CSE491/596 Lecture Wed. 11/11: Time and Space Hierarchy Theorems

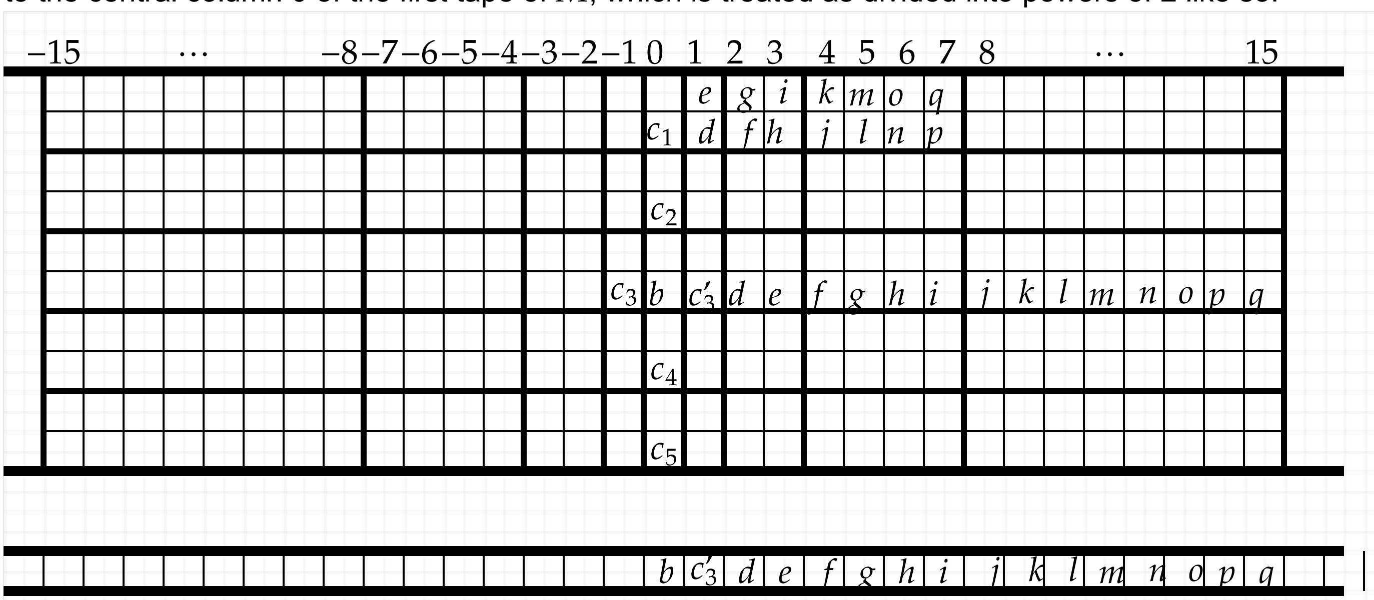
The following theorem implies that the Cook-Levin reduction can be made to run in quasilinear time (for languages that are in nondeterministic $n(\log n)^{O(1)}$ time to begin with).

Theorem 1 [Hennie and Stearns, 1966]: Every k-worktape DTM M (or NTM N) running in time $t(n)$ and space $s(n)$ can be simulated by a 2-worktape DTM M' (respectively, 2-worktape NTM N') that runs in time $O(t(n)\log t(n))$ and space $O(s(n))$. This holds whether the input tape is read-write and counted as a worktape, or read-only and counted separately.

Theorem 1' [Pippenger and Fischer, 1977]: **Moreover**, the simulating machine M' can be **oblivious**, meaning for all n and $x, y \in \Sigma^n$, the locations of the tape heads of M' at any timestep t are the same on input y as they are on input x. In consequence, $L(M)$ can be accepted by a highly uniform family $[C_n]_{n=0}^{\infty}$ of Boolean circuits, where each C_n has size $O(t(n)\log t(n))$.

Proof Sketch. The proof uses a caching scheme with amortized time analysis of successive doubling similar to that of the C++ `vector` class and memory management of arrays in other languages. First treat all tapes of M and M' as being two-way infinite. The k tapes of M are maintained as $2k$ "tracks" of the first tape of M' using work alphabet $\Gamma' = \Gamma^{2k}$.

