Non-local probes do not help with graph problems

Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, Jukka Suomela

Models of computing

Are **parallel algorithms** stronger than **distributed algorithms**?

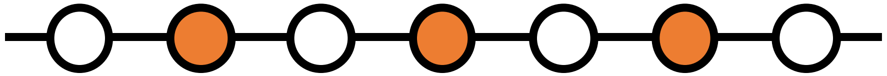
Are there problems that can be solved much faster with parallel algorithms?

Problem setting

Graph problems

Example: ***find an independent set***

- set of non-adjacent nodes

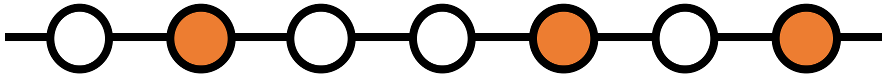


Problem setting

Easy graph problems on **huge** graphs

Example: ***find a maximal independent set***

- set of non-adjacent nodes
- maximal w.r.t. set inclusion



Linear time is trivial — and far too slow

- beyond centralised sequential algorithms…

Centralised & sequential

One processor + one memory

- **input:** stored in memory
- **output:** stored in memory
- e.g. ***RAM model***

Time step:

- read + compute + write

Centralised & sequential

One processor + one memory

- **input:** stored in memory
- **output:** stored in memory
- e.g. ***RAM model***

Trivial lower bound of $\Omega(n)$

- just to read input or to write output

Parallel algorithms

Multiple processors + one **shared memory**

- **input:** stored in memory
- **output:** stored in memory
- e.g. ***CREW PRAM model***

Time step:

- **all processors in parallel:** read + compute + write

Parallel algorithms

Multiple processors + one **shared memory**

- **input:** stored in memory
- **output:** stored in memory
- e.g. ***CREW PRAM model***

n processors, one per node

- O(1) time enough to read all input + write all output

Distributed algorithms

Computer network + **message passing**

- **input:** network topology
- **output:** local output for each computer
- e.g. ***LOCAL model***

Time step:

- **all computers in parallel:** send + receive + compute

Distributed algorithms

Computer network + **message passing**

- **input:** network topology
- **output:** local output for each computer
- e.g. ***LOCAL model***

