tm-compute

Key takeaways

1. Seven hidden machines share one 250 ms, 64 MB evaluation budget.
2. Exact execution exhausts the budget; allocate measurement selectively.
3. You own the allocation and the estimator; the agent writes the plumbing.

Common misunderstandings of runtime ranking under a budget

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

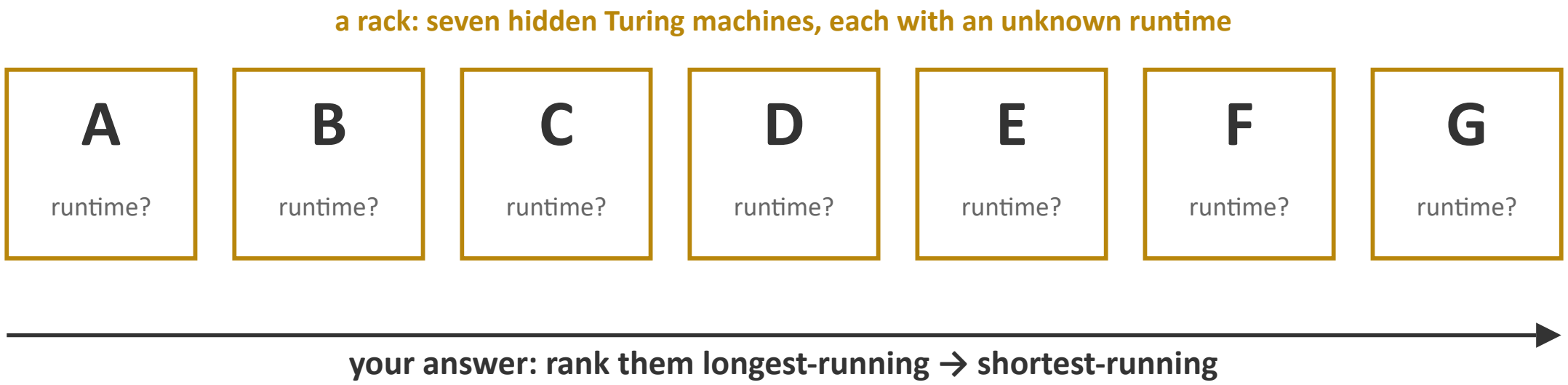
"Each of the seven hidden machines receives its own 250 ms and 64 MB."

"Running every machine until it repeats gives the best feasible ranking."

"A bot must predict exact runtimes to rank the hidden machines well."

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

The game: rank a rack

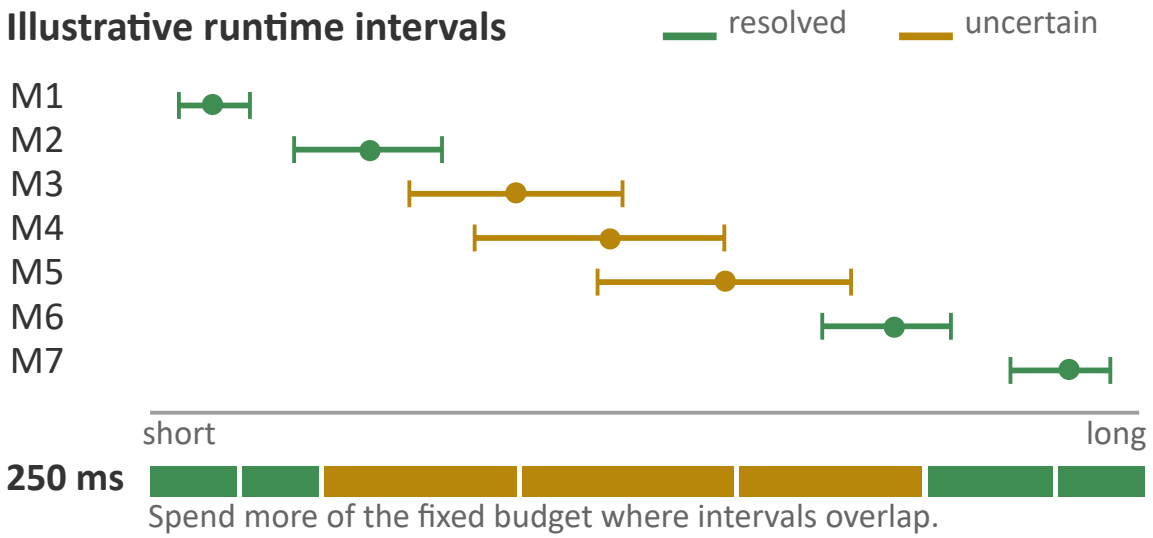


scored on PAIRWISE accuracy: for every pair whose runtimes differ, did you put the longer-running machine first?