The issue with a straightforward simulation of the k heads of M on these tracks is delay when the heads become widely spaced apart---then it takes up to $2s(n)$ steps by M' to read the k chars they are reading and execute the corresponding actions. The caching scheme keeps the k heads always close to the central column 0 of the first tape of M, which is treated as divided into powers of 2 like so:



The second tape is needed only to copy blocks of characters while shifting them into or out of higher cache levels nearer the "CPU" in column 0. These movements by M' are in response to head movements by M but the point of the Pippenger-Fischer refinement is that they can be scheduled in

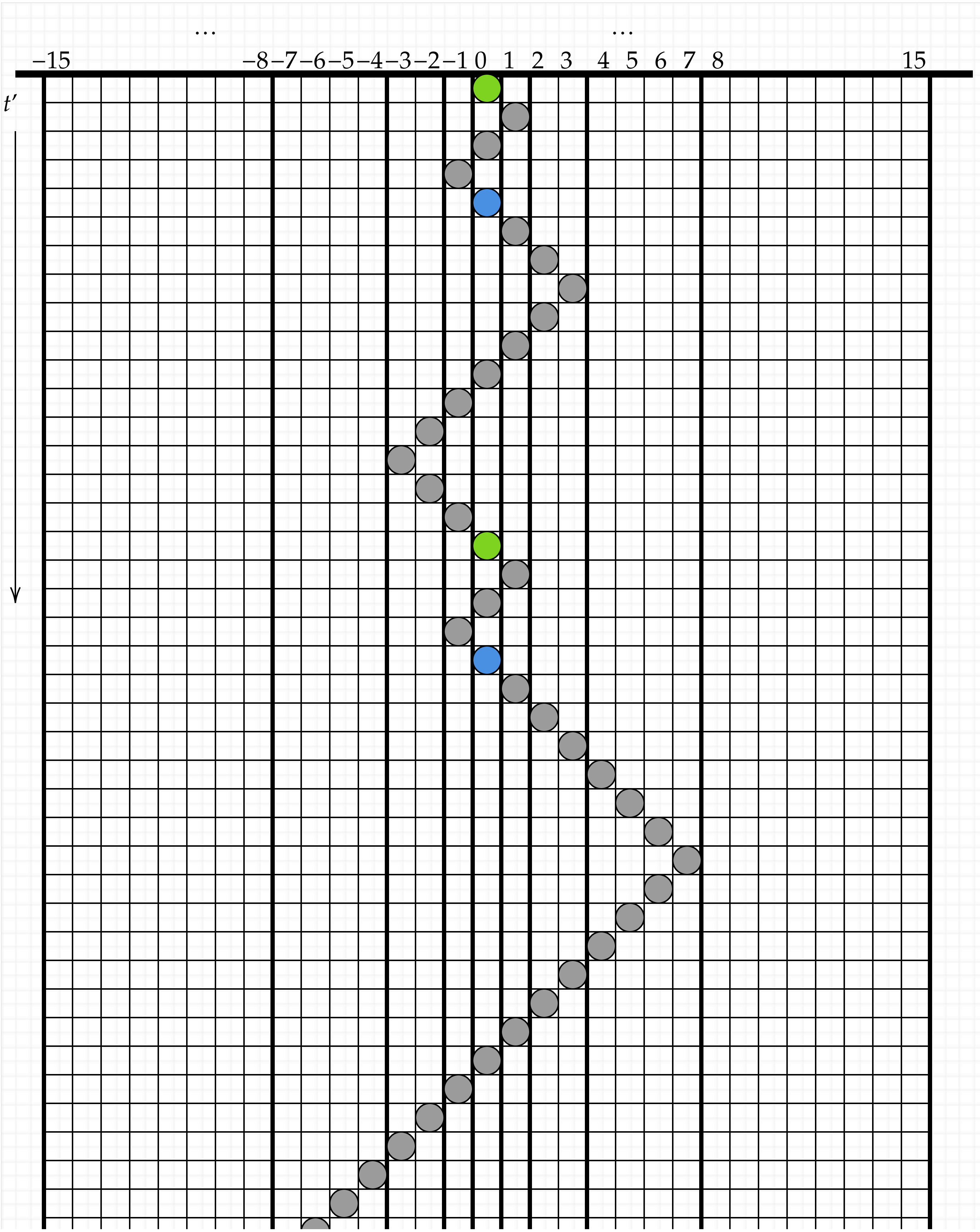
−15 ⋯ −8 −7 −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6 7 8 ⋯ 15