n processors, one per node

- O(1) time enough to read all input + write all output

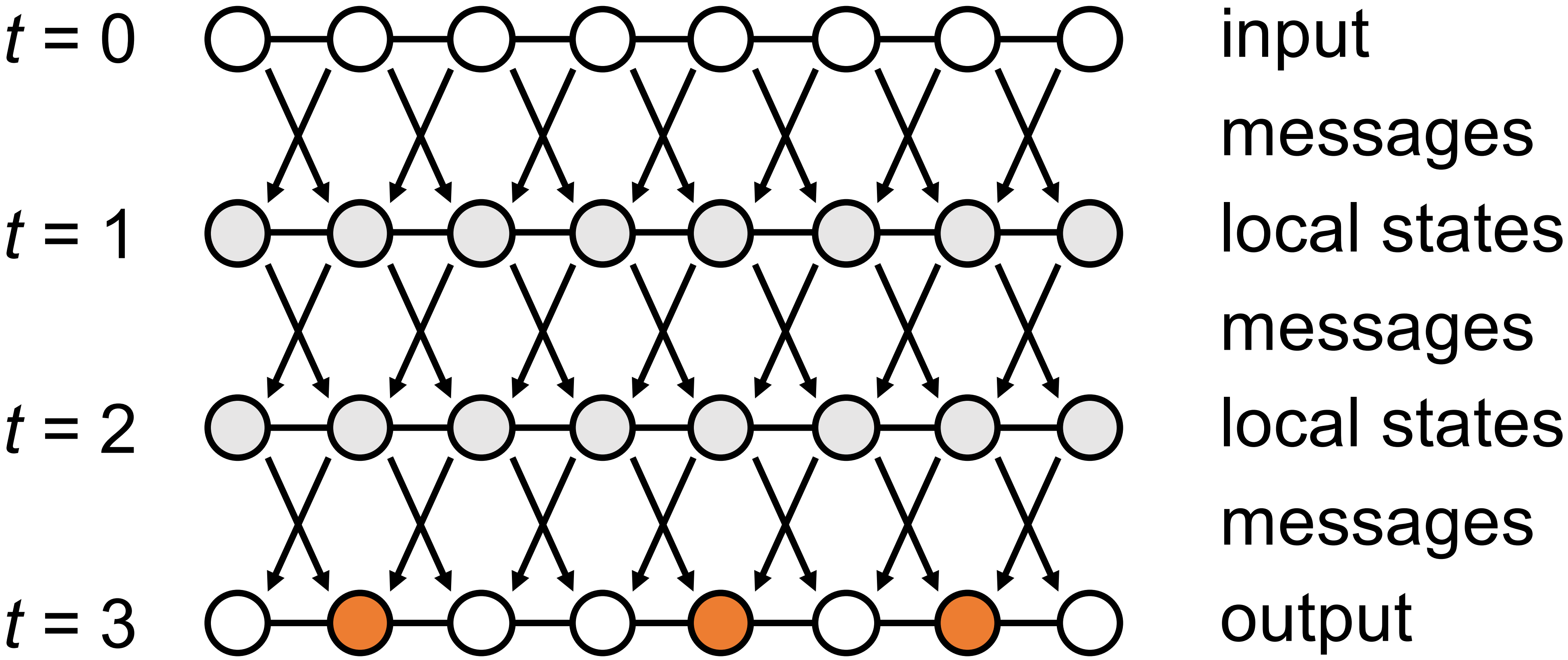
Distributed vs. parallel

Apples vs. oranges?

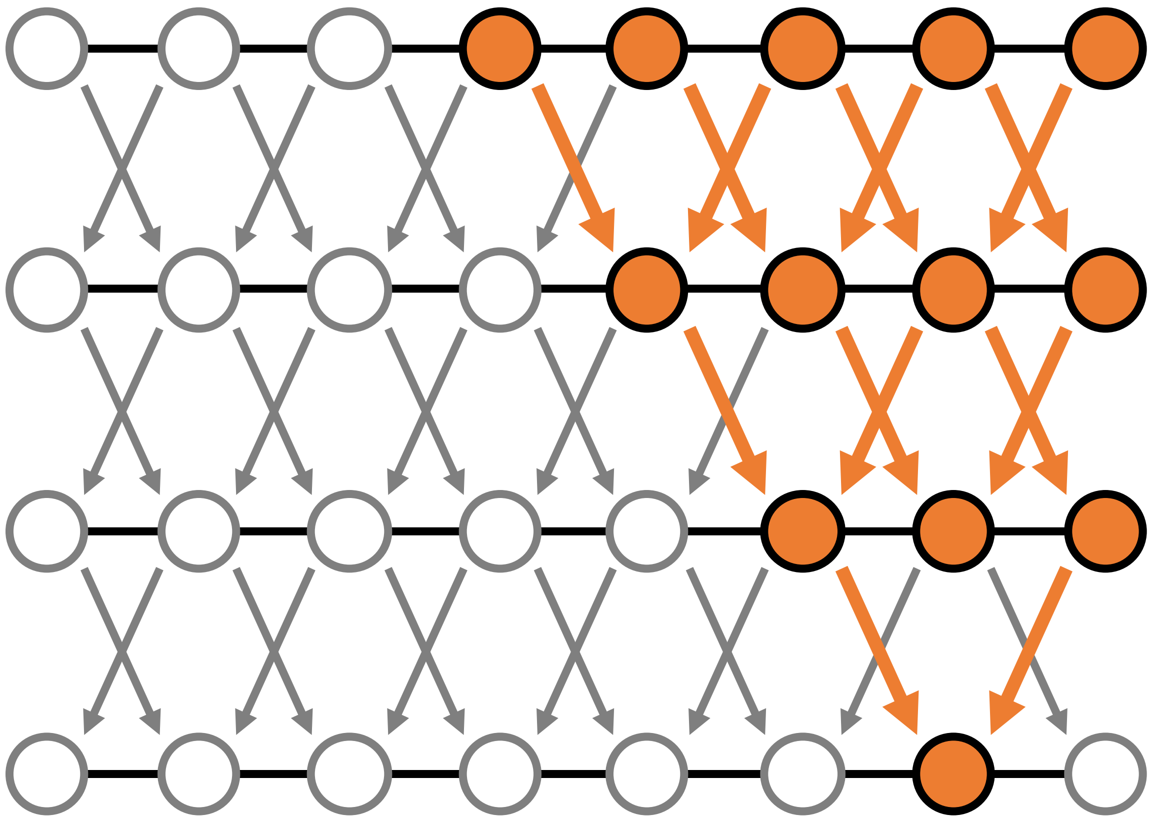
Only one fundamental difference: **locality**

