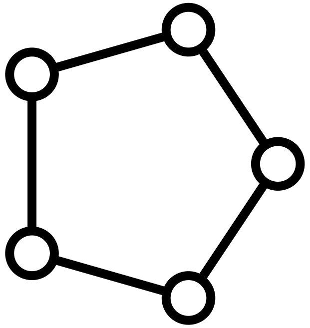
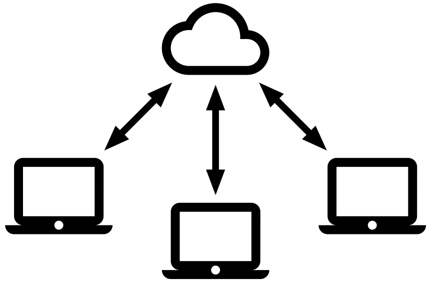


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Jukka Suomela
Aalto University

Low-Degree Graphs, **Sparse Matrices**, and Low-Bandwidth Networks

Based on joint work with

- Keren Censor-Hillel
- Chetan Gupta
- Juho Hirvonen
- Petteri Kaski
- Janne H. Korhonen
- Christoph Lenzen
- Ami Paz
- Jan Studený
- Hossein Vahidi
- and many others…

Matrix multiplication

- Fundamental computational primitive
- Core operation in **modern machine learning**, scientific computation …
- Computationally expensive operation

Matrix multiplication

- So important that there is even special hardware designed to **accelerate** and **parallelize** matrix multiplication!
  - *Nvidia Tensor Core*
  - *Google Tensor Processing Unit*
  - *Intel AMX, Intel XMX …*

Matrix multiplication

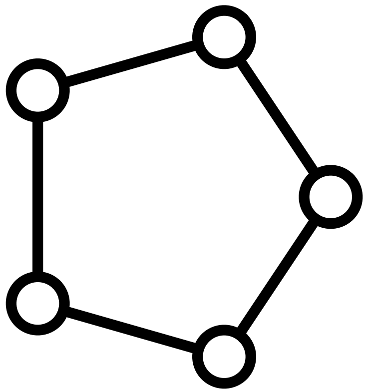
- Interesting "**intermediate**" problem for theory of distributed computing

Problems with simple linear-time centralized algorithms	Problems with nontrivial computation, nontrivial input size	Computationally hard problems
MIS, $(\Delta+1)$-coloring, maximal matching …	matrix multiplication …	SAT, 3-coloring …

Matrix multiplication

- Interesting “**intermediate**” problem for theory of distributed computing
- Direct connections with **graph problems**

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \approx$$



Matrix multiplication

- Interesting "**intermediate**" problem for theory of distributed computing
- Direct connections with **graph problems**
- Natural parameterized family of problems: **sparse** matrix multiplication
  - possible to explore various ***tradeoffs***
  - different ***parameter regimes*** → different tools

Matrix multiplication

- Interesting "**intermediate**" problem for theory of distributed computing
- Direct connections with **graph problems**
- Natural parameterized family of problems: **sparse** matrix multiplication
- Makes sense in virtually **any model** of parallel or distributed computing

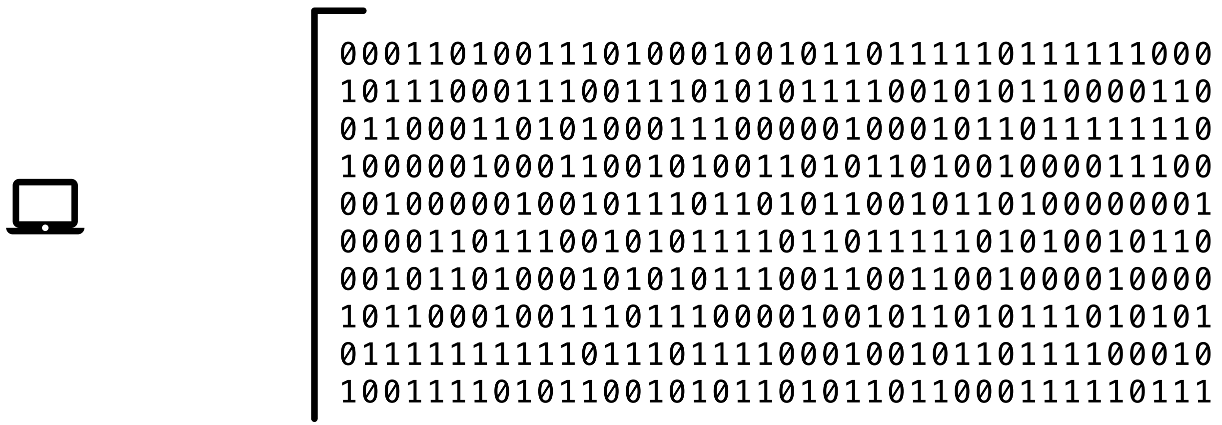
Thought experiment

- How do you multiply matrices with **1,000,000 × 1,000,000 elements**?

```
0001101001110100010010110111110111111000
1011100011100111010101111001010110000110
0110001101010001110000010001011011111110
1000001000110010100110101101001000011100
0010000001001011101101011001011010000001
0000110111001010111101101111101010010110
0010110100010101011100110011001000010000
1011000100111011100001001011010111010101
0111111111101110111100010010110111100010
1001111010110010101101011011000111110111
```

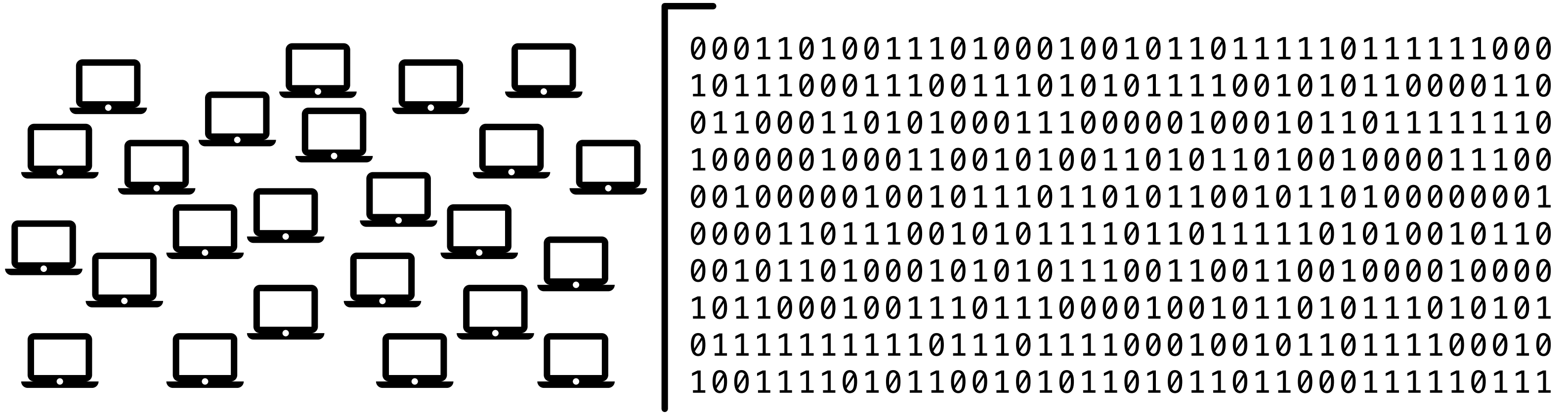
Thought experiment

- How do you multiply matrices with **1,000,000 × 1,000,000 elements**?
  - fits on a hard disk drive
  - naive sequential solution takes ***decades***



Thought experiment

- How do you multiply matrices with **1,000,000 × 1,000,000 elements**?
  - fits on a hard disk drive
  - naive sequential solution takes *decades*
- What if you have **1,000,000 computers**?