Ranking is an allocation problem

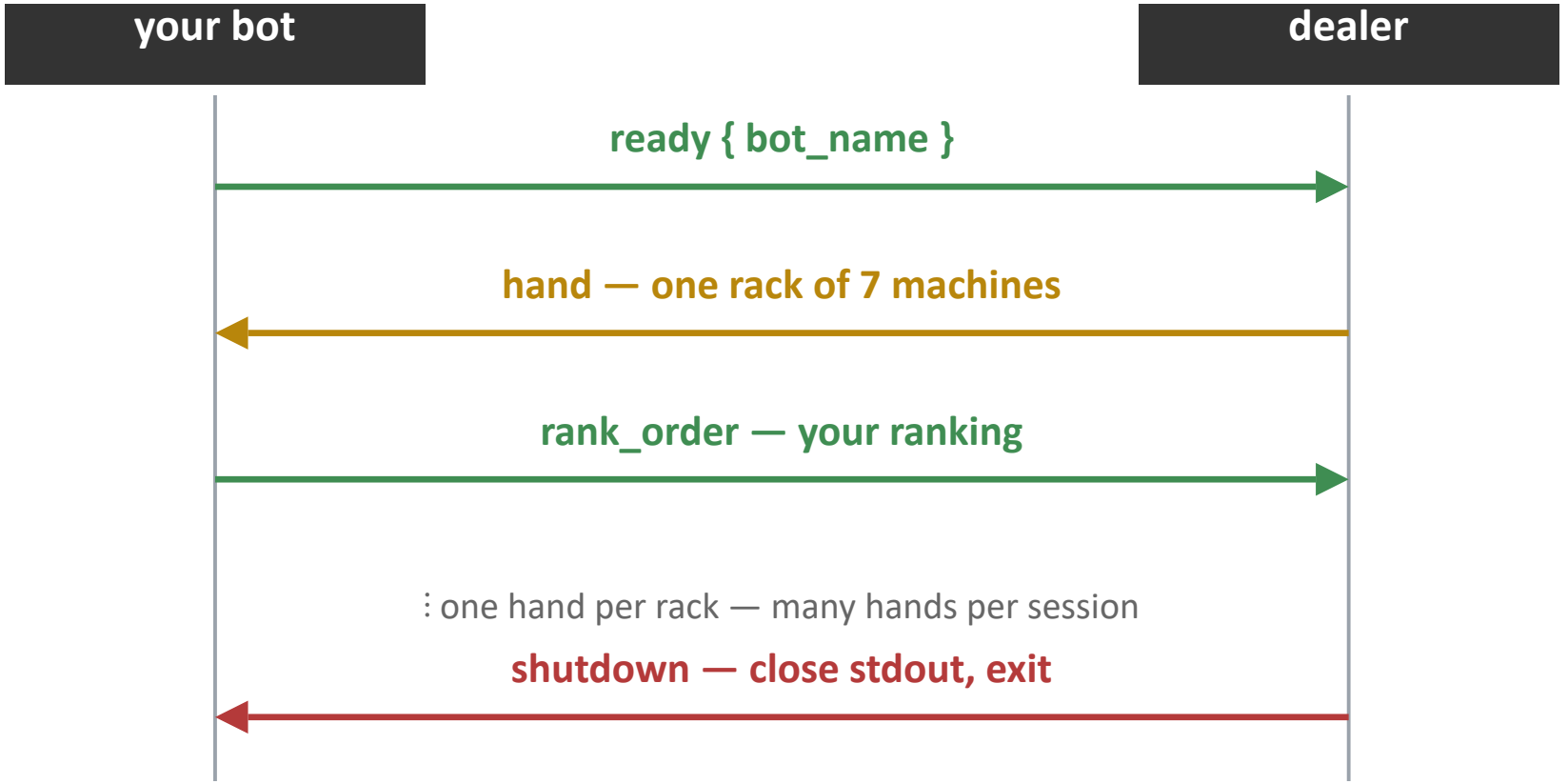
Rank seven hidden machines by runtime — under one tight budget