… and it doesn't matter that much, either

Message passing

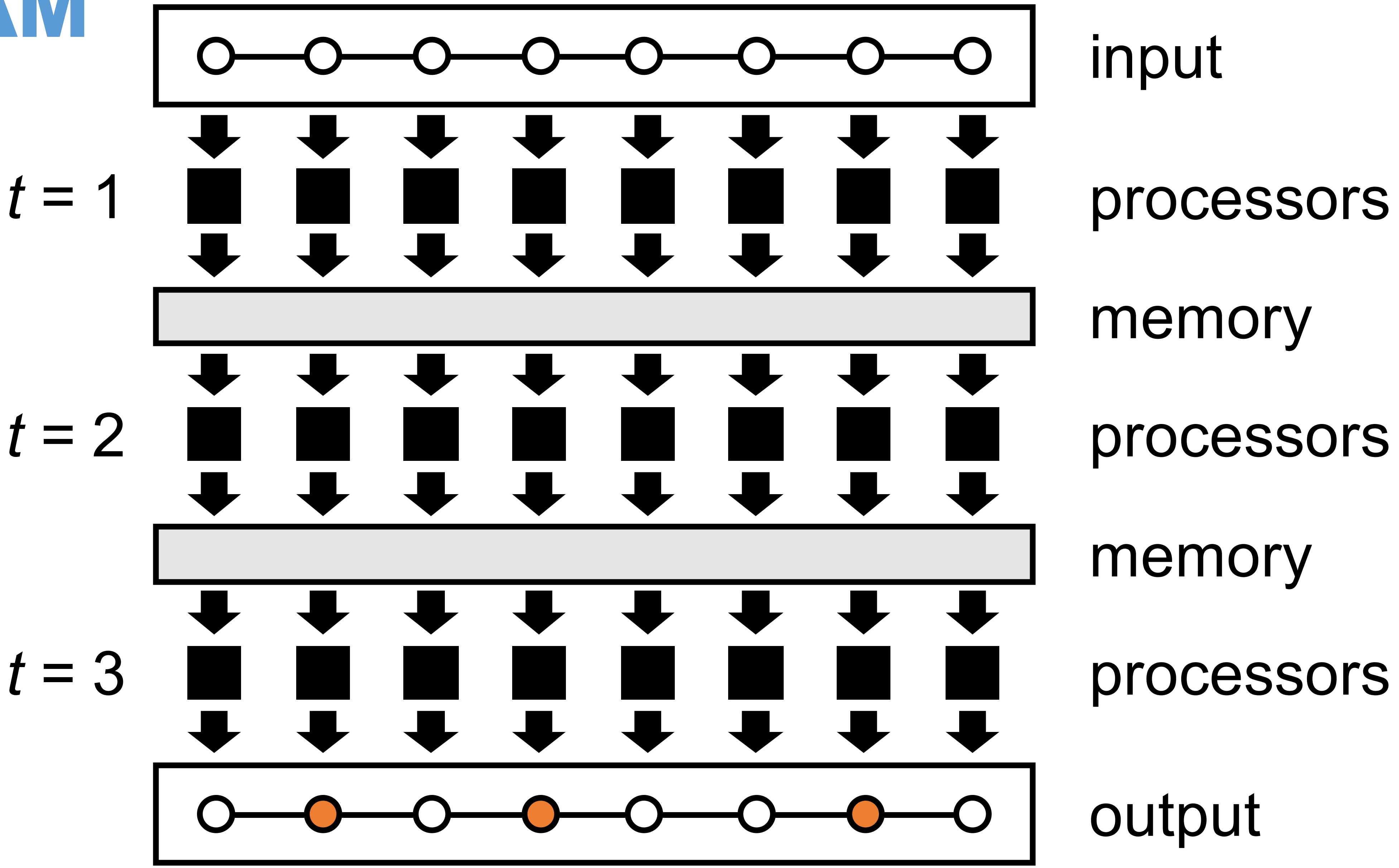


Message passing

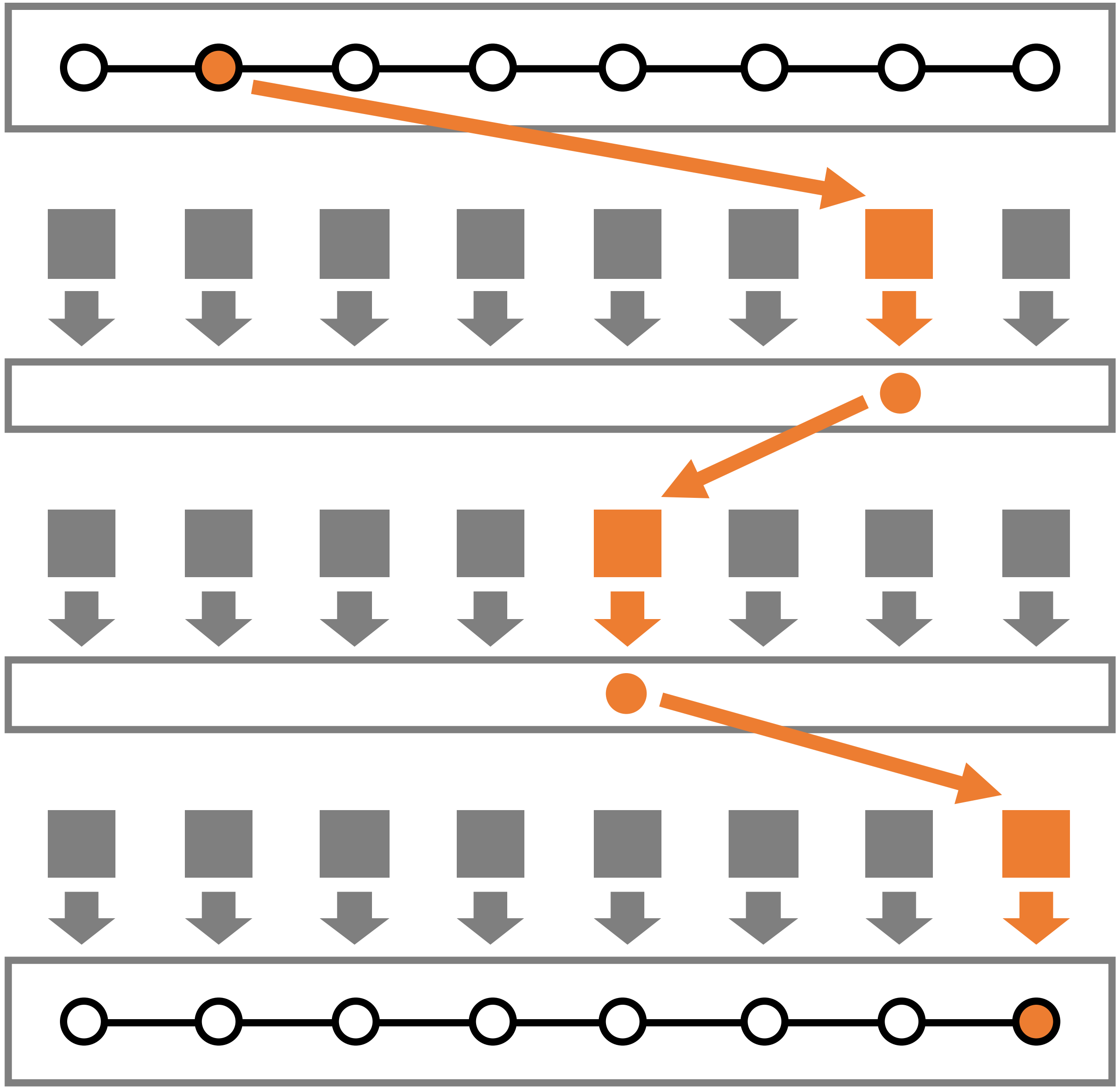


locality

PRAM


input
t = 1
processors
memory
t = 2
processors
memory
t = 3
processors
output

PRAM



no locality

Distributed

- Locality
- Key limitation: **information**
- Well understood

Parallel

- No locality
- Key limitation: **computation**
- Poorly understood

?

