Turing Machines

Key takeaways

1. Runtime is the first step at which the full configuration repeats.
2. The table hides the runtime: a tiny machine can run for millions of steps.
3. Measure short runs; estimate machines whose runtime exceeds the budget.

Common misunderstandings of Turing-machine runtime

Which claim would you defend right now? Choose before the evidence.

"A Turing machine's tape is always infinite and starts at its left edge."

"Runtime ends only when the machine returns to its starting configuration."

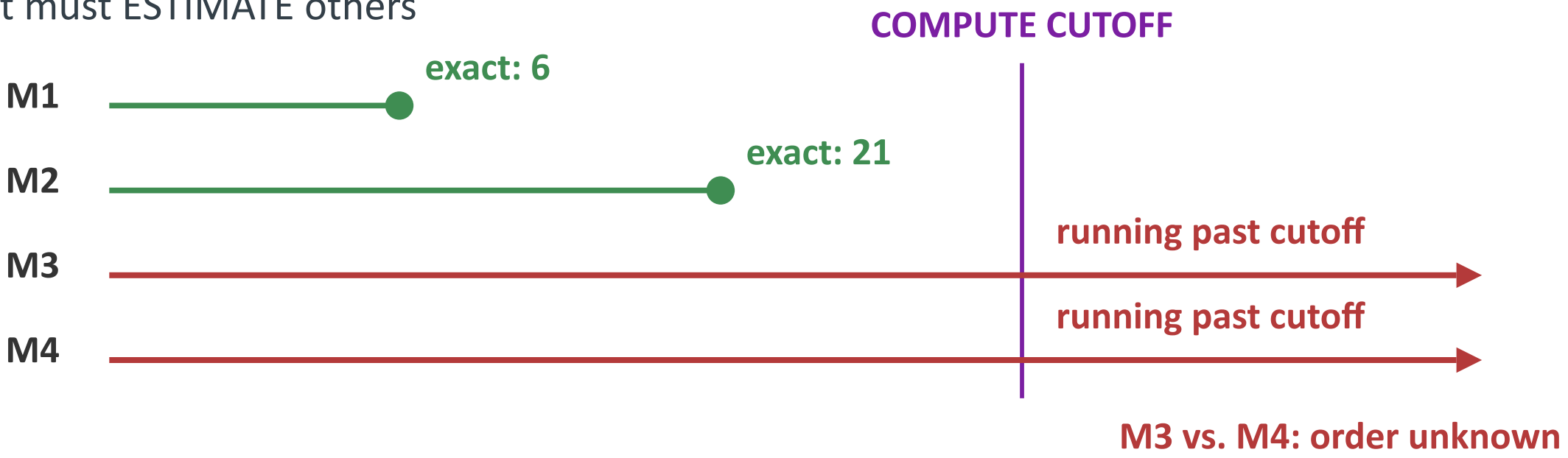
"A small transition table implies a short runtime."

Keep your choice. Look for the mechanism or counterexample that would make it fail.

Why ranking machines by runtime is hard

A head, a tape, a table — the model behind Project 2

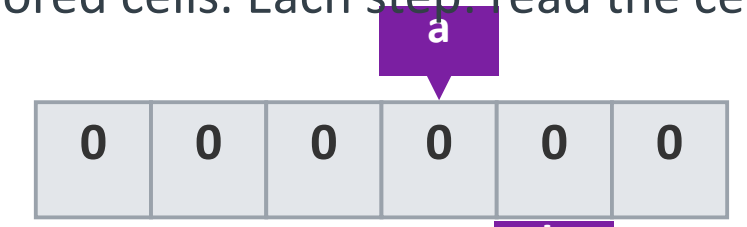
- Project 2 (tm-compute) hands you seven hidden machines and asks: rank them by how long they run
- “Runtime” = the number of steps until the machine’s exact configuration first repeats
- This deck is the model behind that game — and the reason you can MEASURE some machines but must ESTIMATE others



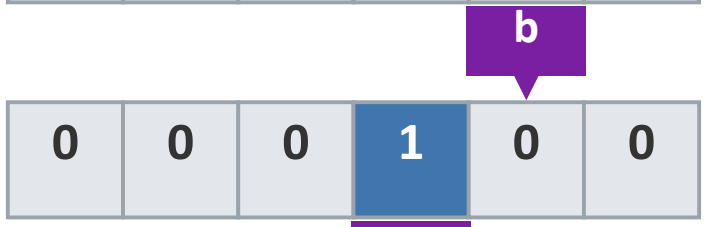
Watch a machine step

A head sits on a tape of colored cells. Each step: read the cell under the head, then write, change state, and move by one.