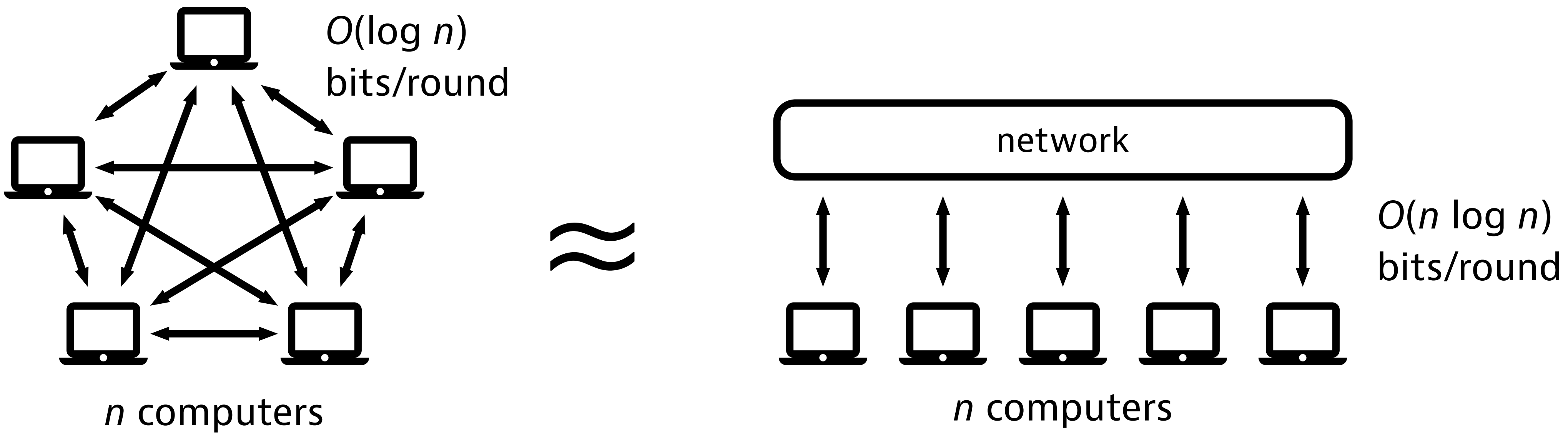
Setting

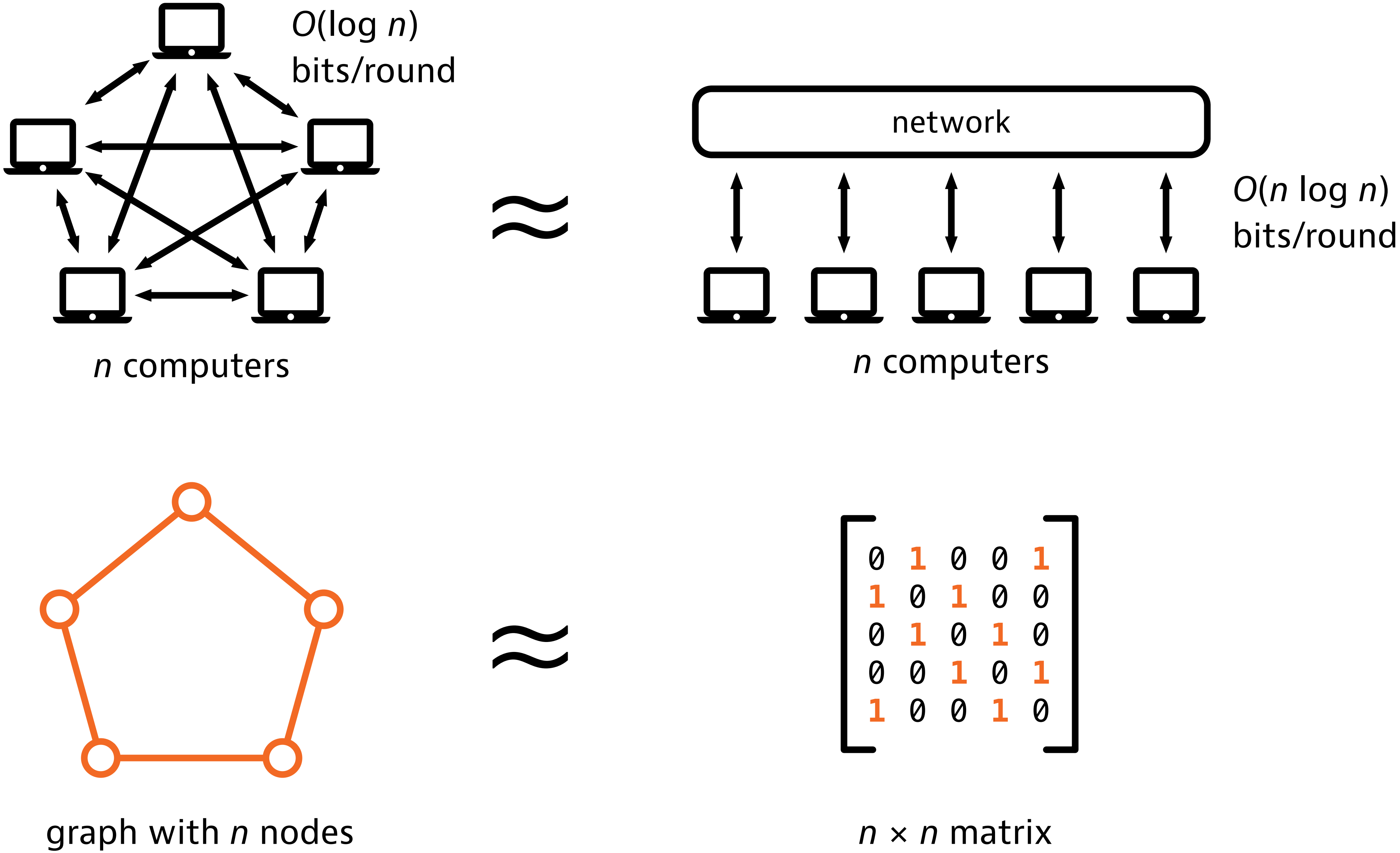
- Convenient choice of parameters:
  multiply $\boldsymbol{n \times n}$ **matrices** using $\boldsymbol{n}$ **computers**

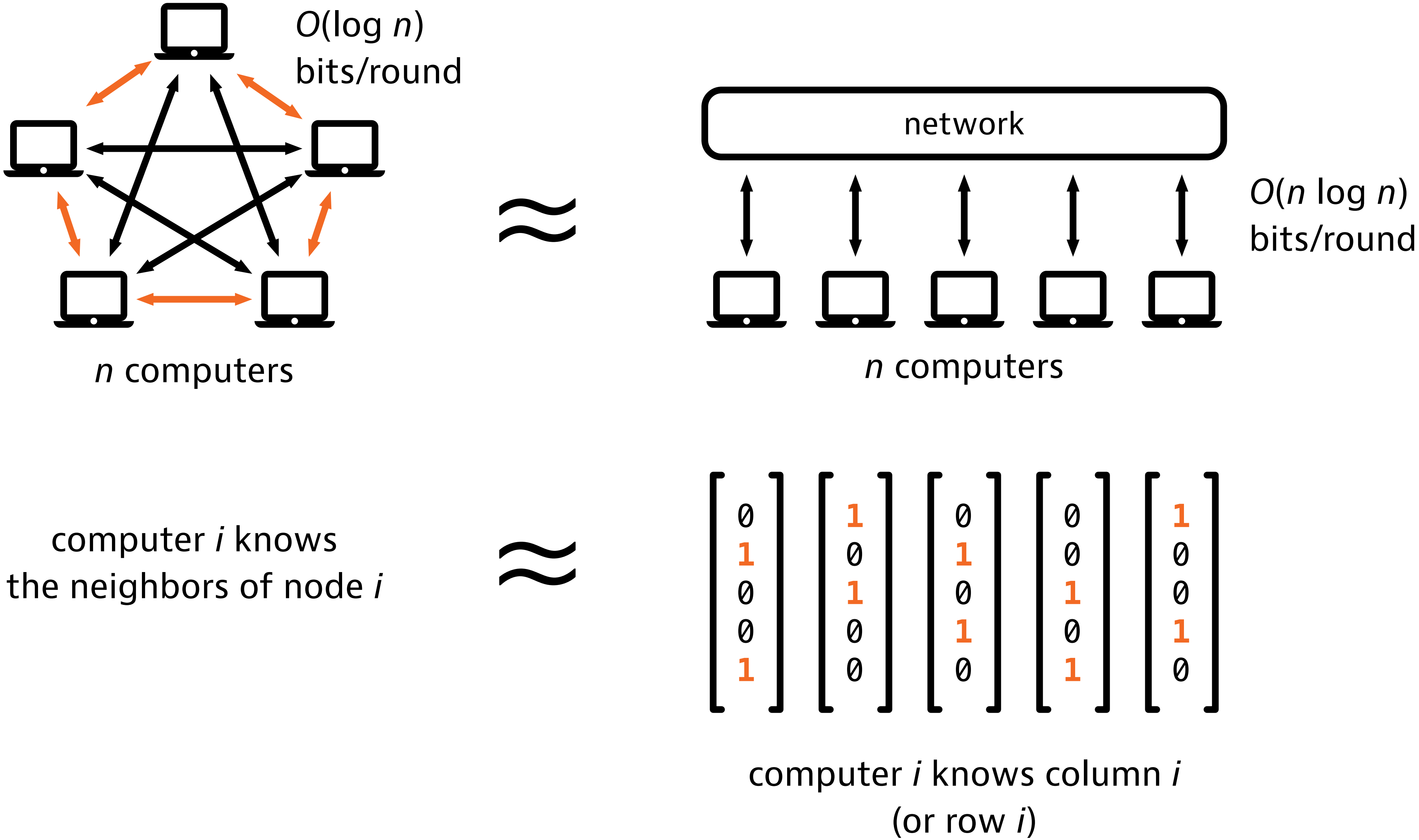
Setting

- Convenient choice of parameters:
  multiply ***n*** **×** ***n*** **matrices** using ***n*** **computers**
- Don’t take “computer” too literally:
  - in practice, one “computer” can be e.g.
    one CPU core + its local cache memory
  - one physical computer can simulate
    many virtual “computers”

Congested Clique


O(log n)
bits/round
n computers
≈
network
O(n log n)
bits/round
n computers


O(log n)
bits/round
n computers
≈
network
O(n log n)
bits/round
n computers
≈
0 1 0 0 1
1 0 1 0 0
0 1 0 1 0
0 0 1 0 1
1 0 0 1 0
graph with n nodes
n × n matrix


O(log n)
bits/round
n computers
≈
network
O(n log n)
bits/round
n computers
computer i knows
the neighbors of node i
≈
0 1 0 0 1
1 0 1 0 0
0 1 0 1 0
0 0 1 0 1
1 0 0 1 0
computer i knows column i
(or row i)

Problem setting

- **Input:** $n \times n$ matrices A and B
  - computer i knows column i of A and column i of B
- **Output:** n x n matrix $X = AB$
  - computer i has to output column i of X
- n computers
- $O(n \log n)$ bits/computer/round

Problem setting

- **Input:** $n \times n$ matrices A and B
  - computer i knows column i of A and column i of B
- **Output:** n x n matrix $X = AB$
  - computer i has to output column i of X
- n computers
- $O(n)$ things/computer/round

It decomposes!

Multiply matrices with **100×100 elements**

$\approx$

Multiply matrices with **10×10 blocks**,
each block contains **10×10 elements**

Key idea

- Look at **centralized** matrix multiplication algorithms
- See what **multiplication operations** they perform, distribute them
- Keep in mind that we can **decompose**

Naive algorithm

- What if our matrices consisted of $\boldsymbol{s} \times \boldsymbol{s}$ elements?
- Naive algorithm would need to calculate $\boldsymbol{s}^3$ products of elements (and do some additions)
- If $\boldsymbol{n} = \boldsymbol{s}^3$, each computer needs to calculate just **one product** of elements — how convenient!

Naive algorithm

- What if our matrices consisted of $\boldsymbol{s} \times \boldsymbol{s}$ ***blocks***?
- Naive algorithm would need to calculate $\boldsymbol{s}^3$ products of ***blocks*** (and do some additions)
- If $\boldsymbol{n} = \boldsymbol{s}^3$, each computer needs to calculate just **one product** of ***blocks*** — how convenient!

Naive algorithm

- What if our matrices consisted of $s \times s$ *blocks*, each with $n/s \times n/s$ elements?
- Naive algorithm would need to calculate s^3 products of *blocks* (and do some additions)
- If $n = s^3$, each computer needs to calculate just **one product** of *blocks* — how convenient!

Naive algorithm

- Split the matrix in $n^{1/3} \times n^{1/3}$ blocks each with $n^{2/3} \times n^{2/3}$ elements
- Route each pair of blocks to a dedicated computers
  - need to send $n^{4/3}$ elements to each computer
  - bandwidth $O(n)$ elements → takes $\boldsymbol{O(n^{1/3})}$ rounds
- Route the results back & aggregate…

Fast algorithm

- What if our matrices consisted of $\boldsymbol{s} \times \boldsymbol{s}$ elements?
- Fast algorithm would need to calculate $\boldsymbol{s}^{2.38}$ products of elements [+ pre/postprocessing]
- If $\boldsymbol{n} = \boldsymbol{s}^{2.38}$, each computer needs to calculate just **one product** of elements

Fast algorithm

- What if our matrices consisted of **$s \times s$** ***blocks***, each with **$n/s \times n/s$** elements?
- Fast algorithm would need to calculate **$s^{2.38}$** products of ***blocks*** [+ pre/postprocessing]
- If **$n = s^{2.38}$**, each computer needs to calculate just **one product** of ***blocks***

Fast algorithm

- What if our matrices consisted of $s \times s$ *blocks*, each with $n/s \times n/s$ elements?
- Fast algorithm would need to calculate $s^{2.38}$ products of *blocks* [+ pre/postprocessing]
- If $n = s^{2.38}$ (that is, $s = n^{0.42}$) each computer needs to calculate just **one product** of *blocks*

Fast algorithm

- Split the matrix in $n^{0.42} \times n^{0.42}$ blocks each with $n^{0.58} \times n^{0.58}$ elements
- Preprocess, then route each pair of blocks to a dedicated computers
  - need to send $n^{1.16}$ elements to each computer
  - bandwidth $O(n)$ elements → takes $\boldsymbol{O(n^{0.16})}$ rounds
- Route the results back & aggregate…

Recap

- Centralized naive matrix multiplication: $O(n^{\mathbf{3}})$
- Congested clique: $O(n^{1-2/\mathbf{3}}) = O(n^{\mathbf{1/3}})$

Recap

- Centralized naive matrix multiplication: $O(n^3)$
- Congested clique: $O(n^{1-2/3}) = O(n^{1/3})$
- Centralized fast matrix multiplication: $O(n^{2.38})$
- Congested clique: $O(n^{1-2/2.38}) = O(n^{0.16})$

Recap

- Centralized naive matrix multiplication: $O(n^{3})$
- Congested clique: $O(n^{1-2/3}) = O(n^{1/3})$
- Centralized fast matrix multiplication: $O(n^{2.38})$
- Congested clique: $O(n^{1-2/2.38}) = O(n^{0.16})$
- Centralized → $O(n^{2})$
- Congested clique → $O(1)$

[PODC 2015]

Recap

- Centralized naive matrix multiplication: $O(n^{3})$
- Congested clique: $O(n^{1-2/3}) = O(n^{1/3})$
- Centralized fast matrix multiplication: $O(n^{2.38})$
- Congested clique: $O(n^{1-2/2.38}) = O(n^{0.16})$
- Centralized → $O(n^{2})$
- Congested clique → $O(1)$

Sparsity?

