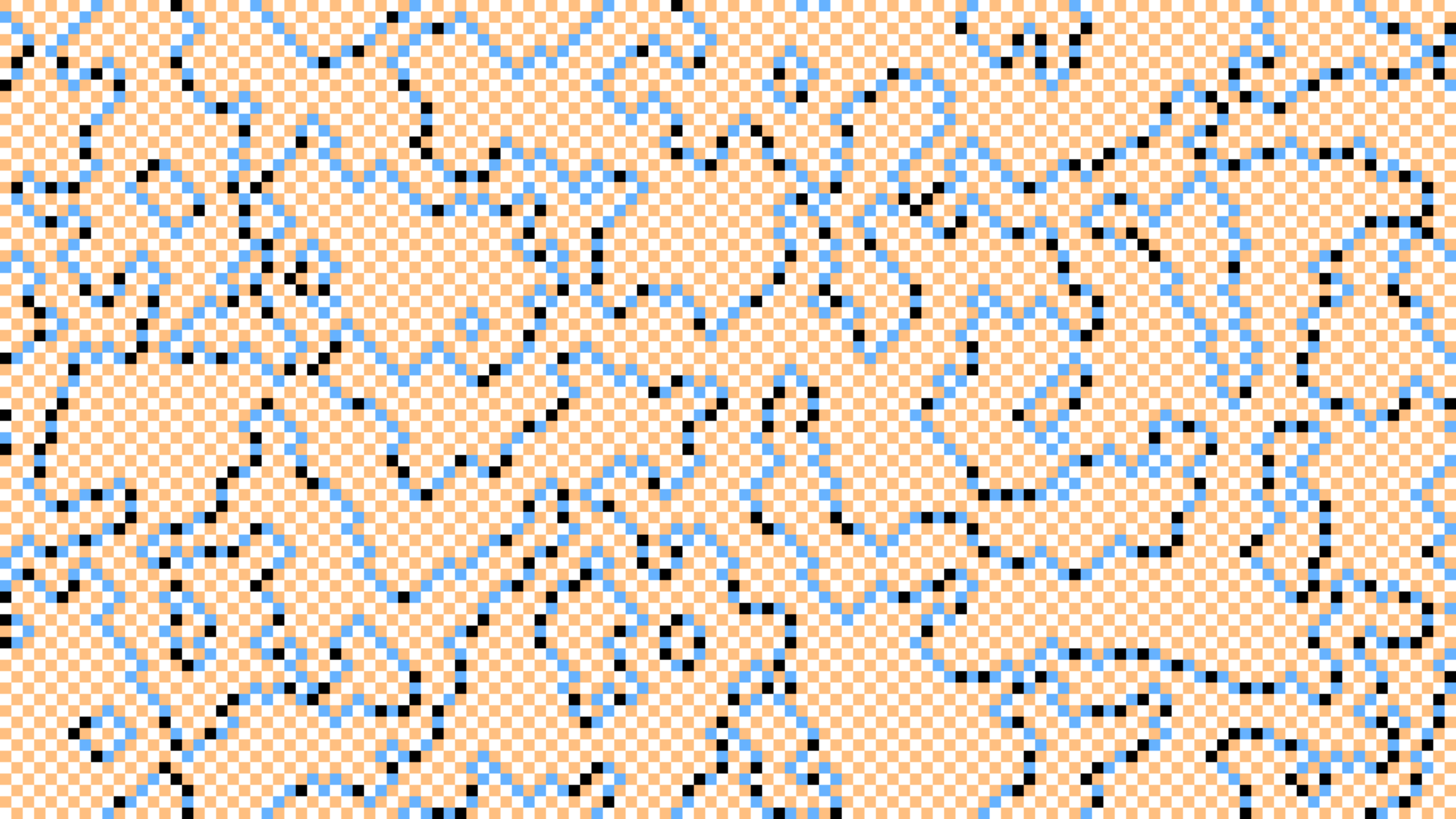
(Panconesi & Srinivasan 1995)

Approach 1: gap theorems

- **Theorem:** In ***2D grids***, time complexity of any LCL problem is $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$
- **Theorem:** In bounded-degree graphs, ***Δ-coloring*** is possible in **polylog(n)** time
- **Corollary:** ***4-coloring in 2D grids*** is possible in $O(\log^* n)$ time

