Reasoning in Large Language Models

From chain-of-thought to search

Part 1 of 2 · Anji Liu

Tutorial on Agentic Reasoning · NUS-MSRA Summer School: Agentic AI

About Me

Anji Liu

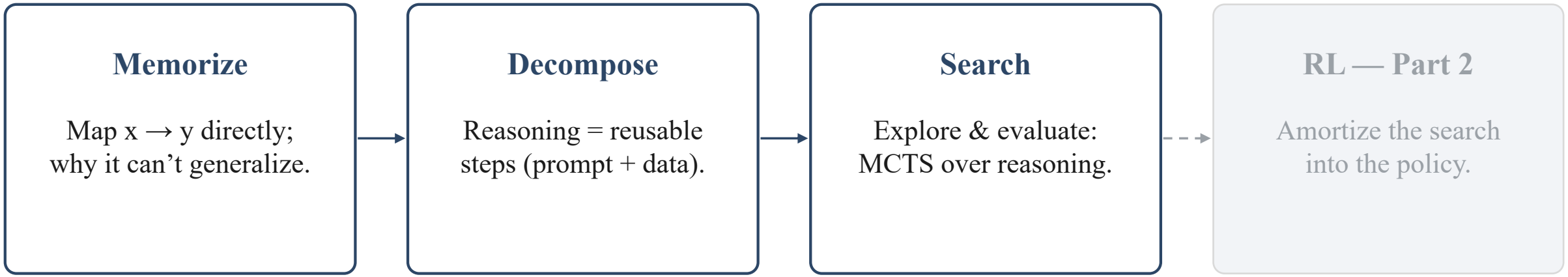
Assistant Professor (PYP)
Department of Computer Science
National University of Singapore

Research interests: Generative AI, tractable reasoning, neurosymbolic AI

I design generative AI models that can do sound reasoning, aiming towards better controllability, reliability, and trustworthiness.

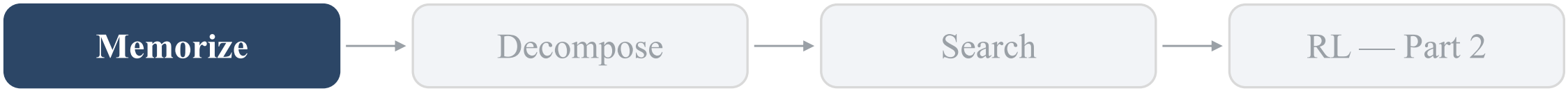
Roadmap

Key idea: **memorizing is brittle, decomposing generalizes, search makes it strong, and RL makes it cheap.**