Sparse matrices

- Let us look at the simplest possible case: **uniformly sparse** input & output
- **Input:** each row and each column contains ≤ ***d*** nonzeros
- **Output:** we only care about ≤ ***d*** elements in each row and column

Sparse matrices

- Example: **triangle detection & counting**
- Let $A = B =$ graph and compute $X = AB$
- X_{ik} = sum of $A_{ij} \cdot B_{jk}$ over all j
  = number of paths of the form i–j–k
- Triangle (i, ?, k) exists if $X_{ik} \neq 0$ and we have edge $\{i, k\}$ in the graph

Sparse matrices

- Example: **triangle detection & counting**
  - ***assume: maximum degree d***
- Let $A = B =$ graph and compute $X = AB$
  - ***A and B are uniformly spares***
- Triangle $(i, ?, k)$ exists if $X_{ik} \neq 0$ and we have edge $\{i, k\}$ in the graph
  - ***we only care about a sparse set of values in X***

Sparse matrices

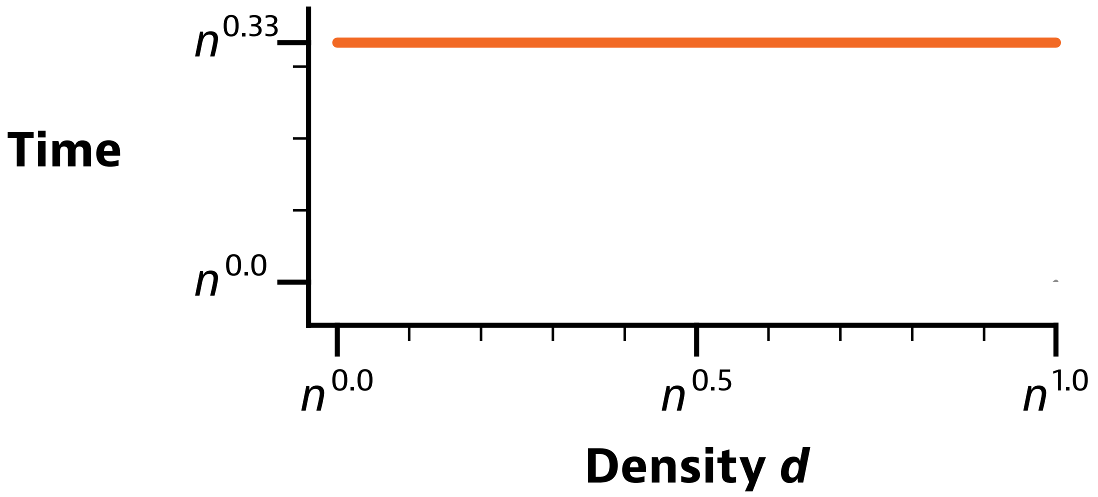
- **Input:** each row and each column contains $\leq$ ***d*** nonzeros
- **Output:** we only care about $\leq$ ***d*** elements in each row and column

Sparse matrices

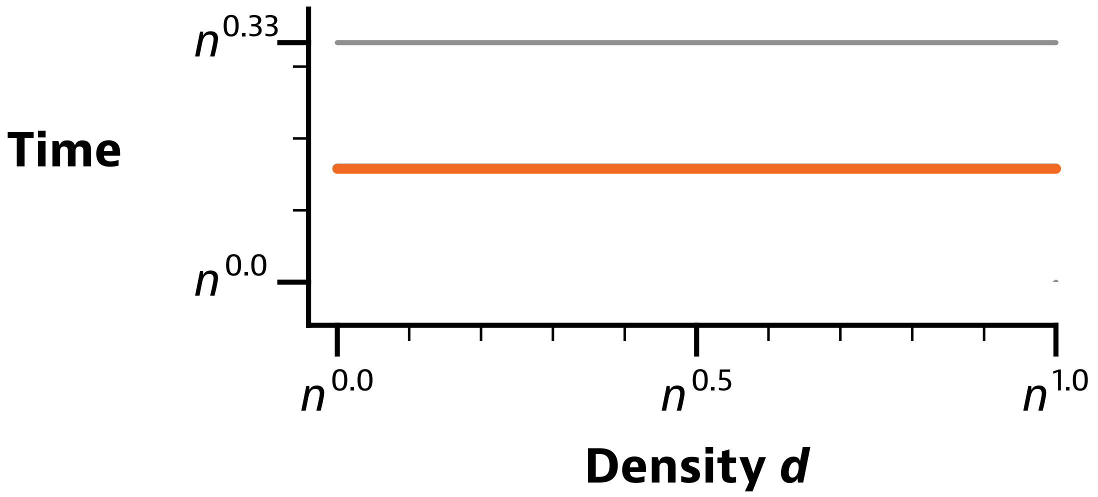
- **Input:** each row and each column contains $\leq d$ nonzeros
- **Output:** we only care about $\leq d$ elements in each row and column
- **Supported model:** matrix structure known
  - locations of (possibly) non-zero inputs
  - locations of output elements we care about

Prior work

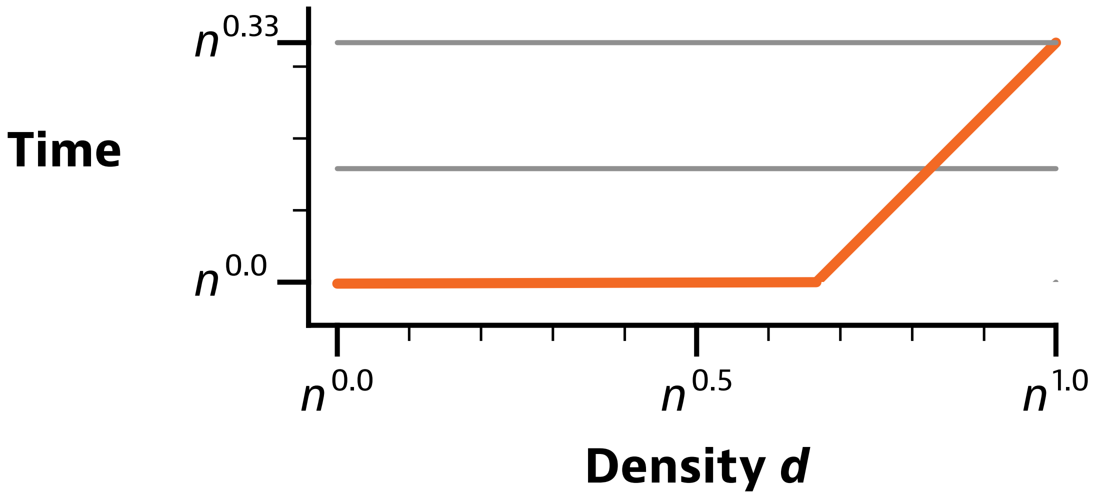
- Trivial dense: $O(n^{1/3})$ rounds
- Censor-Hillel, Dory, Korhonen, Leitersdorf: $O(d/n^{2/3})$ rounds for $d \geq n^{2/3}$
- **$O(1)$ rounds** for $d \leq n^{2/3}$



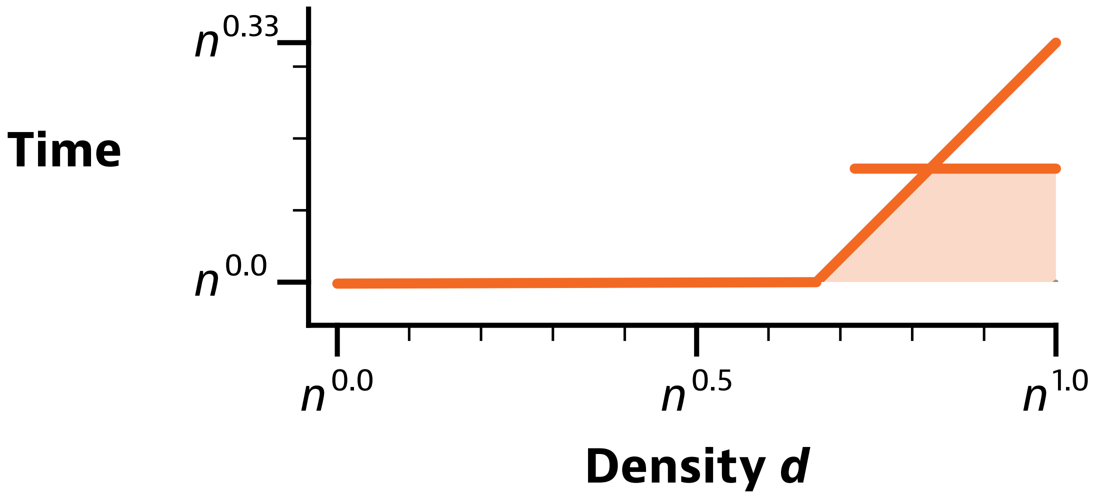
Naive matrix multiplication: $O(n^{0.33})$



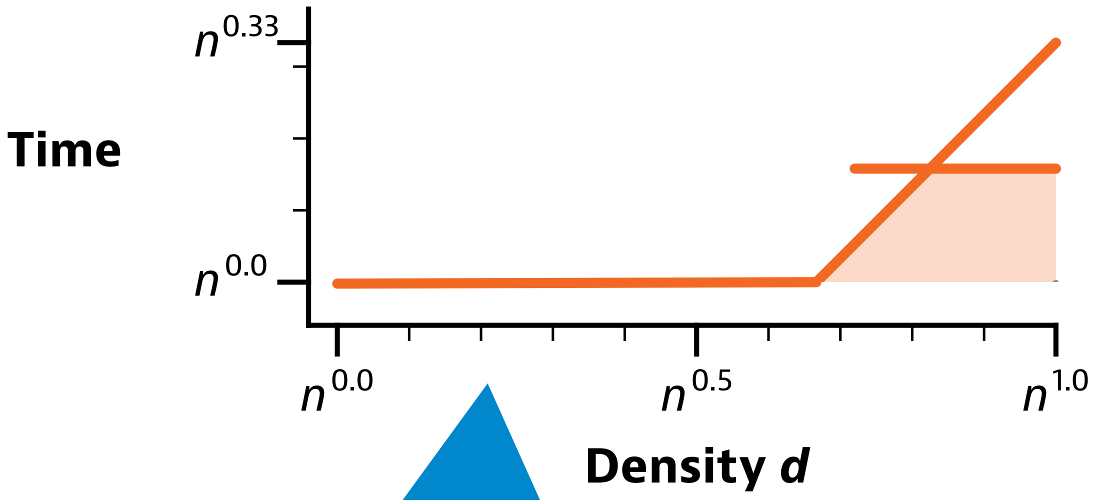
Fast matrix multiplication: $O(n^{0.16})$