Probe–query models

- User makes **queries**:
  "*what is the output of node v?*"

- To answer queries,
  algorithm can **probe** the input:
  "*who are the neighbours of node x?*"

- Time = max number of probes / query

Probe–query models

Simplest special case:

- deterministic
- no preprocessing
- no storage between queries

For a fixed input, defines a fixed output

- independent of e.g. query order

Probe–query models

Parallel decision trees

- depth of decision tree = number of probes

Also known as:

- “stateless deterministic centralised local algorithms”
- “stateless deterministic local computation algorithms”
- ***CentLOCAL***, ***LCA***

Probe–query models

Decision trees ≈ parallel algorithms

- with some caveats, and some overhead…

Probe–query models

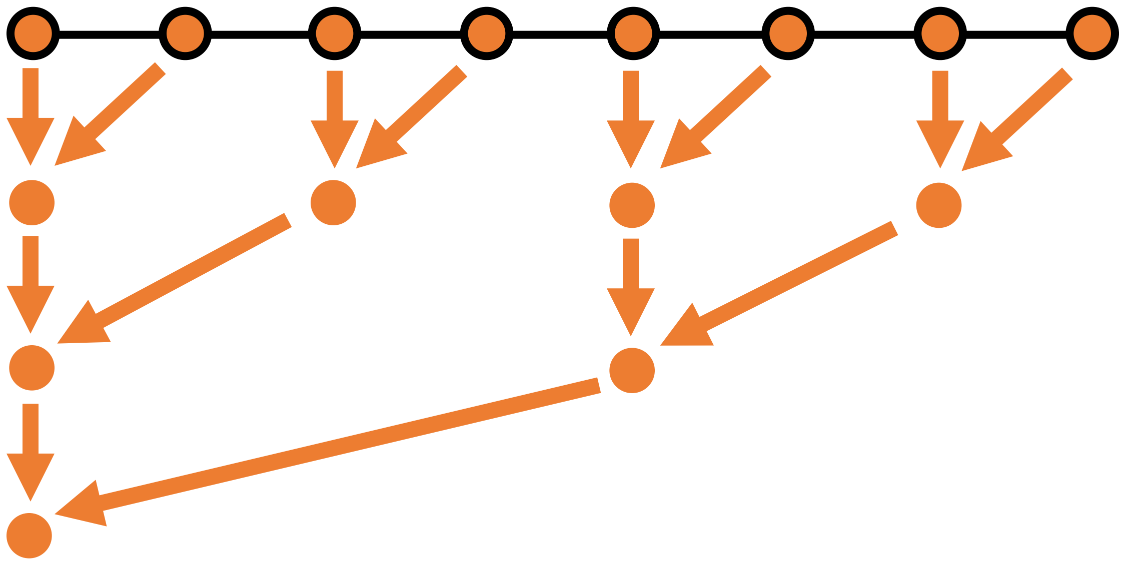
Decision trees → parallel algorithms

- trivial
- let one processor simulate one decision tree

Probe–query models

Parallel algorithms → decision trees

- not always trivial…



after t steps, some outputs might depend on **up to c^t** inputs

Probe–query models

Parallel algorithms → decision trees

- e.g. ***CREW PRAM***: after t steps, each output depends on **at most c^t** inputs
- can be simulated with “only” exponential overhead

- if exponential sounds bad, consider $\Theta(\log \log^* n)$ vs. $\Theta(\log^* n)$