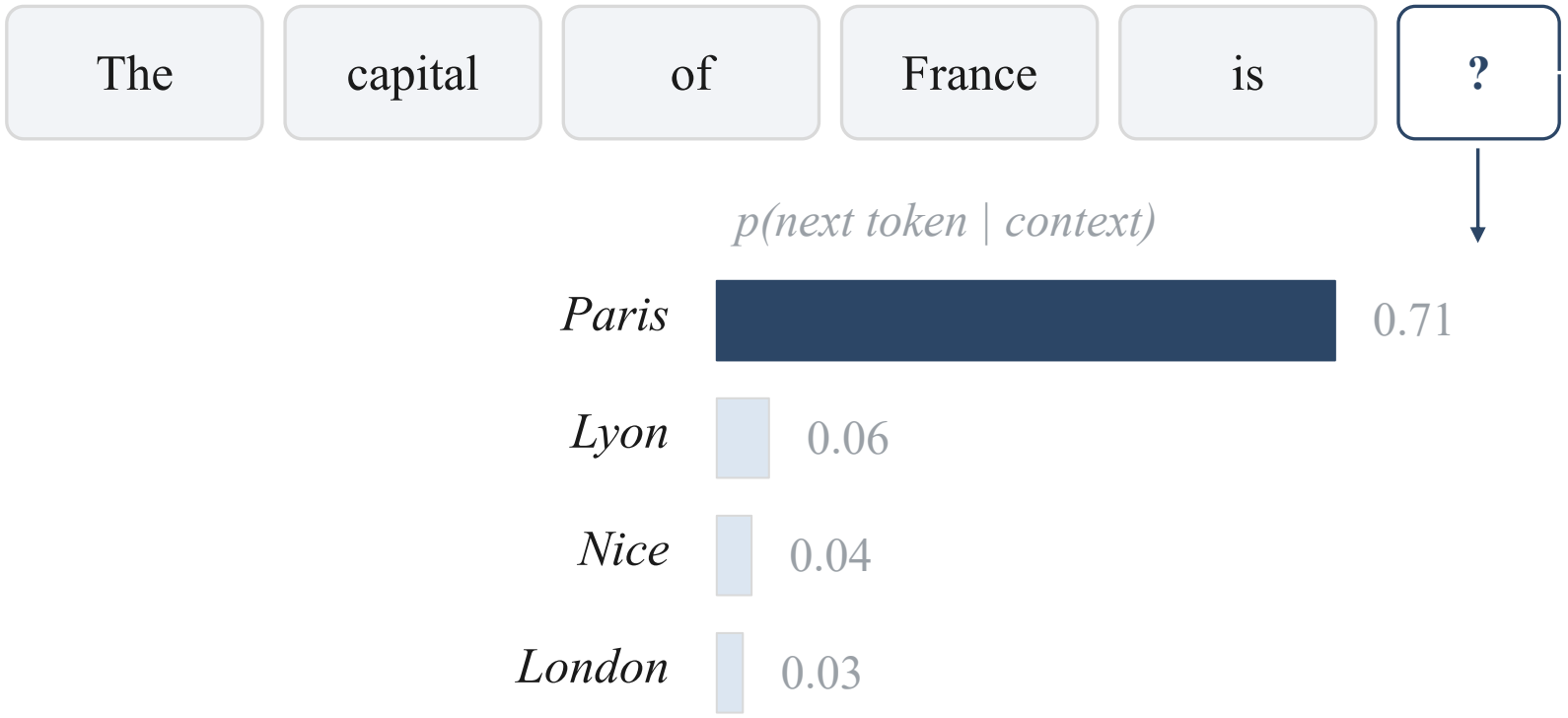
Background

How a language model works



Large language models

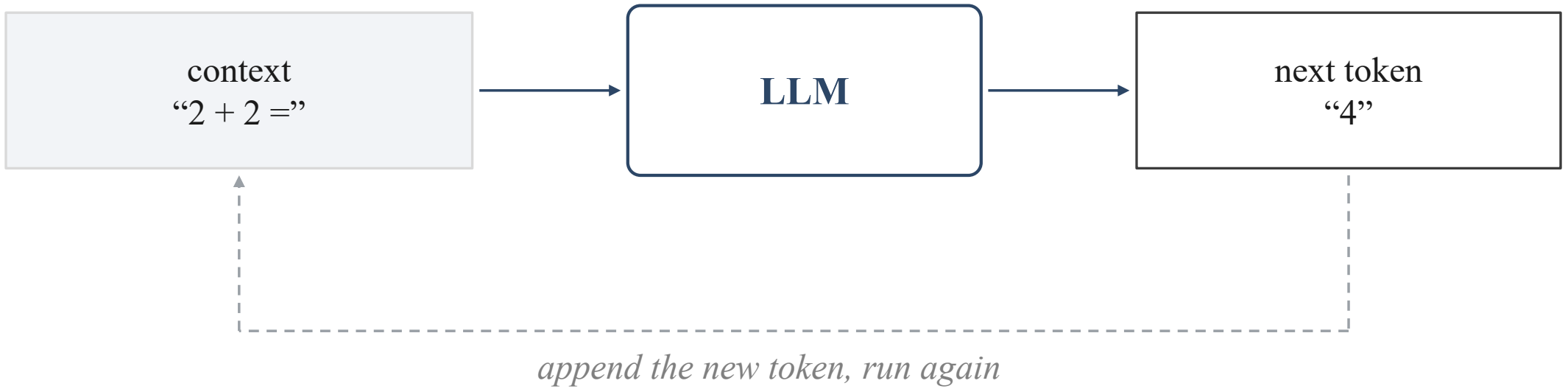
An LLM only ever does one thing: **predict the next token** given all the previous ones.



Everything else such as "knowledge" and "reasoning" has to emerge from this one operation.