Censor-Hillel,
Dory, Korhonen,
Leitersdorf



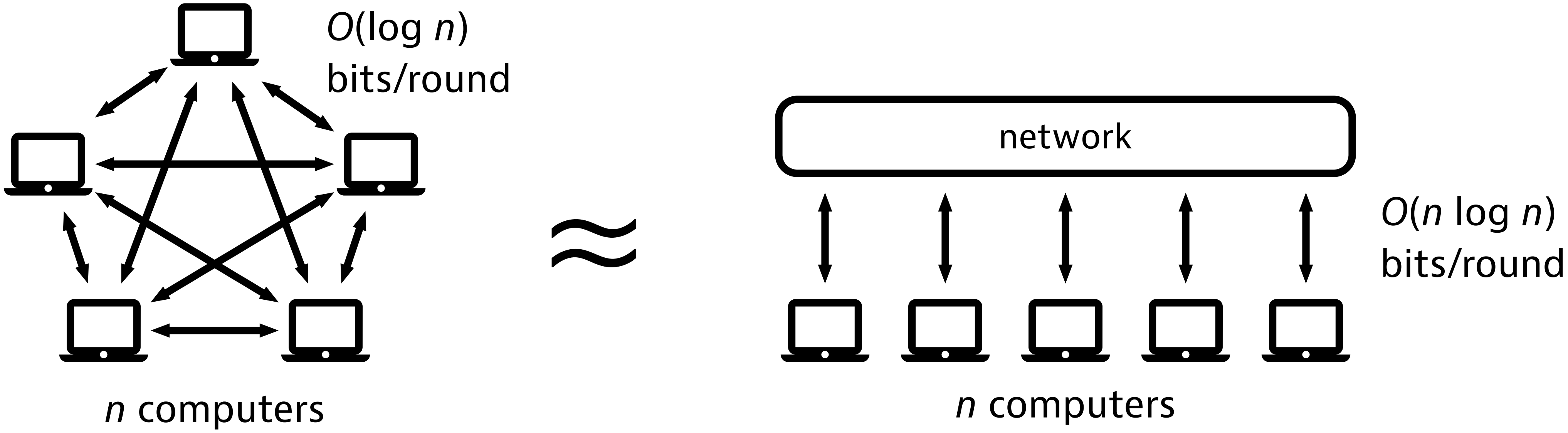
Complexity of sparse matrix multiplication in congested clique



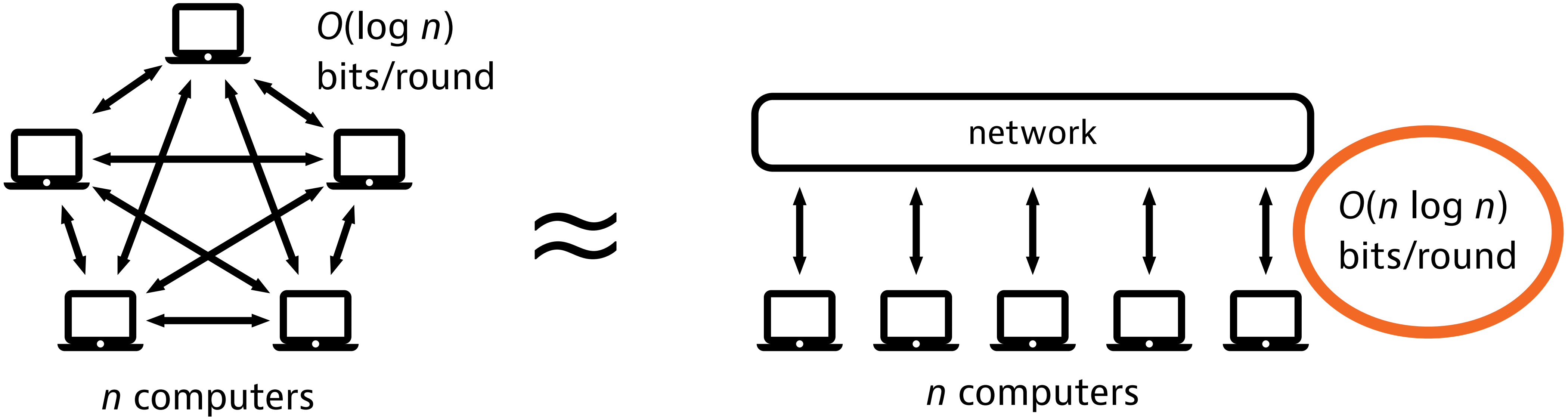
Complexity of sparse matrix multiplication in congested clique

It doesn't matter if the matrix is very sparse or fairly dense???

Wrong model



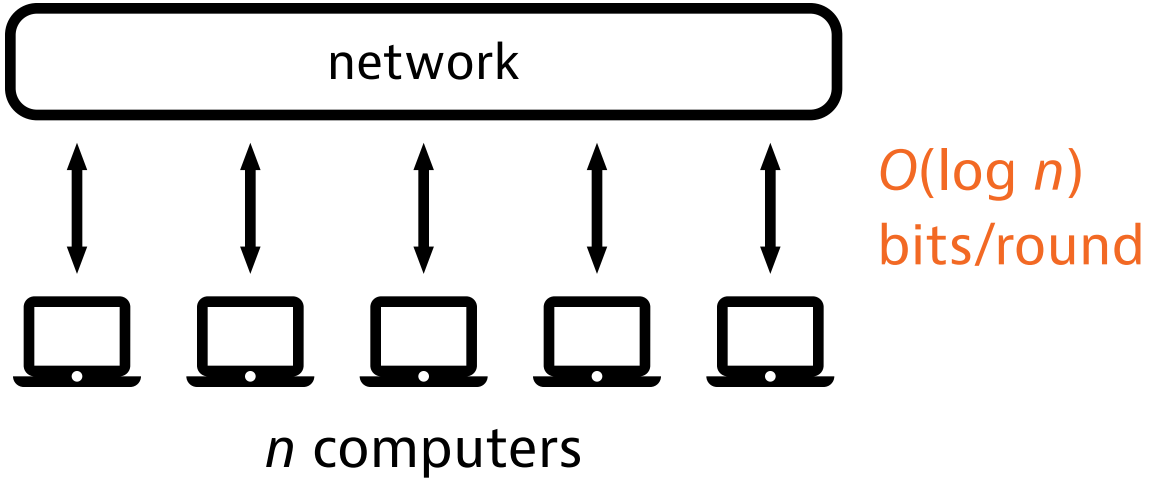
O(log n)
bits/round
n computers
≈
network
O(n log n)
bits/round
n computers



O(log n)
bits/round
n computers
≈
network
O(n log n)
bits/round
n computers

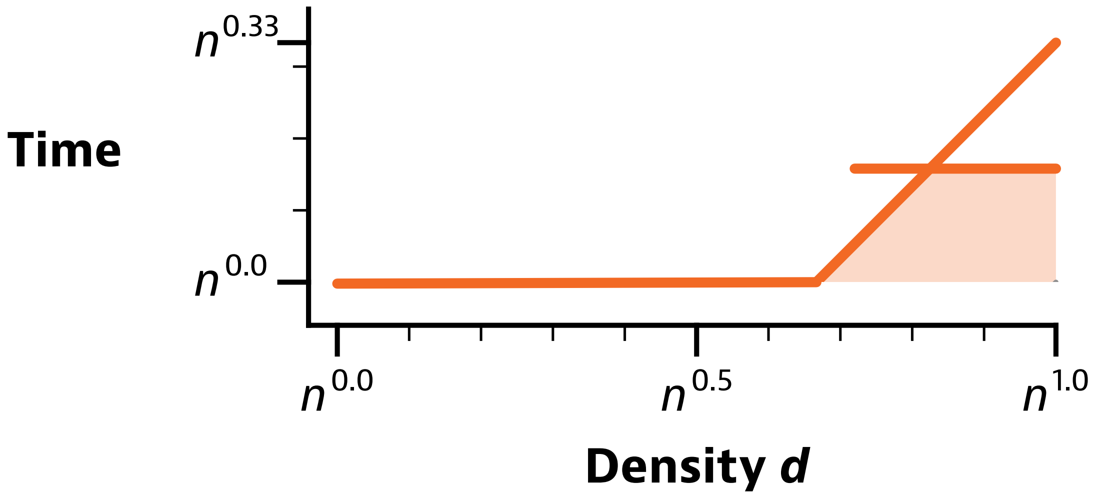
Low-bandwidth model

a.k.a. “node-capacitated clique” or “node-congested clique”