start

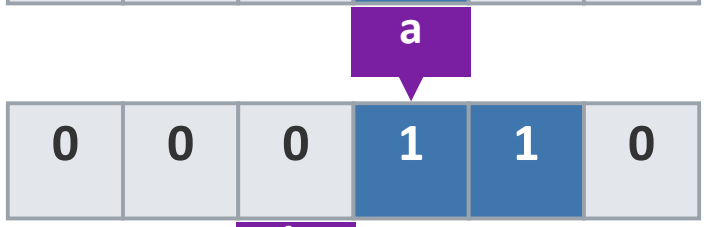


step 1



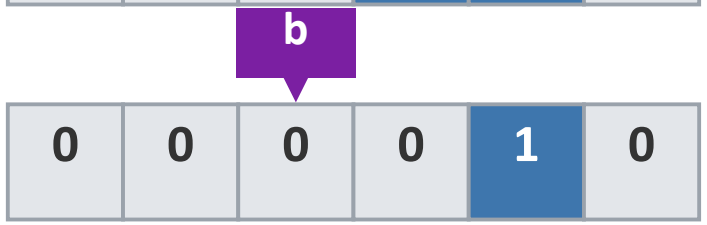
(a,0) → write 1, head→b, move +1

step 2



(b,0) → write 1, head→a, move −1

step 3



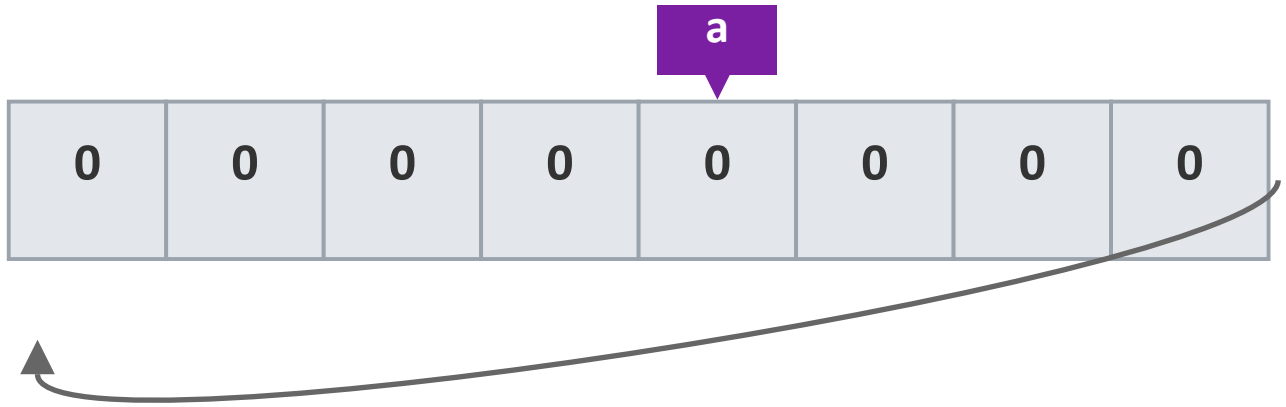
(a,1) → write 0, head→b, move −1

tm-compute's tape is a ring

tm-compute's tape is a RING and the head starts in the middle — not the textbook one-way tape. This trips people up.

center cell = tape_size // 2 (head starts here)

H = # head colors (the head's internal states)



C = # tape colors (the cell alphabet)

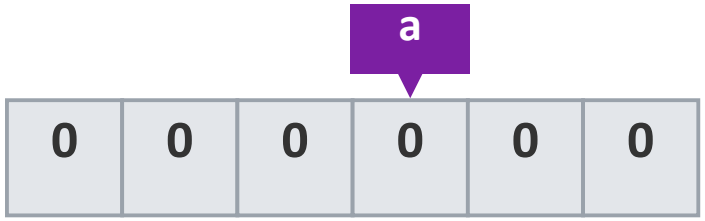
the tape wraps: cell 0's left neighbor is the last cell

every cell starts at color 0 • head starts at color 0

One step: fire a transition-table row

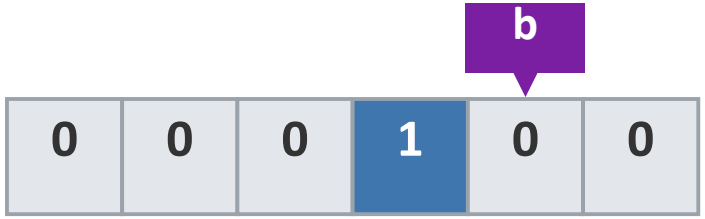
Each row is keyed by (head color, tape color under the head). Find the matching row; do exactly what it says — write, recolor, move.