How it answers a question

A question is just a prefix. The answer is tokens sampled **one at a time, each fed back in.**



Repeat until the model emits a stop token. The output is grown left-to-right.

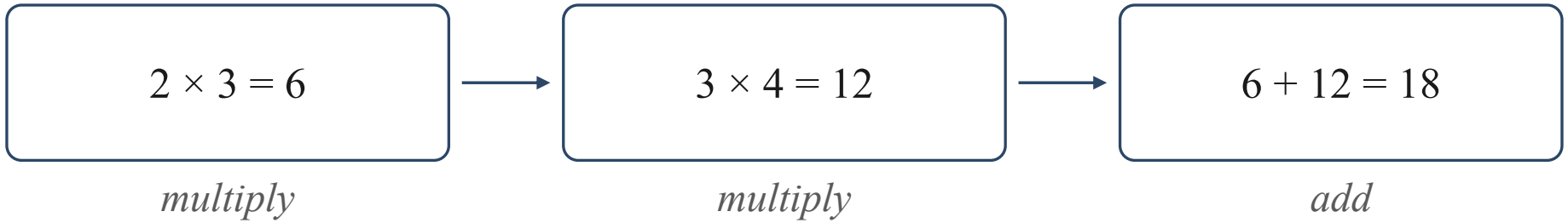
Why answering directly doesn’t generalize

Memorize → Decompose → Search → RL — Part 2

A question that hides a procedure

Pens cost $3, notebooks $4. Buy 2 pens and 3 notebooks — **total?**

$18



We answer instantly — but the answer is the output of a little three-step program, not a fact we recall.

Change a number, the answer changes

Swap “2 pens” for “5 pens”. New total **$27**. Still trivial — we can list a few of these.

input	answer
2 pens, 3 notebooks	**$18**
5 pens, 3 notebooks	**$27**
2 pens, 6 notebooks	**$30**
…	…

“just memorize the table”

Easy — because we understand the words. But the model doesn’t read words; it reads tokens. What does this look like to it?

Now put yourself in the model's shoes

These two questions differ by **a single token**. You can't see what changed — but the answer did. That is the model's view of *every* question.

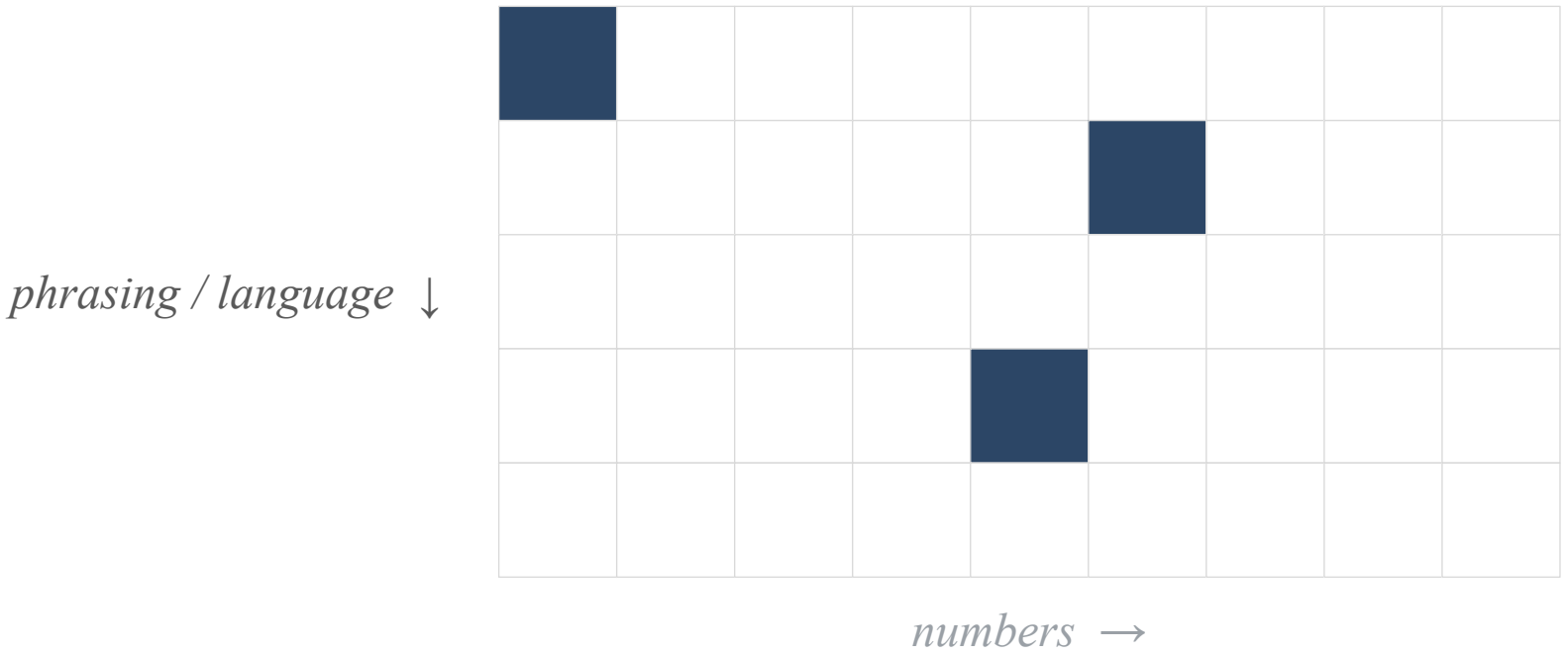
Q1 ብዕር ድ ብር፣ ደብተር ፬ ብር። ፪ ብዕር፣ ድ ደብተር — ድምር? → **18**