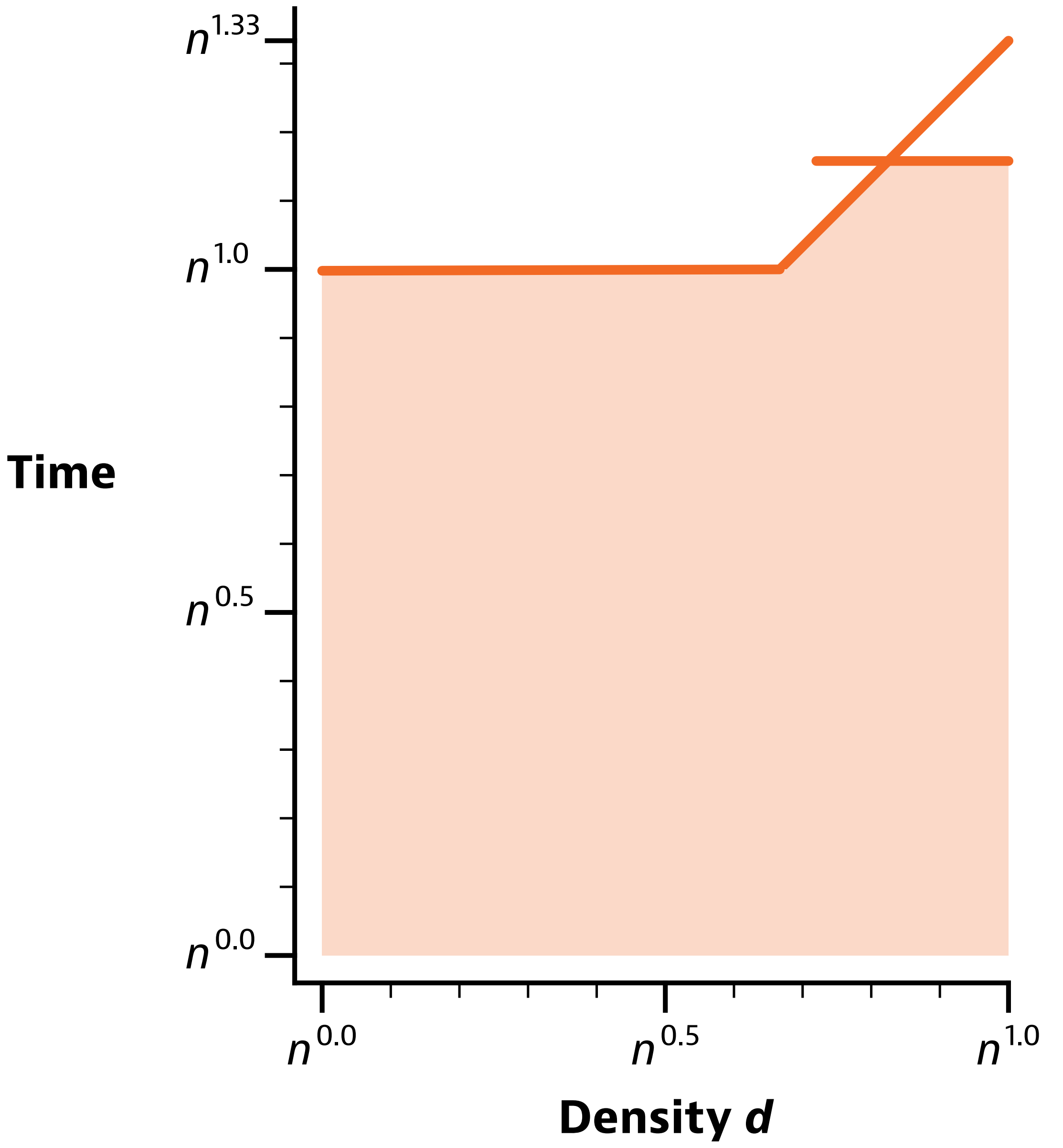


New problem setting

- **Input:** sparse $n \times n$ matrices A and B
  - computer i knows column i of A and column i of B
- **Output:** sparse n x n matrix $X = AB$
  - computer i has to output column i of X
- n computers
- $O(\log n)$ bits/computer/round