Probe–query models

Decision trees ≈ parallel algorithms

Decision trees ≈ distributed algorithms **???**

Probe–query models

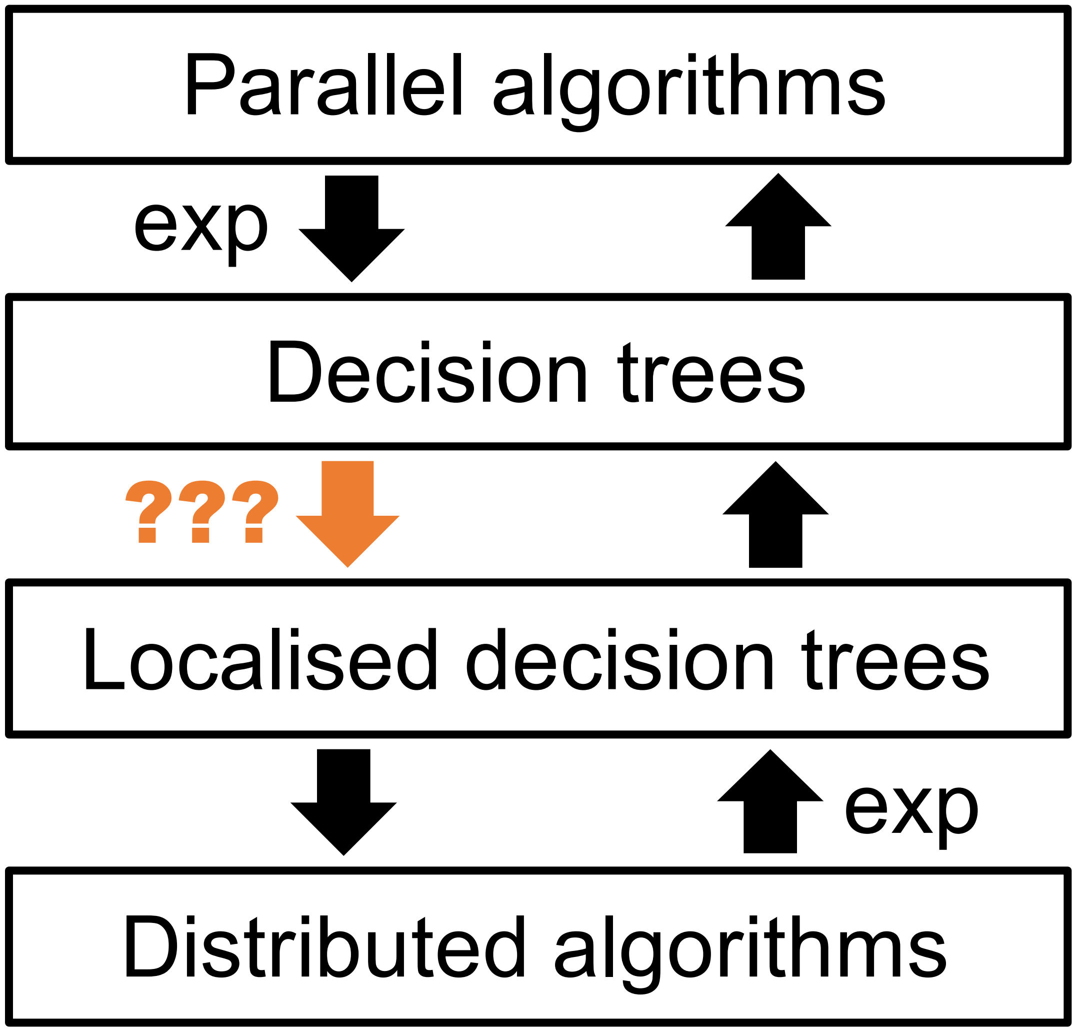
Distributed algorithms → decision trees

- **locality** makes this easy (at least in bounded-degree graphs)
- local output after t rounds depends on at most c^t inputs
- can be simulated with “only” exponential overhead