Q2 ብዕር ድ ብር፣ ደብተር ፬ ብር። ፭ ብዕር፣ ድ ደብተር — ድምር? → **27**

One token → a different answer, with no understanding of the steps behind it. So it can only memorize the mapping — it can't generalize.

Direct answering = an impossible lookup table

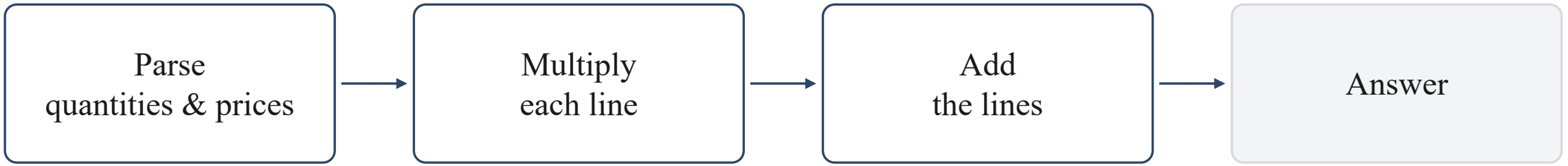
One entry for every **(phrasing × language × numbers)**. That grid is exponential and mostly unseen.



filled = **seen in training**. Everything white must be *generalized* — and a memorizer can't.

The way out: tasks are built from reusable steps

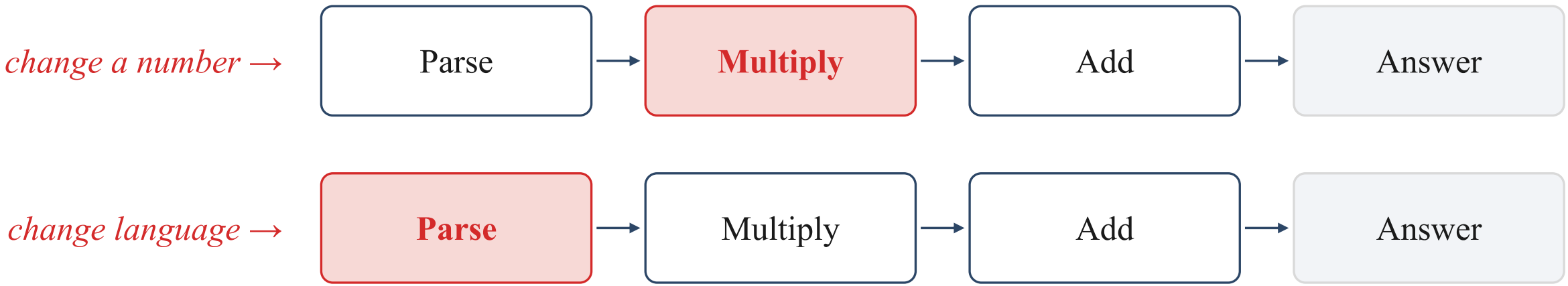
Most problems are **compositions of a few simple sub-procedures** you already know.