before



match (head_color=a, read=0) → write 1, head→b, move +1

after

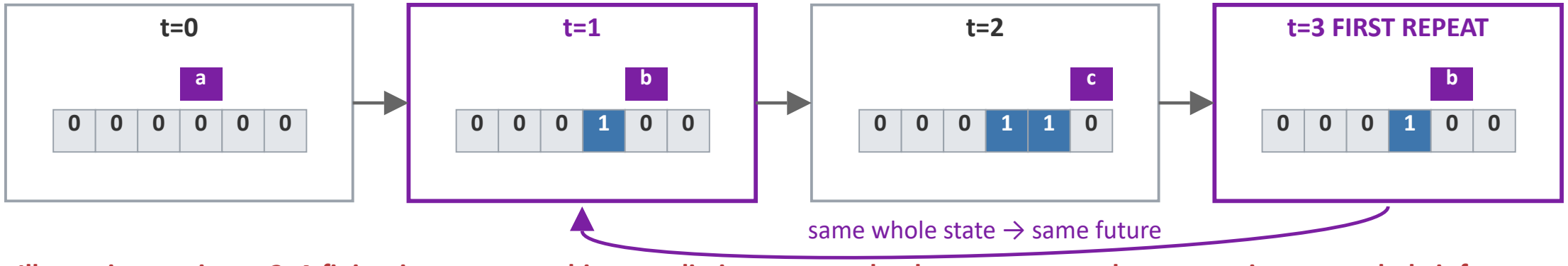


Configuration, and when it repeats

A configuration is the WHOLE machine state at an instant: (head position, head color, the contents of every tape cell).

Runtime is the first-repeat step count (SPEC §4)

```
config = (pos, head_color, tuple(tape))   # the entire state
# runtime = the first step t at which `config`
#           equals some configuration seen earlier
```



Illustrative runtime = 3. A finite ring repeats or hits step_limit; no repeat by the cap means unknown runtime, so exclude it from scoring.

Hard part 1: the state space

You cannot simulate a deep machine to its repeat — there is not enough matter in the universe to store the configurations it would visit.

The size of the configuration space

```
# configurations  ≈   C ** tape_size   ×   tape_size   ×   H
#                     └ cell colorings ┘   └ head pos ┘   └ head color ┘
```

$\sim 10^{79}$

configs: 2 colors, 256 cells

$\sim 10^{80}$

atoms in the observable universe

$\sim 10^{154}$

configs: 4 colors, 256 cells

Hard part 2: busy beavers

A tiny table can run staggeringly long before it repeats — and nothing you can read off the table tells you how long.

Classic busy-beaver step counts (2-symbol machines, Radó 1962; 5-state proved 2024). BB(6) is unknown — provably beyond 10↑↑15.

Runtime scales with tape size: the same table takes 40 steps on 6 cells, 8,777 on 64.

The consequence: measure the shallow, estimate the deep

SHALLOW

Config repeats after a few hundred steps.

You can run it to the repeat and read the exact runtime off the clock.

Cheap to MEASURE.

DEEP

Will not repeat within any feasible budget—not with any amount of memory.

You never reach its runtime. You must put a number on it anyway.

Must be ESTIMATED.

A rack mixes both. Ranking it under one tight budget — which to run, which to estimate — is exactly what tm-compute grades (up next today).

Determinism: the table plus the start fixes everything

There is no randomness anywhere. Given the transition table and the initial configuration, the whole trajectory — and therefore the runtime — is fixed. Run the same machine twice, get the identical step count.

One step — no clock, no entropy, no I/O

```
def step(tape, pos, head, table, n):
    head_out, cell_out, move = table[(head, tape[pos])]
    tape[pos] = cell_out            # write the current cell
    head = head_out                 # change the head color
    pos = (pos + move) % n           # move, wrapping on the ring
    return tape, pos, head
```

So runtime is a property of the machine, not of luck — which is the only reason a rack HAS a true ranking to recover.

Key Takeaways

1. tm-compute's model: a head with H colors on a ring of C-colored cells, starting at the center — runtime is the first step the full configuration repeats
2. Ranking is hard for two reasons: the configuration space is astronomical, and the busy-beaver phenomenon means the table hides the runtime
3. So some machines you MEASURE and some you must ESTIMATE — up next today, the assignment that makes you rank a rack of both under a tight budget

hand-run a machine to its first repeat

10 min

- Machine (2 head colors a,b; 2 tape colors 0,1) on a 4-cell RING. Head starts on the center cell (cell 2), state a, all cells 0.
- Table: (a,0)→write1,head→b,+1 (a,1)→write0,head→b,−1
- (b,0)→write1,head→a,−1 (b,1)→write0,head→a,+1
- In pairs: hand-step it, writing the full config (pos, state, tape) after each step. Stop when a configuration REPEATS one you've seen.
- What is the runtime (the first-repeat step count)? Does it return to the START config, or to an intermediate one — and why does that distinction matter for a bot counting steps?
- Compare with another pair. If your runtimes differ, locate the first different configuration and resolve it before reporting.