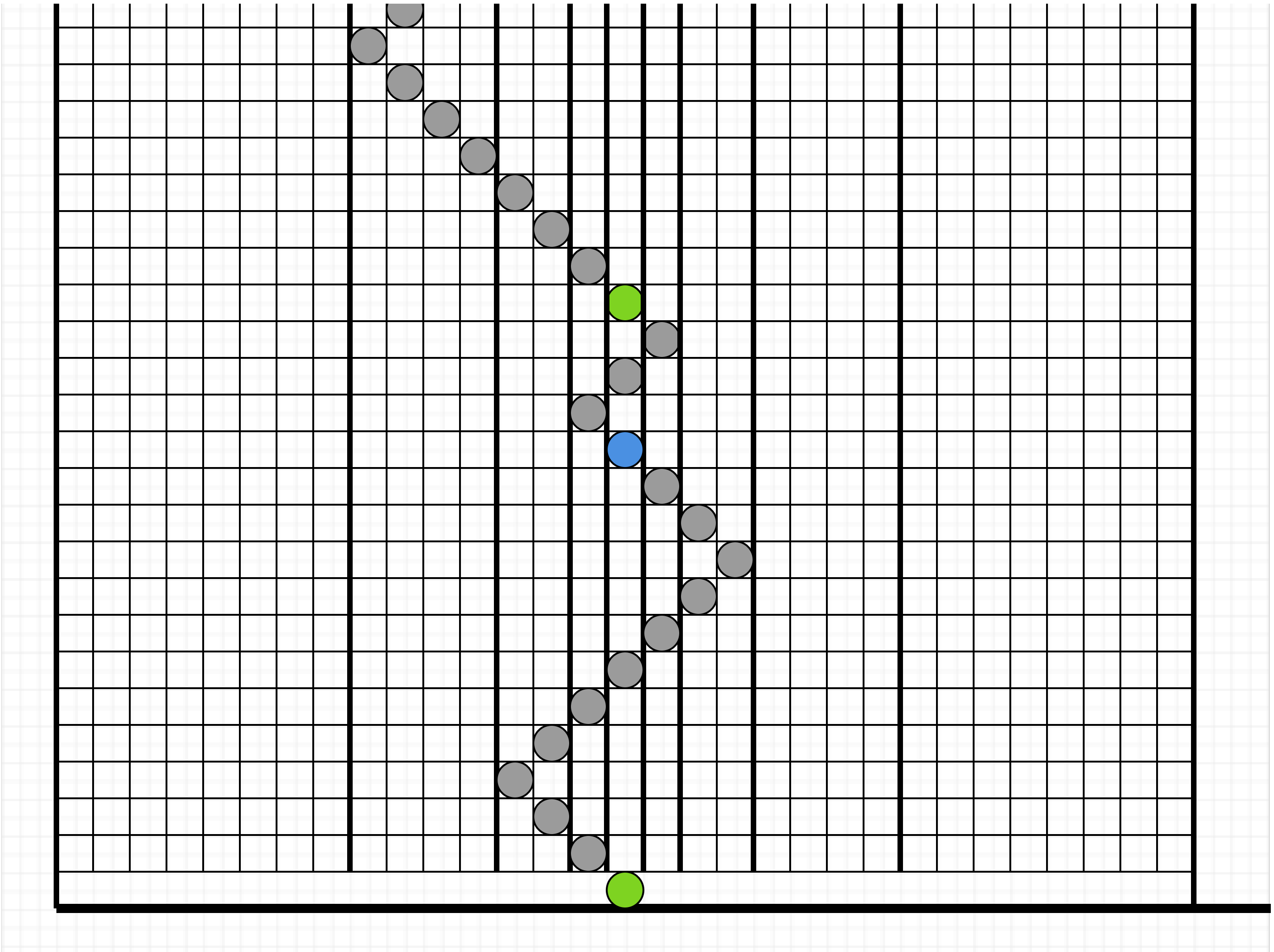
e g i k m o q

c_1 d f h j l n p

c_2

advance---with only the decision to shift-or-not-shift during the movement being taken on-the-fly. Every "inner jag" involving just columns -1, 0, 1 simulates a new step by M,while the outer jags may be needed to help set up for the next step:





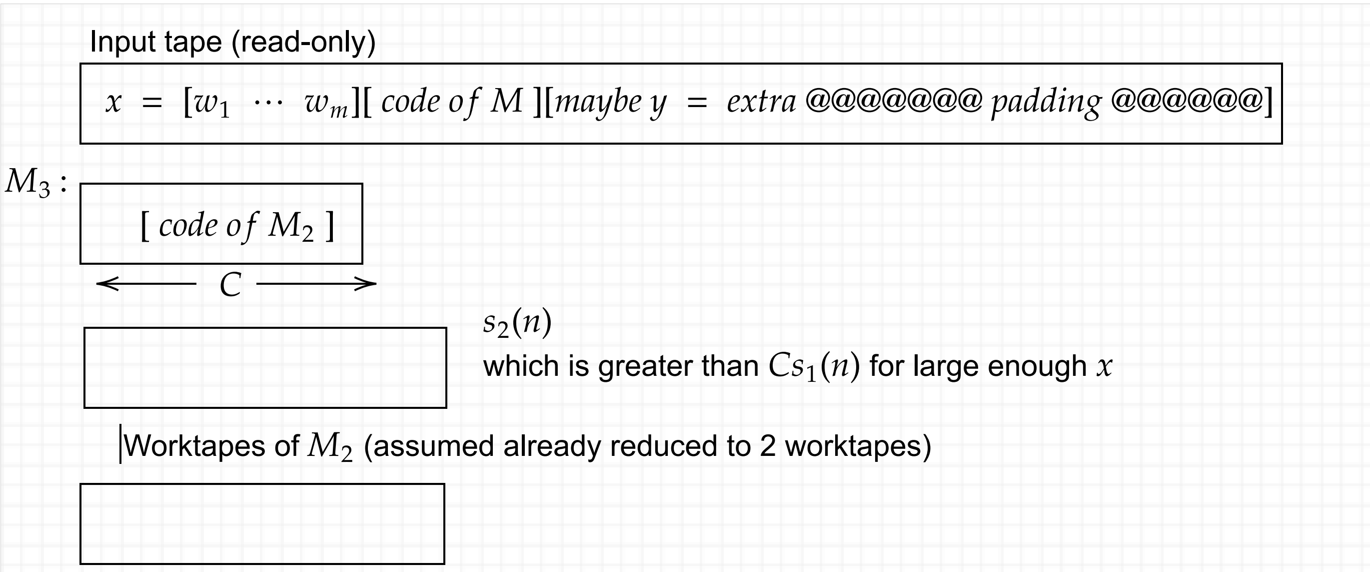
The time analysis simply needs computing the number of timesteps t' by M' as a function $g(t)$ of the number t of "inner jags". A proof by induction shows $g(t) = O(t \log t)$. ☒

This result is used only to justify that the Turing machines we give to our universal and "diagonal" simulators can be thought of as already having been reduced to 2 worktapes. The proof highlights the difference between a timestep of the machine M being simulated and each step **of the simulator M'**.

Offline Simulation Lemma: We can build a single 3-tape DTM M_3 with tape alphabet $\Gamma_3 = \{0, 1, \_\}$ such that for any DTM M with input alphabet $\Sigma = \{0, 1\}$ but any number k of tapes and work alphabet Γ_M of any size, there is a constant $C > 0$ such that for any $w \in \Sigma^*$ and $t > 0$, the first t steps of the computation of M on input w are simulated by the first $C + Ct\log(t)$ steps of M_3 on input $x = \langle w, M \rangle$, using at most C times as much space, where M_3 simulates $M(w)$.

