Cell-probe model: (for lower bounds)
- memory (DS) consists of \(w\)-bit cells
- just count # reads & writes
- computation is free
- typically assume \(w \geq \lg n\) or even \(w = \Theta(\lg n)\)

Dynamic connectivity lower bound \([\text{Pătraşcu & Demaine-}
\text{STOC 2004 & SICOMP 2006}]\)
- \(\Omega(\lg n)\) cell probes/op.
- holds even with amortization; here just worst case

Proof:
- consider \(n^2 \times n^2\) grid with perfect matching between consec. columns \(i\) & \(i+1\) \(\rightarrow\) permutation \(\pi_i\)
- block operations:
  - update \((i, \pi)\): \(\pi_i \leftarrow \pi\)
    \(= O(n^2)\) edge insertions/deletions
  - verify-sum \((i, \pi)\): \(\sum_{j=1}^{i} \pi_j = \pi?\) (\(\Sigma = \text{compose}\))
    \(= O(n^2)\) connectivity queries
- **Claim**: \(\sqrt{n^2}\) updates + \(\sqrt{n^2}\) verify-sums require \(\Omega(\sqrt{n^2} \cdot \sqrt{n^2} \cdot \lg n)\) time (cell probes)
  \(\Rightarrow\) dynamic connectivity requires \(\Omega(\lg n)\) time
Construction of bad access sequence:
- permutation $\pi$ in each $\text{update}(i, \pi)$ is chosen uniformly at random
- permutation $\pi$ in each $\text{verify-sum}(i, \pi)$ is the correct sum (but DS doesn't know)
- i's follow bit-reversal sequence: $000 \rightarrow 000 \rightarrow 001 \rightarrow 011 \rightarrow 010 \rightarrow 100 \rightarrow 110 \rightarrow 111$
- pairs: $\text{verify-sum}(i, \frac{\pi}{i})$
  $\text{update-sum}(i, \pi_{\text{random}})$
- tree over time:

- left & right subtrees of each node interleave

- **Claim:** for every node $v$ in tree, say with $l$ leaves in its subtree, during right subtree of $v$, must do $\Omega(l \sqrt{n})$ cell probes in expectation that read cells written during left subtree last

- summing over all levels (read $r$ of write $w$ is counted only at $\text{lca}(r, w)$)
  $\Rightarrow \Omega(n \lg n)$ lower bound overall
Proof of claim:
- left subtree has \( \frac{l}{2} \) updates with \( \frac{l}{2} \) rand. perms.
- any encoding of these permutations must use \( \Omega(l \sqrt{n} \lg n) \) bits [info. theory/Kolmogorov arg.]
- if claim doesn't hold, we'll derive a smaller encoding \( \Rightarrow \) contradiction.
- set up: know the "past" (before \( v \)'s subtree)
- goal: encode (verified) sums in right subtree
  \( \Rightarrow \) can recover (updated) perms. in left subtree

Warmup: query is \( \text{sum}(i) \rightarrow \frac{1}{2^n} \prod_j \)
- let \( R = \{ \text{cells read during right subtree}\} \)
  \( |W| = |\text{cells written during left subtree}| \)
- encode \( R \cap W \) (address & contents of each cell)
  \( \Rightarrow |R \cap W| \cdot O(\lg n) \) bits [assume poly. space, \( W = \Theta(\lg n) \)]
- decoding strategy for sums in right subtree:
  - simulate sum queries in right subtree
  - to read cell written in right subtree \( \Rightarrow \) easy
    in left subtree \( \Rightarrow \) \( R \cap W \)
    in past \( \Rightarrow \) known

\( \Rightarrow |R \cap W| \cdot O(\lg n) = \Omega(l \sqrt{n} \lg n) \)
\( \Rightarrow |R \cap W| = \Omega(l \sqrt{n}) \) \( \checkmark \)
Verify-sum instead of sum:
- permutations π given to verify-sum
  encode the information we want
- set up:
  - know (fixed) past
  - don’t know updates in left subtree
  - don’t know queries in right subtree
  - but know queries returned YES
- decoding idea:
  - simulate all possible input permutations
    for each query in right subtree
  - know one returns YES; all others return NO
- trouble: incorrect query simulation reads cells R'≠R
  - if read reR'\setminus R, it must be incorrect
  - but can’t tell whether re\setminus W'R or past\setminus(R\setminus W)
  - can’t afford to encode R or W
- idea: encode separator S for R\setminus W & W\setminus R
- when decoding, to read a cell
  written in right subtree ⇒ easy
  in R\setminus W ⇒ encoded explicitly
  in S ⇒ must be in past ⇒ known
  not in S ⇒ must not be in R ⇒ wrong guess ⇒ ABORT
- only one simulation will return YES;
  rest will return NO or ABORT.

⇒ |encoding| = \Omega(\log n)
Separators
- given universe \( U \) & a number \( m \)
- separator family & for size-\( m \) sets if
  \[ \forall A, B \subseteq U \text{ with } |A|, |B| \leq m \text{ & } A \cap B = \emptyset: \]
  \[ \exists C \subseteq U \text{ such that } A \subseteq C \text{ & } B \subseteq U \setminus C \]
- claim: there is a separator family &
  for size-\( m \) sets with \( |S| \leq 2^{O(m + \log |U|)} \)
  proof sketch: perfect hash family \( H \)
  with \( |H| \leq 2^{O(m + \log |U|)} \)
  gives mapping from \( A \cup B \) to \( O(m) \)-size table
  - store bit in each table entry: \( A \) vs. \( B \)
  - \( 2^{O(m)} \) such bit vectors
    \[ \Rightarrow 2^{O(m)} \cdot 2^{O(m + \log |U|)} = 2^{O(m + \log |U|)} \]

Encoding: \( R \cap W + \text{ separator of } R \cap W \cup W \cup R \)
  size = \( |R \cap W| \cdot O(\log n) + O(|R| + |W| + \log \log n) \)
  \[ = \Omega(l\sqrt{n}\log n) \]
  \[ \Rightarrow |R \cap W| = \Omega(l\sqrt{n}) \text{ or } |R| + |W| = \Omega(l\sqrt{n}\log n) \]
  \[ \Rightarrow \text{claim} \]
  \[ \Rightarrow \Omega(\log n) \text{ directly} \]
  Formally: if all ops. = \( o(\log n) \) \( \Rightarrow |R| + |W| = o(\log n) \)
  then claim holds