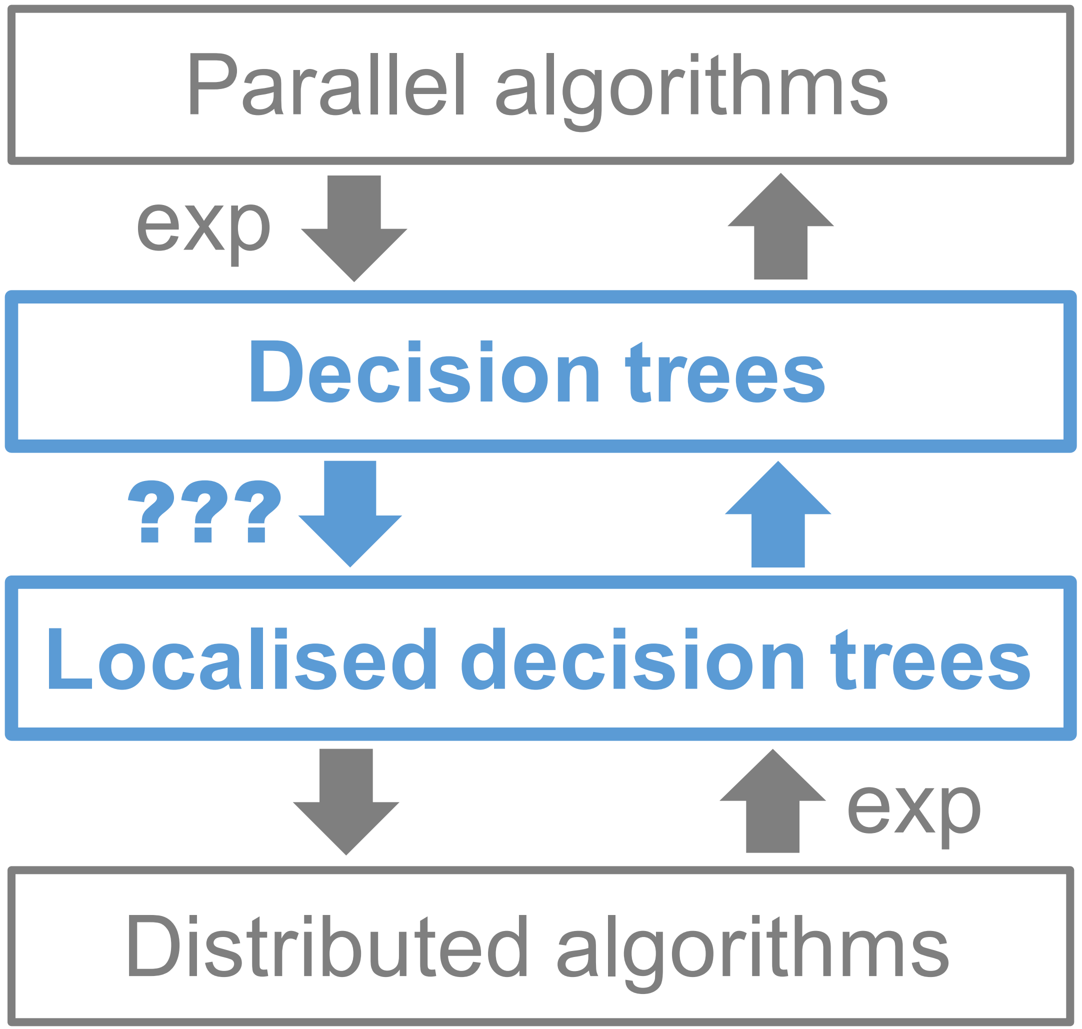
Probe–query models

Decision trees $\rightarrow$ distributed algorithms

- **???**
- distributed algorithms are **localised**
- decision trees are not necessarily localised, can probe everywhere



Parallel algorithms
exp
Decision trees
???
Localised decision trees
exp
Distributed algorithms


Parallel algorithms
exp
Decision trees
???
Localised decision trees
exp
Distributed algorithms

Decision trees and locality

Do we ever benefit from **non-local probes**?

Example:

- I need to find the solution for node v
- makes sense: probe the **neighbours of v**, and then probe their neighbours, etc.
- would it ever make sense to probe node **12345**, or node $12v + 7$?

Decision trees and locality

Do we ever benefit from **non-local probes**?

Yes, for certain (artificial?) problems

- example: binary consensus on graphs

No, for "**nice**" problems

- examples: graph colouring, independent sets, matchings, vertex covers, dominating sets…

Binary consensus

Problem definition:

- input: nodes labelled with 0 and 1
- output: nodes labelled with 0 and 1
- all nodes must produce the same output
- common output equal to at least one input

0000 → 0000 1111 → 1111 0111 → 0000 or 1111

Binary consensus

Separation:

- **$O(1)$** non-localised probes
- **$\Omega(n)$** localised probes

Binary consensus

Trivial with O(1) non-localised probes:

- local output of node *v* =
  local input of node 1

Binary consensus

Requires $\Omega(n)$ localised probes:

00000000000000 $\rightarrow$ **00000000000000**

11111111111111 $\rightarrow$ **11111111111111**

00000001111111 $\rightarrow$ **00000000000000** or
11111111111111

Binary consensus

Requires $\Omega(n)$ localised probes:

00000000000000 $\rightarrow$ **00000000000000**

11111111111111 $\rightarrow$ **11111111111111**

00000001111111 $\rightarrow$ **00000000000000** or
11111111111111

Binary consensus

Requires $\Omega(n)$ localised probes:

00000000000000 $\rightarrow$ 00000000000000

11111111111111 $\rightarrow$ 11111111111111

00000001111111 $\rightarrow$ 00000000000000 or
11111111111111

“Nice” problems

- Defined for bounded-degree graphs
- Invariant under permutation of labels
- Can be solved component-wise

“Nice” problem are localised

Theorem: non-local probes do not help much with “nice” graph problems

If solvable with $t(n) \ll \log^{1/2} n$ probes,
then also solvable if limited to radius-$t(n^{\log n})$
local neighbourhoods