Proof: The constant C depends on the given M. It does not depend on w or on the simulating machine's own input x. It mostly comes from the string length of the code $\langle M \rangle$ of M and reflects not only the number of states and instructions but also the overhead for encoding Γ_M by the binary-plus-blank alphabet Γ_3. It also gets a contribution from the constant factor in the $O(\log t)$ time overhead for reducing k tapes to 2 tapes to produce the machine M' above, and then re-code its big alphabet over

$\{0, 1, \_\}$ to produce a machine we call M_2. Note that going from time t to time $O(t \log t)$ is markedly better than the $O(t^2)$ time shown in class for getting down to a single tape. The machine M_3 on input $\langle w, M \rangle$ first copies the M part to its third tape. The w part is an "extra" that can be ignored in the main proof (i.e., pretend $w = \epsilon$); it would come into play if we talked about "universal languages for complexity classes," not just diagonal languages for them. Here is a picture:



The rest of the proof is by optical inspection of this memory map. There are two sources of slowdown:

- It can take up to C steps for M_3 to find the next instruction in the code of M as given. (There's also the initial C steps to copy the code of M to the third worktape to begin with.)
- Every character in the work alphabet Γ_M---which might be huge coming out of the k-track tape construction---has to be recorded and updated in binary code. But the extra time and space for this is at most the constant factor C depending only on M again.

But the slowdown is never more than this, working step-of-M by steps-of-M'. If $M_2(w)$ halts within t' steps of its own time-clock, the time for $M_3(w, M, maybepadding)$ is at most $C + Ct'$. And the space used is at most $C + Cs(n)$. ☒

Now we are ready to employ the padding feature to prove the main results:

Space Hierarchy Theorem:
If s_1, s_2 are "reasonable" space functions and $s_1(n) = o(s_2(n))$, then DSPACE$[s_1(n)]$ is properly contained in DSPACE$[s_2(n)]$.

Thus for example, in cases starting with $s_1(n) = \log_2 n$ and $s_2(n) = \log^2(n) \stackrel{\text{def}}{=} (\log n)^2$, we get:

$$\begin{aligned}\text{DLOG} &\subsetneq \text{DSPACE}\left[(\log n)^2\right] \subsetneq \text{DSPACE}\left[(\log n)^3\right] \subsetneq \cdots \subsetneq \text{DSPACE}[O(n)] \\ &\subsetneq \text{DSPACE}[n \log n] \subsetneq \text{DSPACE}\left[n^2\right] \subsetneq \text{DSPACE}\left[n^3\right] \subsetneq \cdots \subsetneq \text{PSPACE}\end{aligned}$$

The hierarchy for deterministic time is almost as tight:

Deterministic Time Hierarchy Theorem:
If t_1, t_2 are "reasonable" time functions and $t_1(n)\log(t_1(n)) = o(t_2(n))$, then $\mathsf{DTIME}[t_1(n)]$ is properly contained in $\mathsf{DTIME}[t_2(n)]$.