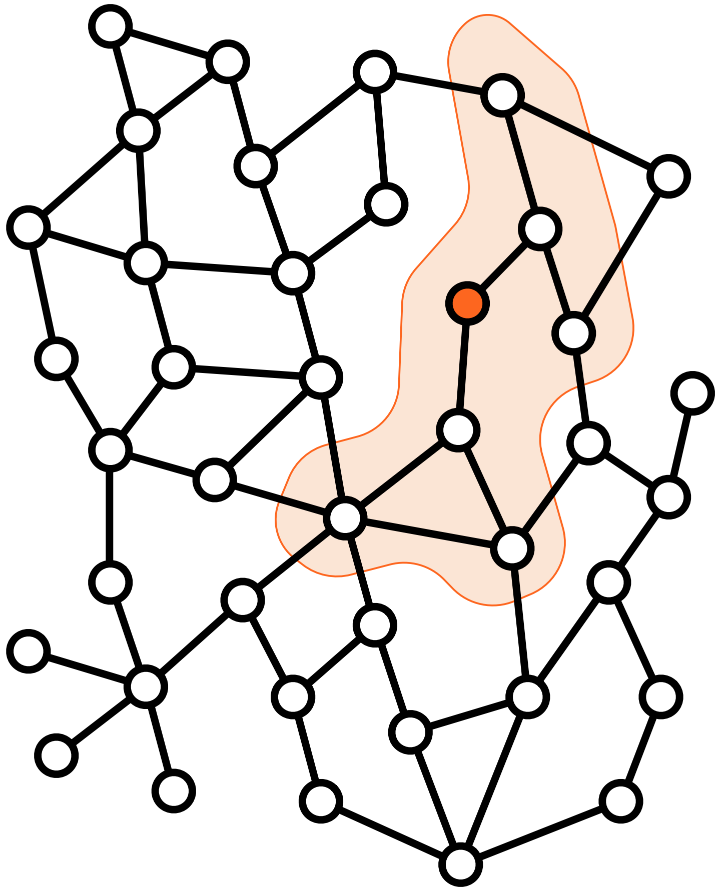
Approach 2: using computers

- In 2D grids, any LCL problem that can be solved in $\Theta(\log^* n)$ time can also be solved with a ***normalized two-part algorithm***:
  1. **symmetry-breaking part:** always the same
  2. **problem-specific part:** finite
- We can ***use computers*** to find the problem-specific part!