“Nice” problem are localised

Theorem: non-local probes do not help much with “nice” graph problems

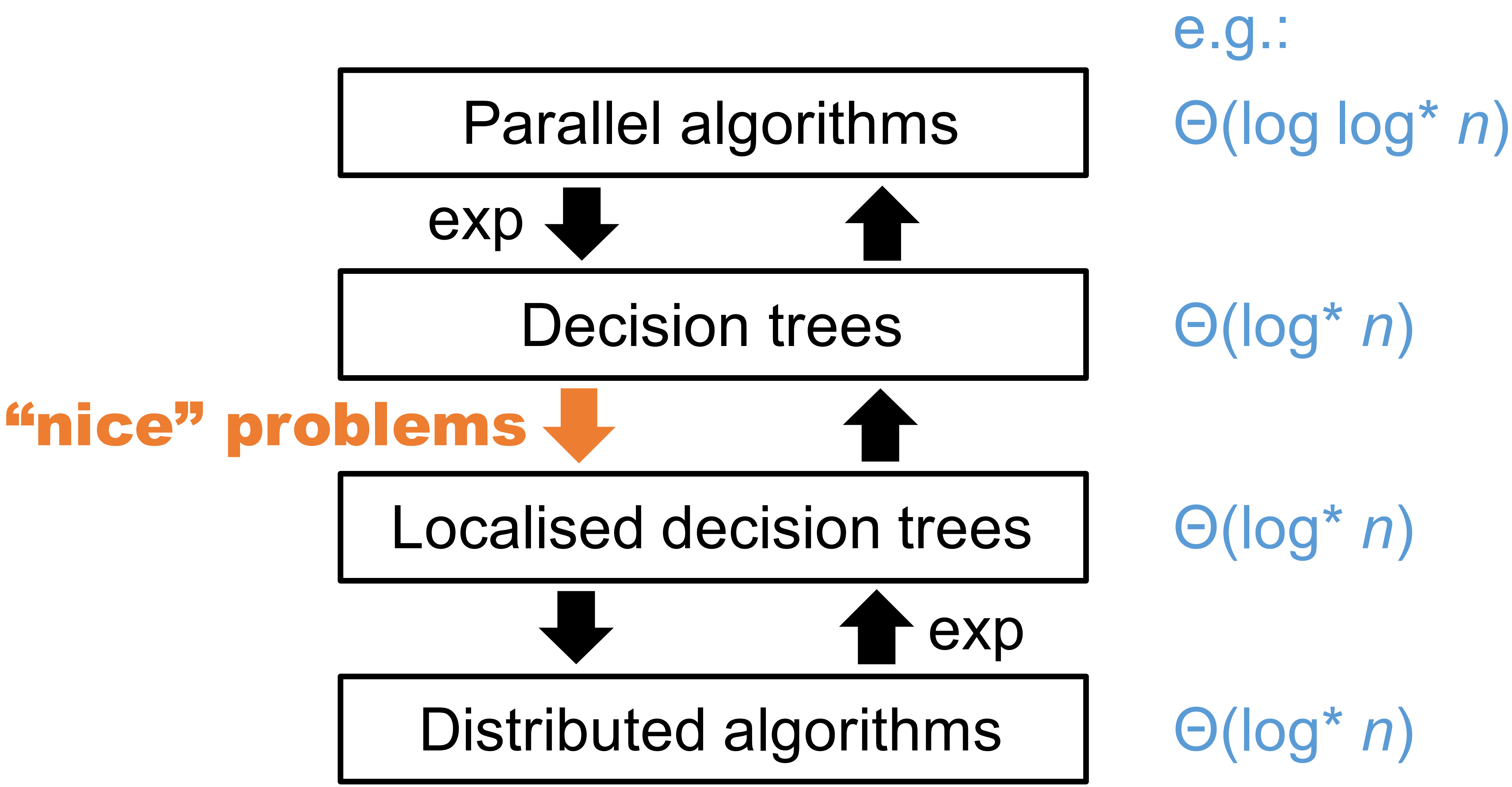
If solvable with $O(\log^* n)$ probes,
then also solvable if limited to radius-$O(\log^* n)$
local neighbourhoods

“Nice” problem are localised

Given: decision tree A for inputs of size N

Construct: local algorithm B for inputs of size $n \ll N$:

- fix a **huge dummy graph** H, **node permutation** π
- B with **input** G: simulate A on input $\pi(G + H)$
- non-local queries: “typically” in H, can answer them
- technical part: there is a fixed π good for any G


e.g.:
Parallel algorithms
Θ(log log* n)
exp
Decision trees
Θ(log* n)
"nice" problems
Localised decision trees
Θ(log* n)
exp
Distributed algorithms
Θ(log* n)