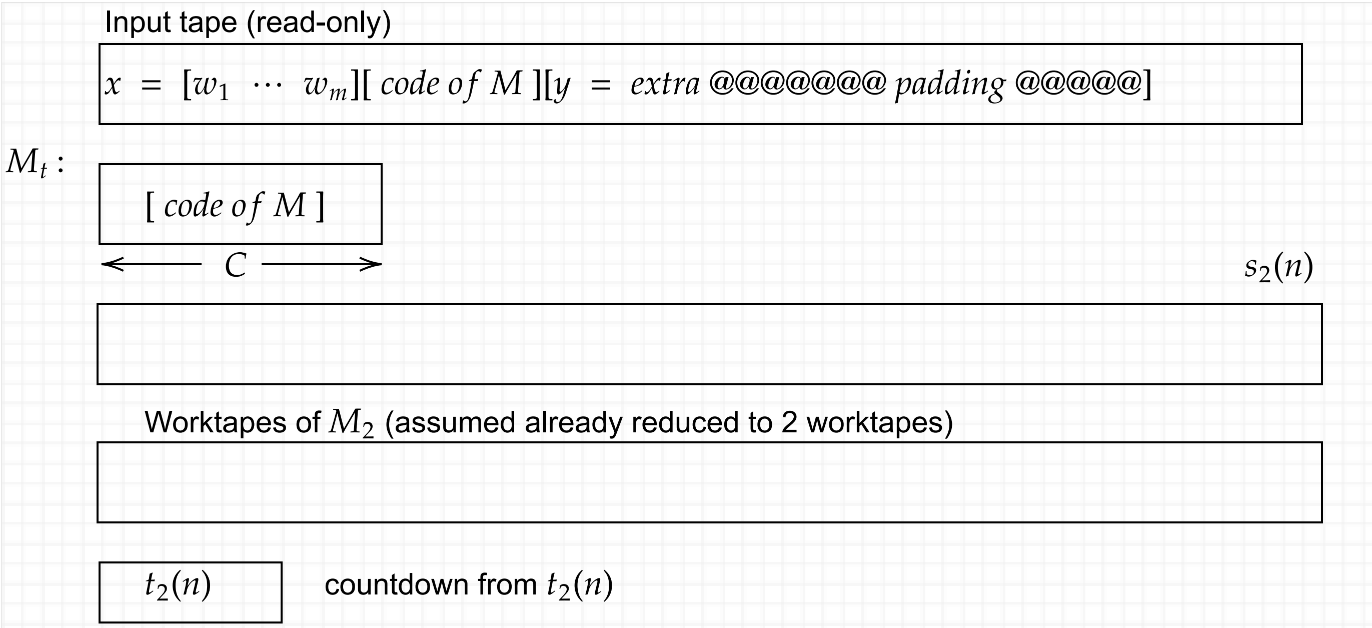
In particular, this means that even *within* P, deterministic time is quite stratified:

$$\mathsf{DLIN} \stackrel{\text{def}}{=} \mathsf{DTIME}[O(n)] \subsetneq \mathsf{DTIME}\left[n^{1.000001}\right] \subsetneq \mathsf{DTIME}\left[n\sqrt{n}\right] \\ \subsetneq \mathsf{DTIME}\left[n^2\right] \subsetneq \mathsf{DTIME}\left[n^3\right] \subsetneq \cdots \subsetneq \mathsf{P}.$$

So why can't we tell that **SAT** does not belong to $\mathsf{DTIME}\left[n^{1.000001}\right]$, let alone that it does not belong to P? A good question! The best road for understanding the issue is to see how the common proof of both theorems works. The notes by Debray prove only a weaker version with $\sqrt{t_1(n)}$ in place of the $\log(t_1(n))$ factor, and the reason the professor at Stanford did this is that existing presentations of the stronger result are so 'yucky' that Allender and Loui and I didn't prove them in our notes either. However, I found the above way to roll several technical propositions into a single statement that gives the springboard for the final diagonalization step of the proof.

Proof of the Time and Space Hierarchy Theorems:
We describe diagonal languages $D_s \in \mathsf{DSPACE}[s_2(n)] \setminus \mathsf{DSPACE}[s_1(n)]$ and $D_t \in \mathsf{DTIME}[t_2(n)] \setminus \mathsf{DTIME}[t_1(n)]$ in terms of machines M_s and M_t that expressly run within the space bound $s_2(n)$ and time bound $t_2(n)$, respectively. Since their descriptions differ only in the initial detail, we describe both machines in the same breath. They each have the same three tapes as M_3 above, plus M_t has a fourth tape to count up to $t_2(n)$---which is possible by the definition of $t_2(n)$ being "reasonable" (as said in Debray's notes---see note at the end on "fully time constructible" as said in other sources). The point is that the machines M_s and M_t "embody" M_3 but have three differences:

- The enforce the space bound $s_2(n)$ and/or the time bound $t_2(n)$ on themselves;
- They run M not just on the given w but on the whole input tape $x = \langle w, M, y\rangle$; and
- They make the opposite accept/reject decisions from M_3 simulating $M(x)$.



On any input x, taking $n = |x|$, M_s lays out $s_2(n)$ tape cells that its run of M_3 will be allowed to use, while M_t starts counting down from $t_2(n)$.

Both machines try to decode $x = \langle w, M, y \rangle$ for some Turing machine M. If this is not possible, they reject x.

On success, they begin simulating $M_2(x)$. Note that the "own code" $\langle M \rangle$ remains part of x, as does the "padding" y. Since the $\langle M \rangle$ part still gets copied to the third tape, this is a real-not-virtual run of M_2 with no overhead. If the simulation doesn't stay within the $s_2(n)$ marked-off cells in M_s, or takes longer than $t_2(n)$ steps in M_t, the overstep is immediately detected and the machine rejects x.