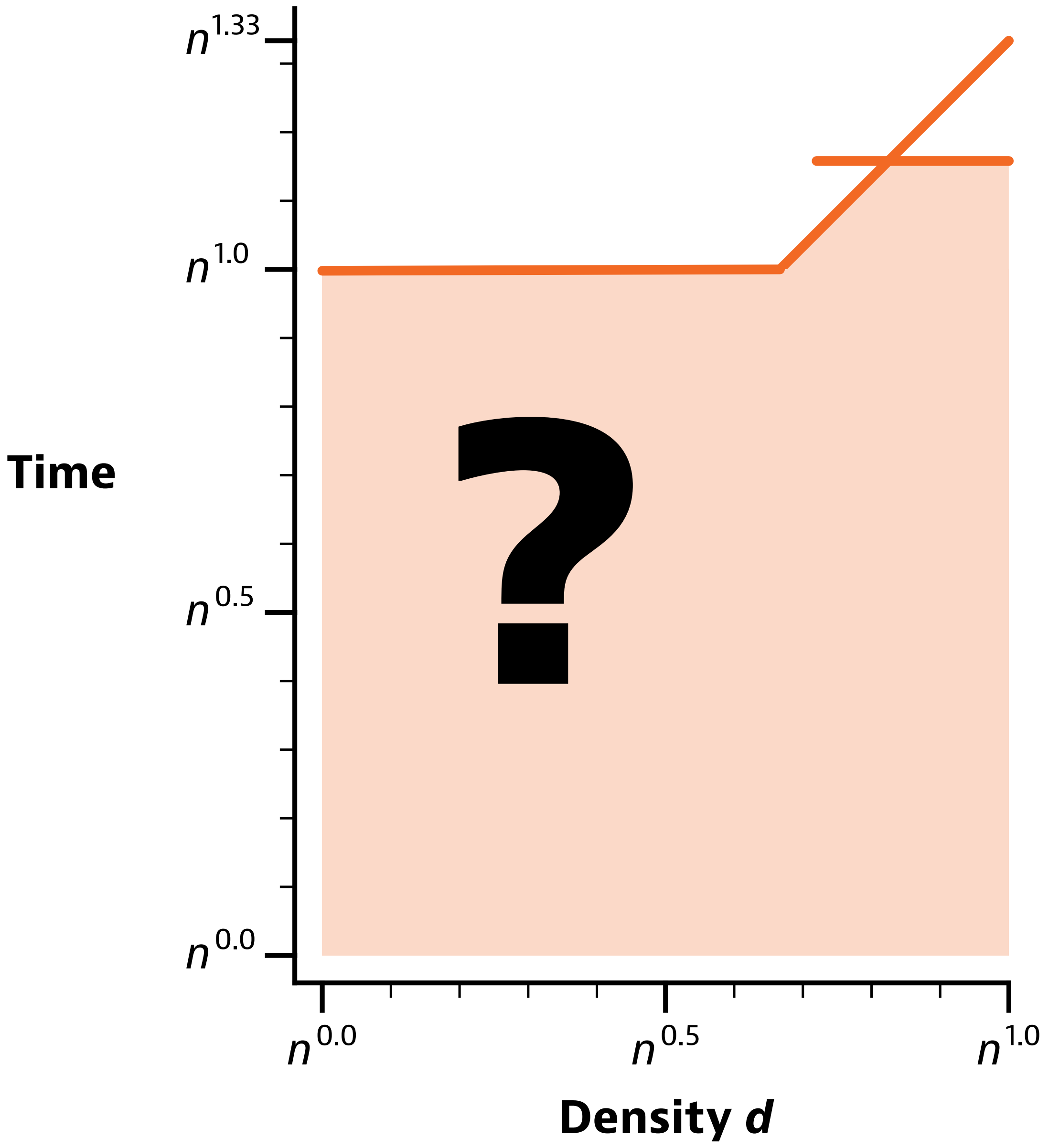


Complexity of sparse matrix multiplication in congested clique

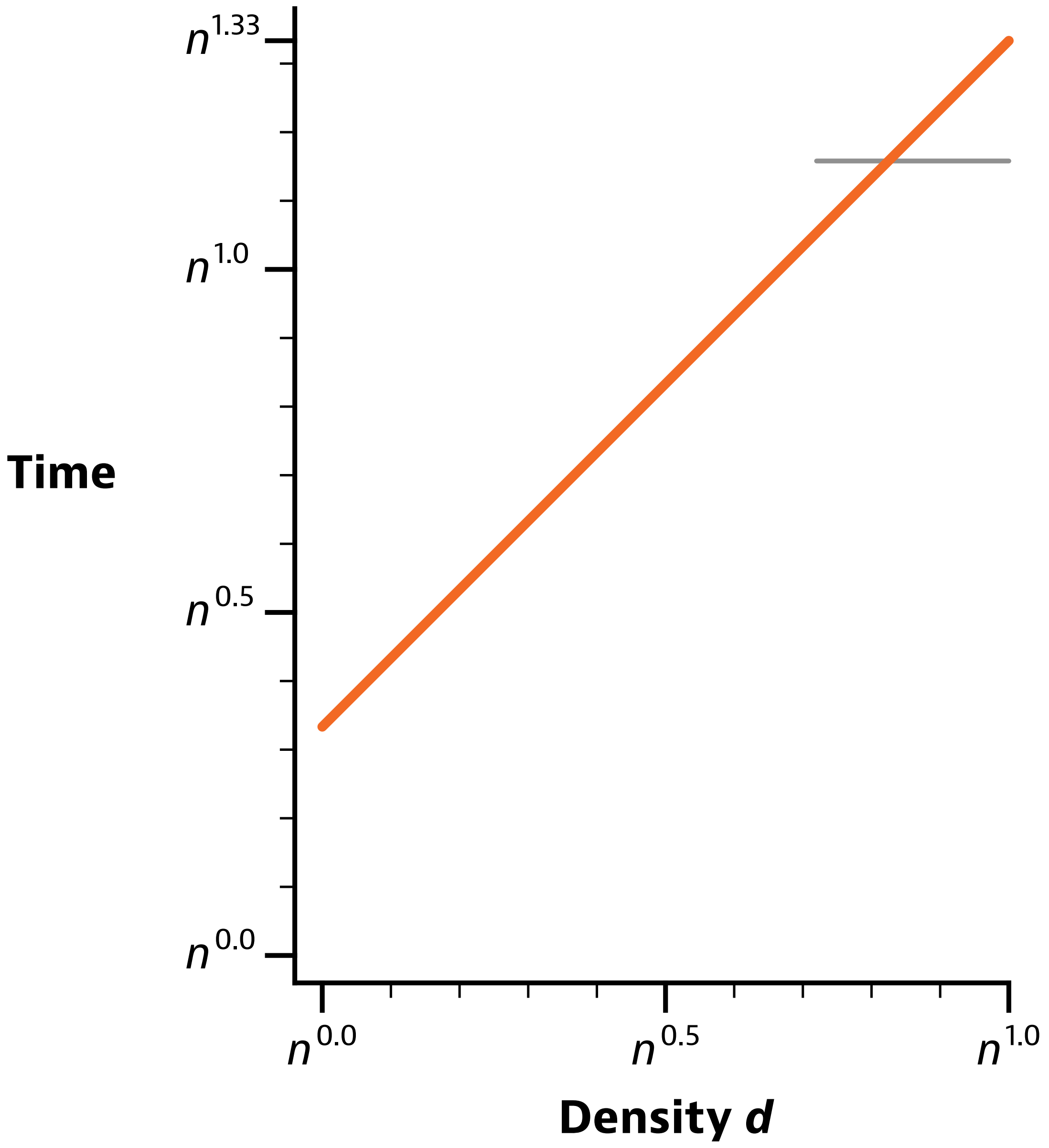


Complexity of sparse matrix multiplication in low-bandwidth model

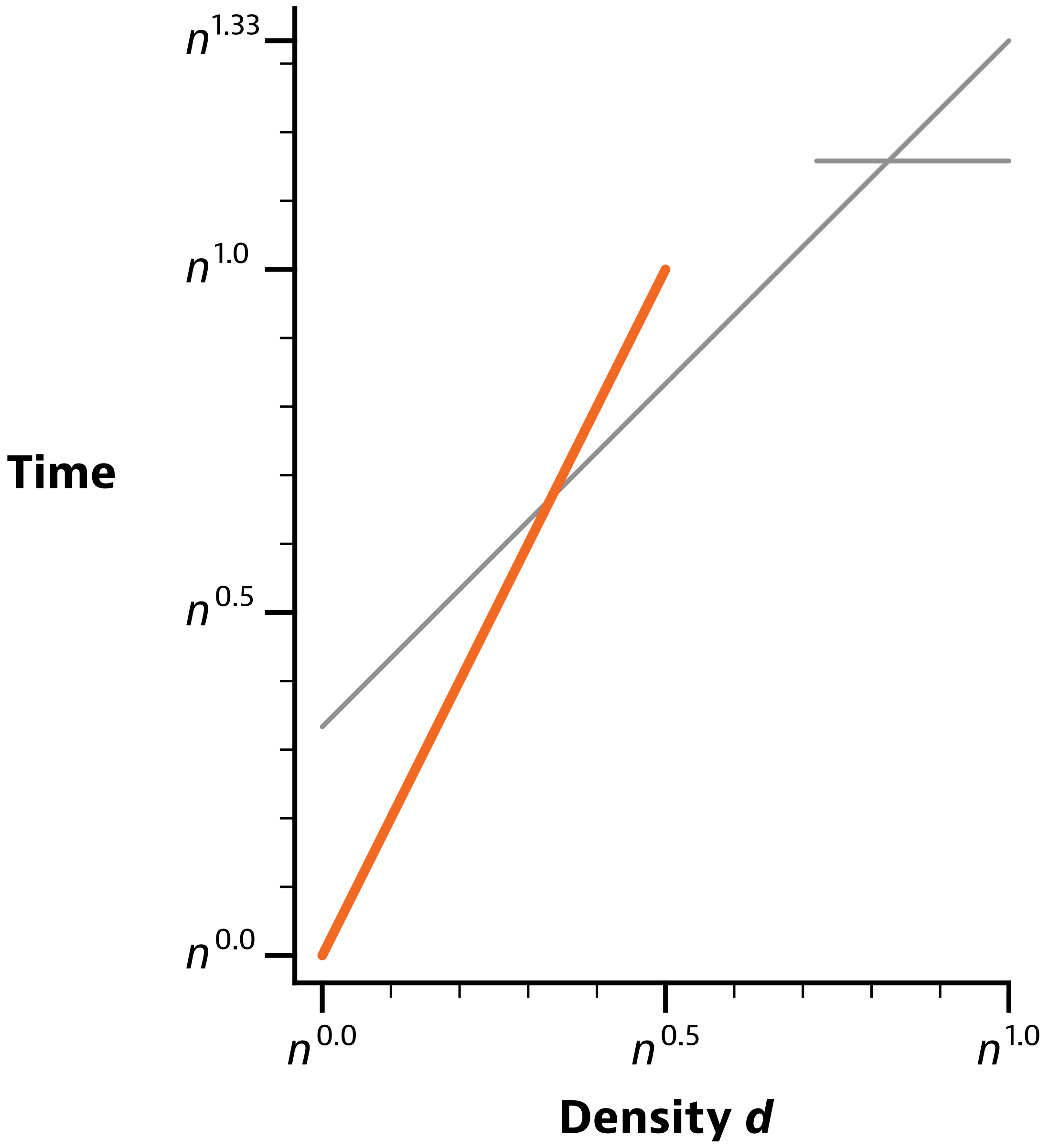
1 clique round can be simulated in *n* low-b/w rounds