- Project 2: write a bot.
- The catch is the budget — 250 ms and 64 MB per rack — which makes “just run them all” impossible
- The graded skill is allocation: which machines to execute, which to estimate, and how to split the clock across seven



Your bot: a persistent process on stdio

You write one long-lived process that speaks tmc.v1 — JSON, one object per line. It handles the whole session without restarting.



The budget is the assignment

Two hard limits, applied to the whole rack of seven TOGETHER — not to each machine alone. Enforced by cgroup v2 on the Linux grading host; advisory on a local macOS run.

250 ms

wall time, per rack

64 MB

memory, per bot process

7

machines share that one budget

The budget is not a coincidence — it is the central problem. A slow laptop simply completes fewer steps than the grading host.

Your first instinct dies on a rack

Your first instinct — run every machine to its repeat, then rank by measured steps — works on shallow machines and DIES on a rack.

MEMORY

To detect a repeat you must remember every configuration seen. That record grows with every step — so a deep machine exhausts the 64 MB cap and is KILLED before it ever repeats.

TIME

The deepest machines never repeat within 250 ms — not with any amount of memory. Every millisecond spent driving a doomed machine is STOLEN from the ones that would have finished.

So the graded skill: which machines to execute and how far, which to estimate without finishing, and how to divide the wall across seven.

Scoring: count the correctly ordered pairs

You are scored one pair at a time, over the gradeable machines only.

A hand with more scorable pairs carries more weight

```
normalized = correct_pairs / total_scorable_pairs   # per hand, [0,1]
aggregate  = Σ correct_pairs / Σ total_scorable_pairs  # pair-weighted
```

- Pairs involving a step_limit machine (runtime unknown) are skipped entirely — so is any pair whose two runtimes are equal
- A non-permutation reply forfeits the whole hand: 0 correct pairs
- A crash contributes 0 over its scorable-pair denominator — a failed hand is never dropped, so a crash cannot inflate your score

Ground truth: A > B > C

bot reply	correct pairs	score
A > B > C	A > B, A > C, B > C (3)	3 / 3
A > C > B	A > C, A > B (2)	2 / 3
B > A > C	A > C, B > C (2)	2 / 3
B > C > A	B > C (1)	1 / 3
C > A > B	A > B (1)	1 / 3
C > B > A	none (0)	0 / 3

The grading host is authoritative, not your laptop

Local wall time is a conservative proxy; the grading host (chambers) is authoritative for throughput. If it passes locally it should pass there — but the two hosts are not equivalent.

- The timing axis is a boundary-rider, so chambers grades the K=3 median of whole graded sessions — one run of a boundary build is not trustworthy
- This is the same lesson as the W7 measurement-honesty deck: trust the ordering and the ratio, not an absolute step count or a laptop millisecond
- Validate the memory lever on a cgroup-v2 Linux host — macOS only reports an advisory peak RSS, it does not enforce the 64 MB cap

You vs. the Agent: tm-compute

Your Responsibility	Agent's Responsibility
The execute-vs-estimate allocation — the graded decision	The tmc.v1 stdio plumbing: read hands, emit rank_order
The estimator: a signal for runtimes you never measure	The stepper: one transition, wraparound, config-repeat check
The budget split across seven; the pre-registered prediction	The local-runner glue and the failure-category readout
Authored racks + ground truth for the PR round	Minimizing a failing rack to a small repro

Shared: the agent can build any estimator — only you choose which machines to trust the clock on

Key Takeaways

1. tm-compute: rank a rack of 7 hidden machines by runtime under 250 ms × 64 MB per rack, scored on pairwise accuracy over the gradeable machines
2. The budget IS the assignment: exact execution dies on memory and time, so deciding what to run, what to estimate, and how to split the wall is the graded skill
3. You own the allocation, the estimator, and a writeup with a pre-registered prediction; the agent writes the plumbing. Then the PR round: improve a peer's bot, defend your own