Otherwise, the run of $M_2(x)$ successfully completes. If M accepts x, then M_s and M_t each **reject** x. If M rejects x, that's when M_s and M_t **accept** x.

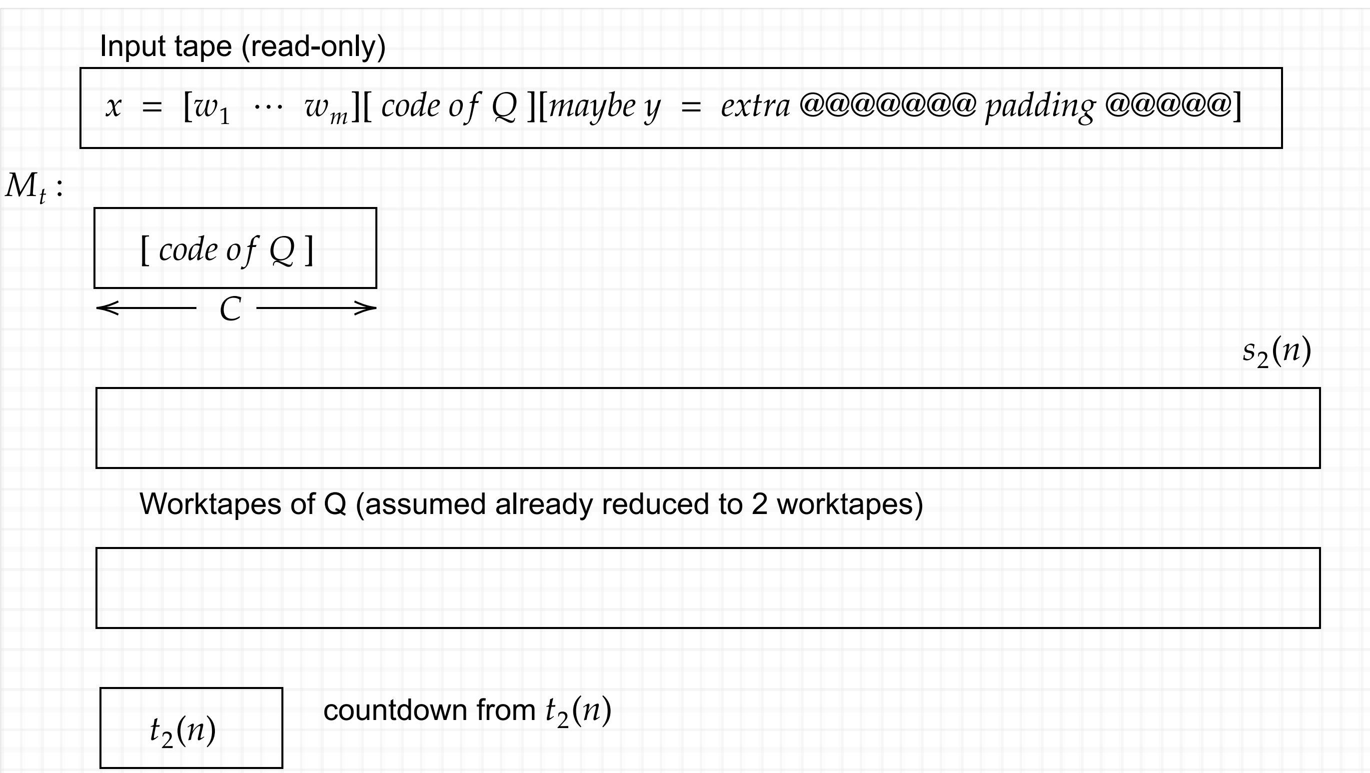
Considering first the case of space, M_s enforces the $s_2(n)$ space bound on itself, so $D_s \stackrel{\text{def}}{=} L(M_s) \in$ DSPACE$[s_2(n)]$. Now suppose we had $D_s \in$ DSPACE$[s_1(n)]$. Then there would be a DTM Q running in $s_1(n)$ space such that $L(Q) = D_s$. Now consider what happens when M_s runs on inputs of the form $x = \langle w, Q, y \rangle$:

1. After taking $n = |x| = |\langle w, Q \rangle| + |y|$ and laying out $s_2(n)$ tape cells, M_s successfully decodes x into $\langle w, Q \rangle$ and y.
2. M_s segues into simulating $M_3(\langle Q, x \rangle)$ step-for-step. There is a constant C depending only on Q such that this takes at most $C + Cs_1(n)$ tape cells. What's important from the **Offline Simulation Lemma** is that C doesn't change if the padding-y part of x changes.