Complexity of sparse matrix multiplication in low-bandwidth model

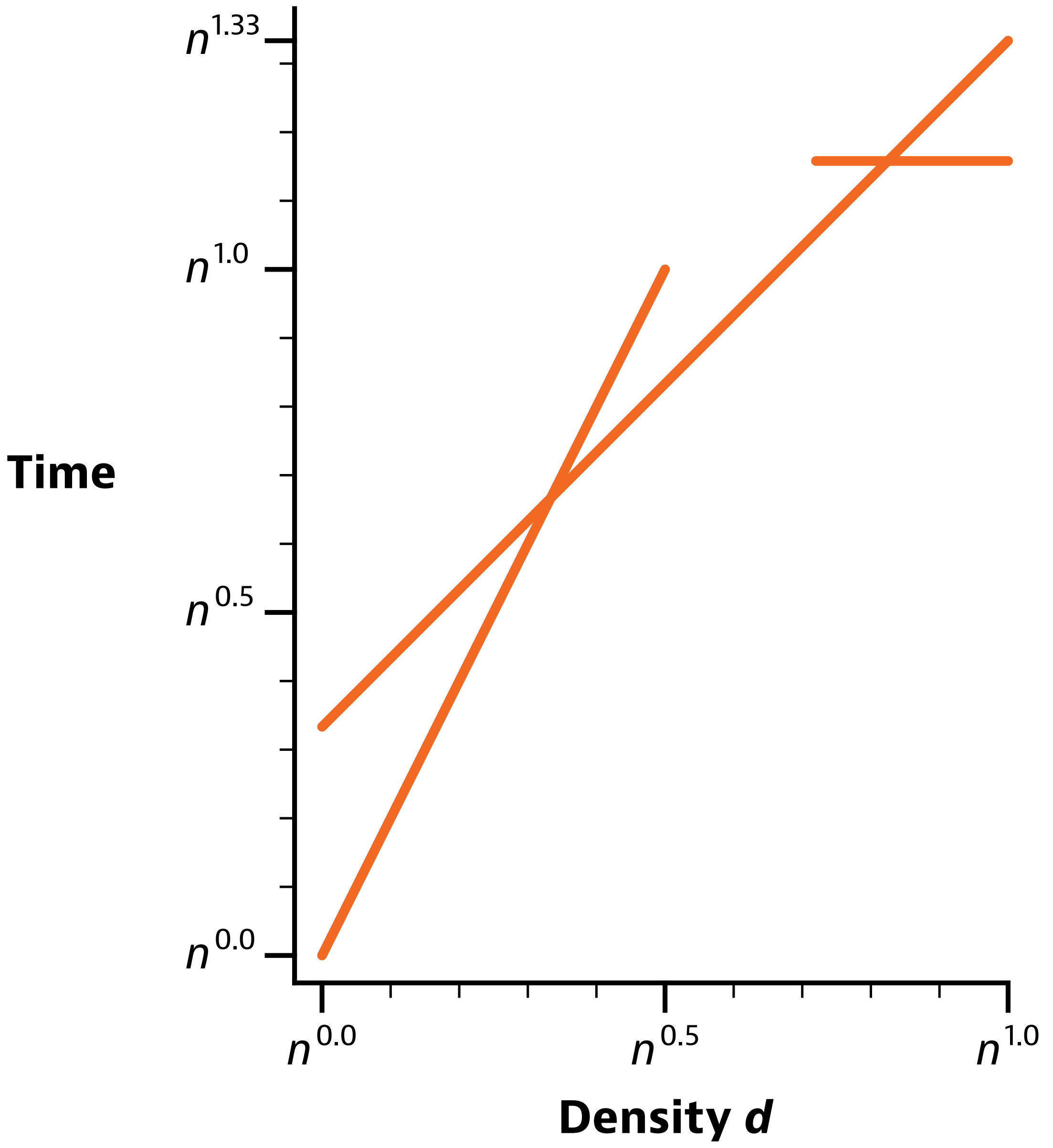


Censor-Hillel, Dory, Korhonen, Leitersdorf works also here

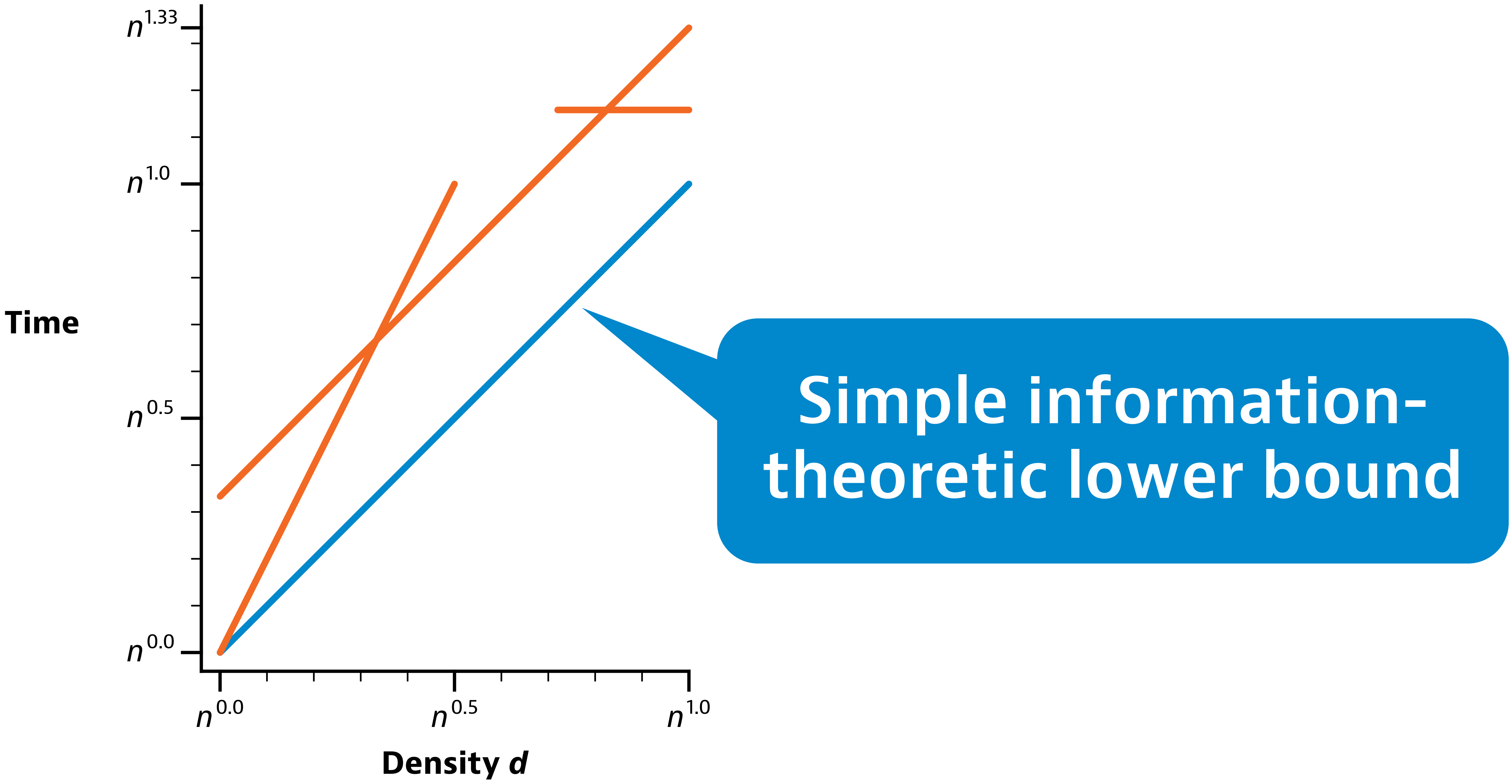


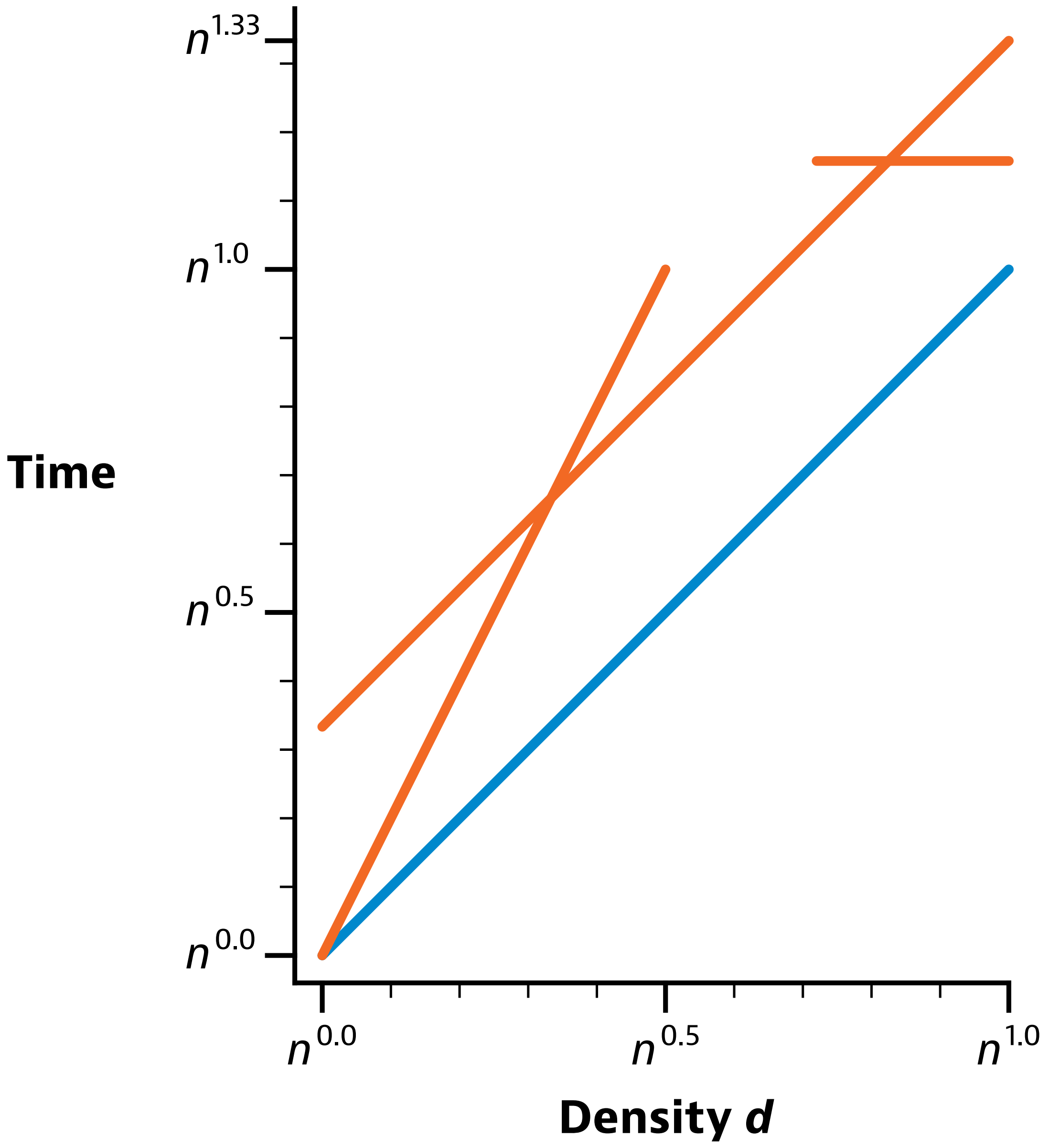
Very sparse matrices: trivial solution, $O(d^2)$-rounds

- Each computer outputs *d* results
- Each output depends on *d* inputs

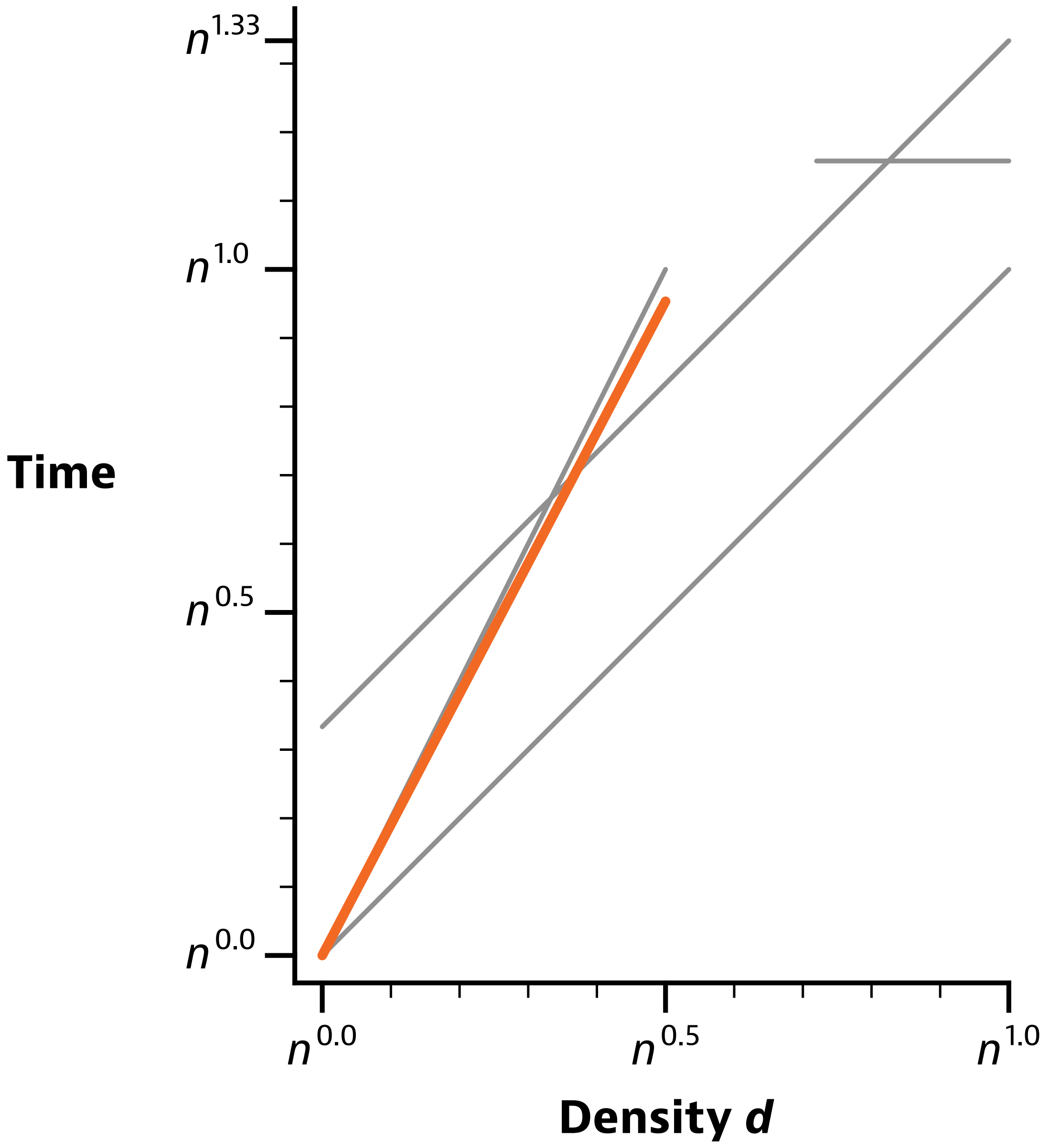


Are we done?

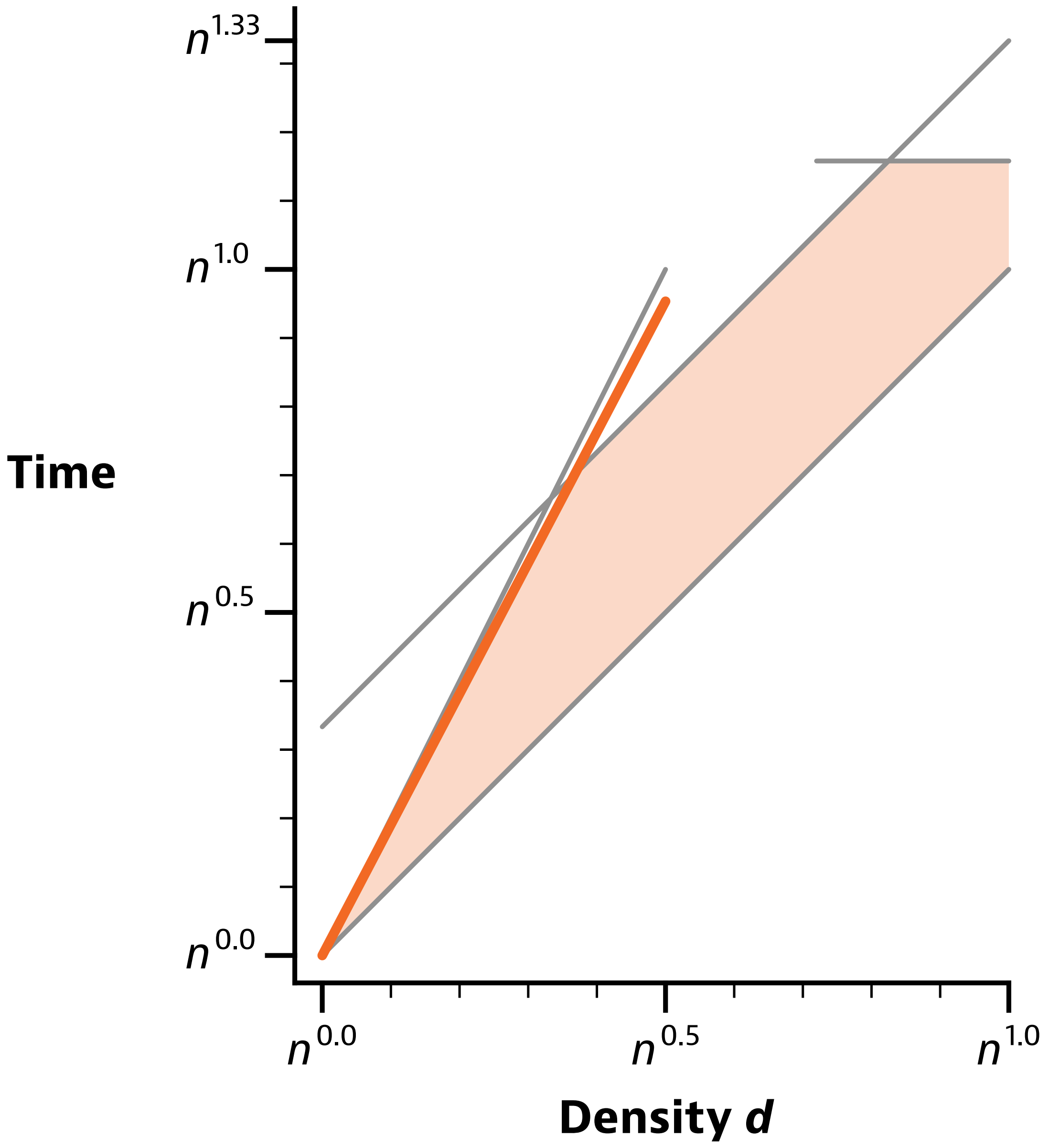

$n^{1.33}$
$n^{1.0}$
$n^{0.5}$
$n^{0.0}$
Time
$n^{0.0}$
$n^{0.5}$
$n^{1.0}$
Density *d*
Simple information-theoretic lower bound



Are we done?