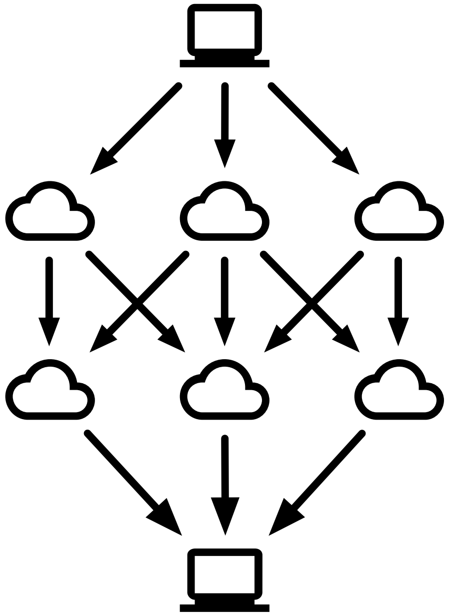
Recap

- Network algorithms
  - **LOCAL model**

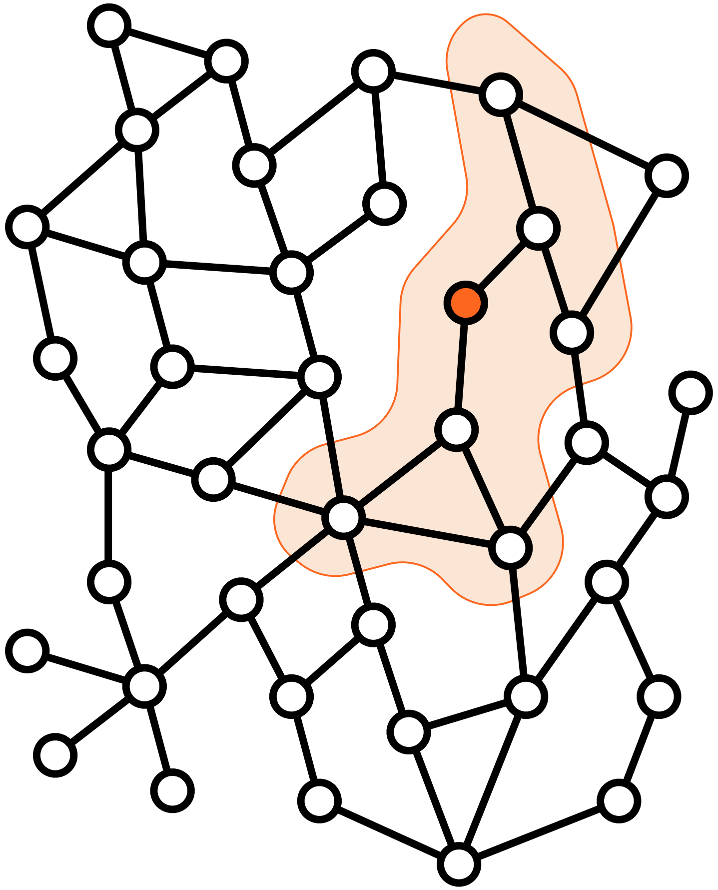


- Key questions about scheduling problems:
  - is this problem **solvable locally**?
  - given a solution, can you **verify** it locally?
  - is it an **LCL** problem?

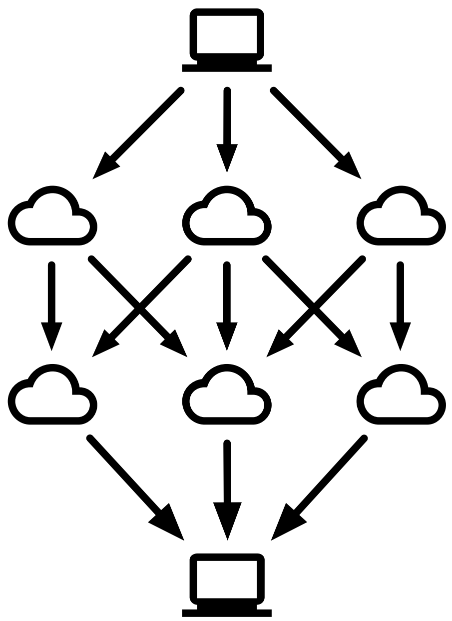
Big data perspective



Network algorithms



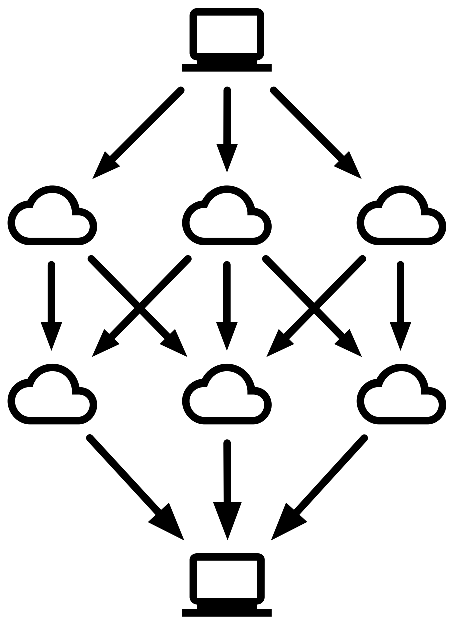
Big data perspective



Network algorithms

- LOCAL
  - unlimited bandwidth
  - unlimited local computation
  - only distance matters

Big data perspective



Network algorithms

- CONGEST
  - just like LOCAL, but with limited **bandwidth**

Big data perspective

- **BSP**
  - p computers
  - each holds $1/p$ of input, needs $1/p$ of output
  - computers can **directly** talk to each other
  - limited **bandwidth**

Network algorithms

- **CONGEST**
  - just like LOCAL, but with limited **bandwidth**

Big data perspective

- BSP
  - no need for concept of "network", everyone can talk to everyone
  - no need to have graph problems
  - any input encoded as a string is fine

Network algorithms

- CONGEST
  - inherently related to networks
  - inherently related to graph problems
  - network structure = input graph

Big data perspective

- BSP

Network algorithms

- CONGEST

What if we studied network algorithms on **complete** graphs?

Big data perspective

- BSP
- Congested clique
  - a special case of BSP: *n* processors, *n* log *n* bandwidth
  - but we don't care about local computation

Network algorithms

- CONGEST
- Congested clique
  - a special case of CONGEST: network = *n*-clique
  - input graph is some subgraph of the clique

Scheduling & congested clique

- Lots of work related to ***graph problems***
  - connectivity, shortest paths, subgraph detection …
- But what is known about ***scheduling and resource allocation***?
  - many efficient algorithms need to split “work” between “workers” in a nontrivial manner
  - is this something we could formalize & study?

Summary

- If someone is studying "***distributed computing***", ask what they mean by it…
- "**Big data algorithms**" and "**network algorithms**" very different concepts
  - focus on computation vs. communication
  - some bridging models exist, though
  - scheduling relevant in all of these models