3. The space usage by $M_3(\langle Q, x\rangle)$ still could overstep the boundaries laid out by M_s. But by $s_1(n) = o(s_2(n))$, for all C there is an n_0 such that whenever $n \geq n_0$, $C + Cs_1(n) \leq s_2(n)$. We may also wlog. suppose that $n_0 \geq |\langle w, Q\rangle|$.
4. So consider what happens on the particular input $x = \langle w, Q, y\rangle$ with $y = @^{n_0 - |\langle w, Q\rangle|}$. Then x has length $n = n_0$, so $C + Cs_1(n) \leq s_2(n)$.
5. Thus the simulation of $M_3(\langle Q, x\rangle)$ stays within the bound and runs to completion. So $M_s(x)$ gives the opposite answer to $M_2(x)$.
6. But $M_2(x)$ gives the same answer as $Q(x)$, so we get $M_s(x) \neq Q(x)$. This contradicts $L(Q_s) = D_s$.

As with the original "diagonal contradiction," this implies that the "quixotic" machine Q running in space $s_1(n)$ cannot exist. So D_s does not belong to DSPACE$[s_1(n)]$.

The argument for time is entirely similar. The $t_1(n) \log t_1(n) = o(t_2(n))$ business is a bit of a red herring. The conclusion really is that if $t_0(n) = o(t_2(n))$ then M_t can accept a language in time $t_2(n)$ that **cannot** be accepted by a **2-worktape** machine Q in time $t_0(n)$. The simulation for $t_1(n) \log t_1(n) = O(t_0(n))$ then extends this conclusion down to DTIME$[t_1(n)]$.



Suppose Q accepts $D_t \stackrel{\text{def}}{=} L(M_t)$ in time $t_1(n)$. Then for any y, M_3 on input $\langle Q, x\rangle$ where $x = \langle w, Q, y\rangle$ stops within $Ct_0(n) + C$ steps, where the constant C depends only on Q. Since $t_1(n) \geq n+1$ by assumption about time functions, we can add in the initial $2n$ steps for decoding x into $\langle w, Q, y\rangle$ and get n_0 such that for all $n \geq n_0$, $Ct_0(n) + C + 2n + [\textit{time to initialize } t_2(n)] \leq t_2(n)$. Thus on the input $x = \langle w, Q, @^{n_0 - |\langle w,Q\rangle|}\rangle$ defined as before, the whole run by $M_t(x)$ finishes $M_3(\langle Q, x\rangle)$ and gives the opposite answer before the $t_2(n)$ "clock" counts all the way down and "rings." So $L(M_t) \neq L(Q)$, which contradicts $L(Q) = D_t$. ☒

A key hidden detail here is that the process of initializing the countdown clock to $t_2(n)$ must itself run within $t_2(n)$ time (on any input x of length n). This is the definition of $t_2(n)$ being **fully time constructible**---and finally we see why it is included under the notion of being a "reasonable" time bound. Of course we should expect this property for "natural" time-bounding functions: polynomials, exponentials...but the notion is needed to "protect" the theory from paradoxes that could arise from weird functions being used as time bounds [such as n^c for an uncomputable number c as the power, but actually much weirder ones are the issue].

A word-to-the-wise about $\mathrm{DTIME}\left[n^c\right]$: If c is a rational number ($c \geq 1$), then the time function $t(n) = n^c$ is "reasonable." If c is an **uncomputable** irrational number, however, then **not**. If c is a computable irrational number, *hmmmmm*... it depends... But it is worth pointing out that for any real numbers $c < d$, there are rational numbers q, r such that $c < q < r < d$.