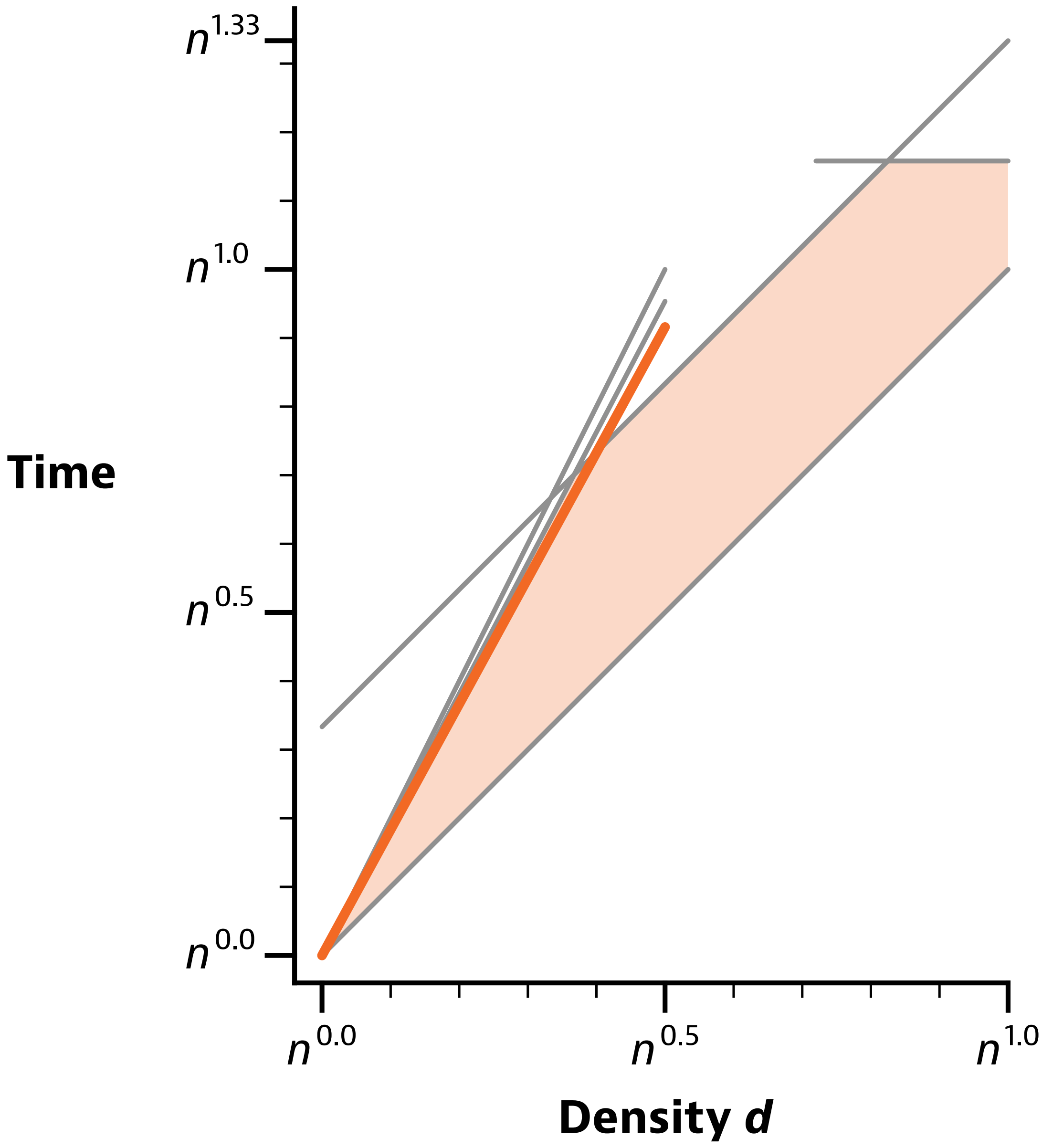


No, there is
a slightly better
algorithm for
sparse cases:
$O(d^{1.91})$ rounds



This is what the landscape looks like today

[SPAA 2022]



And it turns out that further improvements are possible: $O(d^{1.84})$ rounds

[unpublished]

How?

It's just triangles

If you can do matrix multiplication,
you can detect, count, etc. triangles

If you can "process" triangles,
you can do matrix multiplication

It's just triangles

"Process" triangle (i, j, k)

$$\approx$$

Add A_{ij} B_{jk} to X_{ik}

It’s just triangles

Dense matrix multiplication

$\approx$

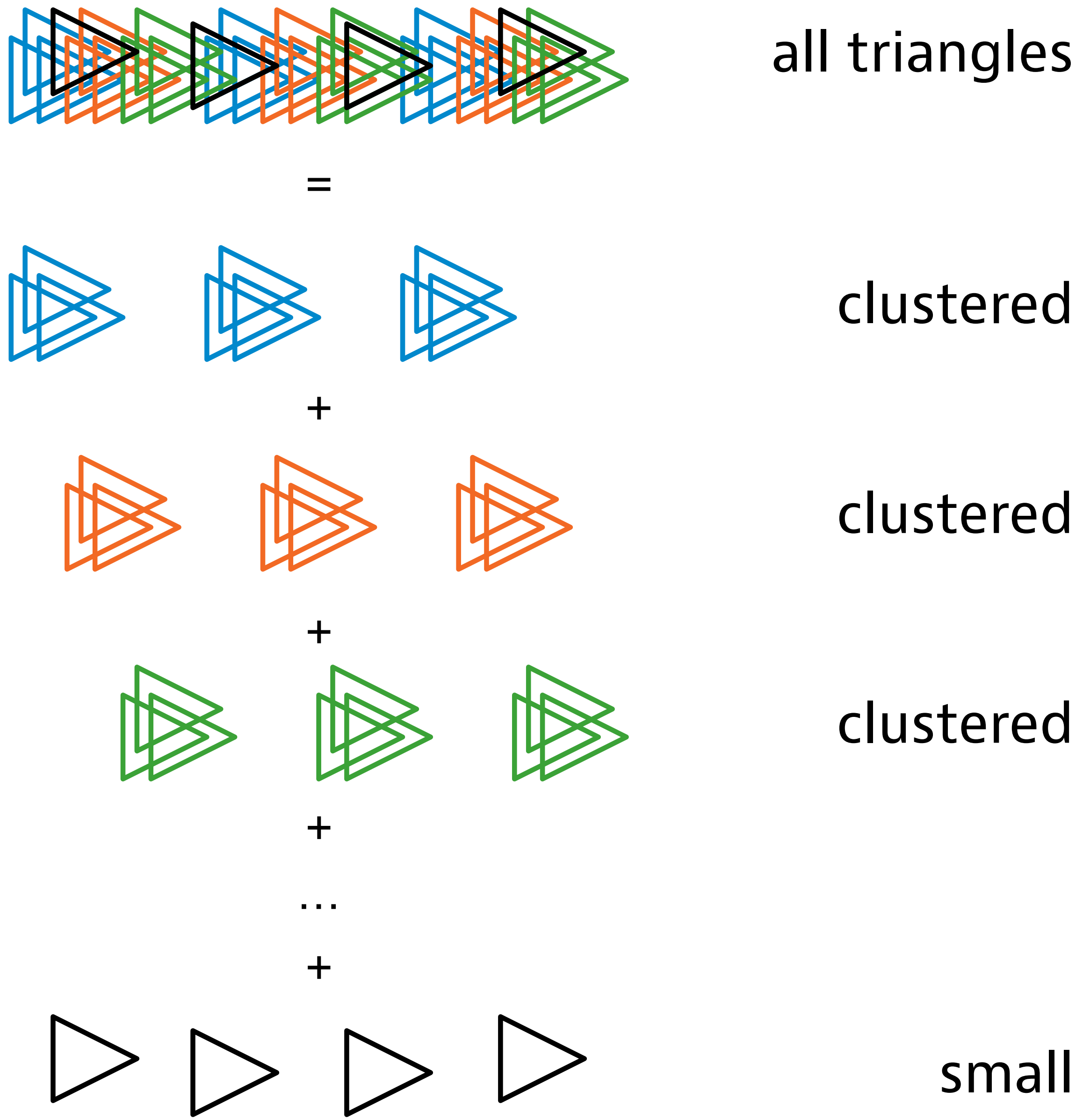
Batch-process many
overlapping triangles

It's just triangles

- **Many triangles:**
  - find clusters of overlapping triangles
  - batch-process with dense matrix multiplication
  - many triangles eliminated
- **Few triangles:**
  - can afford to process them individually

Key lemma:

If there are many triangles, there is a dense cluster



What next?

Beyond uniformly sparse

- **Different notions of sparsity:**
  - uniformly sparse
  - rows are sparse
  - columns are sparse
  - ***bounded degeneracy:*** can repeatedly find and eliminate a sparse row or column
  - average sparse …

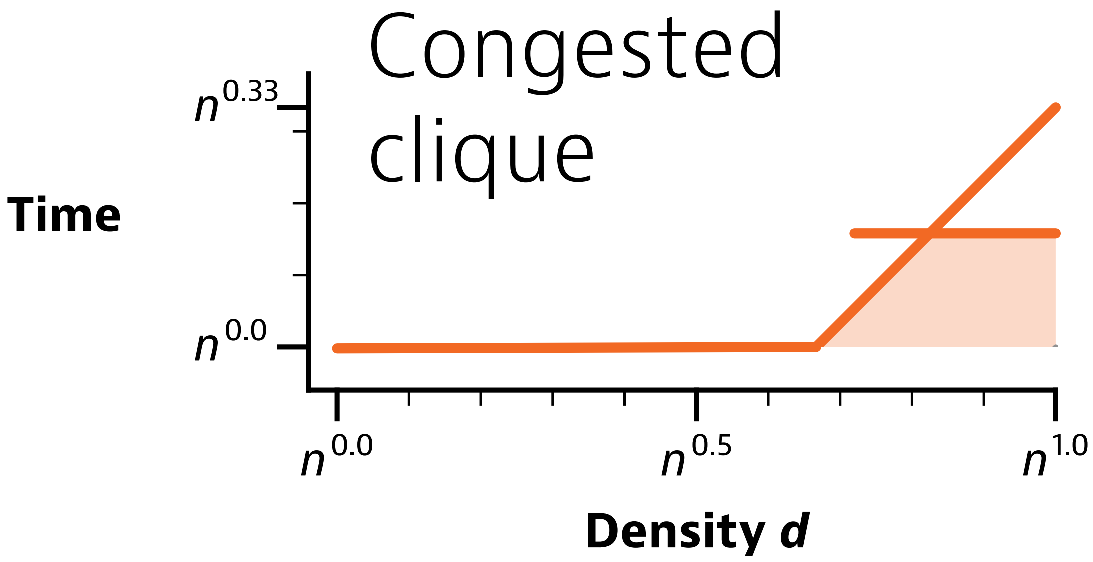
Beyond uniformly sparse

- **Different notions of sparsity**
- **Which of these admit:**
  - $o(d^2)$-round algorithms?
  - $O(d^2)$-round algorithms?
  - $O(d^2 + \log n)$-round algorithms?

Beyond uniformly sparse

- **Ongoing work:** answers to many of these questions coming!
- But these are **still open:**
  - if we can do something in $O(d^2)$ rounds,
    can we always push it down to $o(d^2)$ rounds?
  - could we go all the way to $O(d^{4/3})$ rounds?

Conclusions



Dense: split work following centralized algorithms

Sparse: process triangles