Each box is a small operation reused across countless problems. Reasoning is stringing them together.

Decomposition localizes the change

Now a surface change touches **one step**; the operations are reused — you can even *guess the pattern* in a new language.



"Multiply" and "Add" never change — they are shared across every version of the problem.

This is generalization: reuse the invariant steps, adapt only the one that must change.

Definition

What is “reasoning”?

Reasoning, defined

Reasoning = inserting **intermediate steps z** that compose reusable operations, instead of jumping $x \rightarrow y$.

$$x \longrightarrow z_1 \rightarrow z_2 \rightarrow \cdots \rightarrow z_T \longrightarrow y$$

$$p(y \mid x) = \sum_z p(z \mid x)\, p(y \mid x, z)$$

sum over the possible reasoning traces z

x = question y = answer z = the steps

What the steps buy us

Three things a direct answer can never give you:

1

Generalization

Reuse invariant steps; adapt only what changes.

2

Interpretability

You can read the trace and see why.

3

Error localization

A wrong answer has a wrong step — findable, fixable.

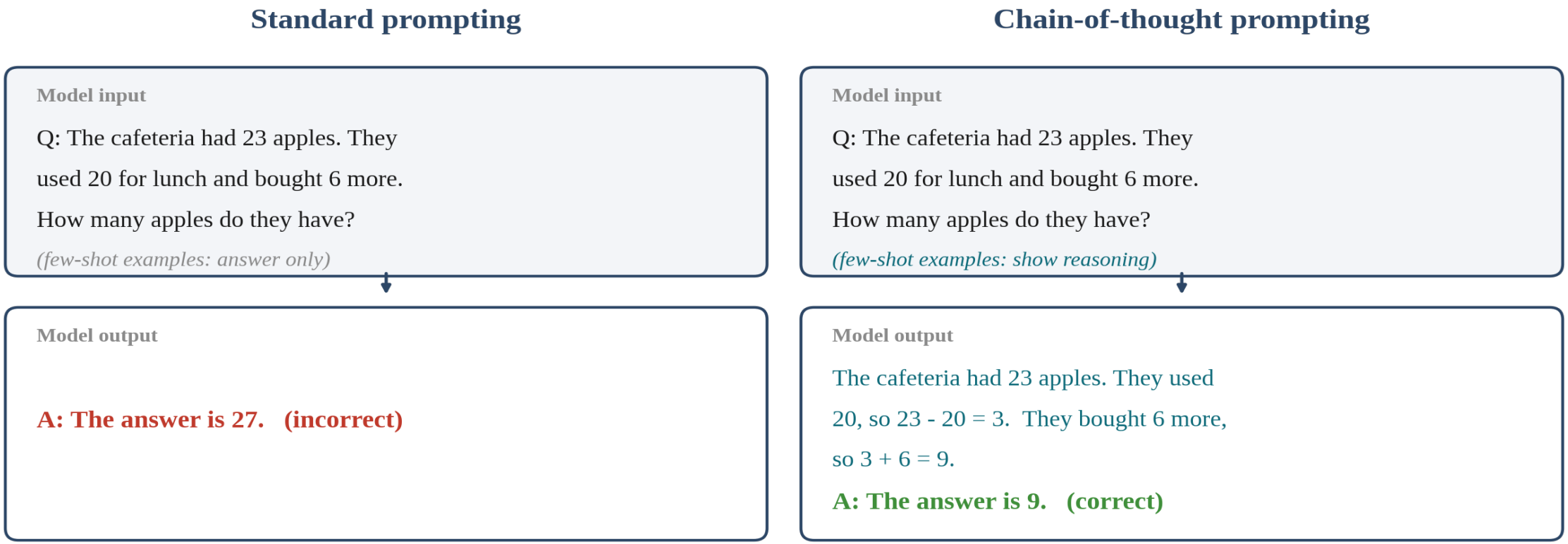
Hold on to #3 — "localize the error" — it is the crack that search and RL will pry open.

Making models reason · 1

Prompting and data

Chain-of-thought: just ask for the steps

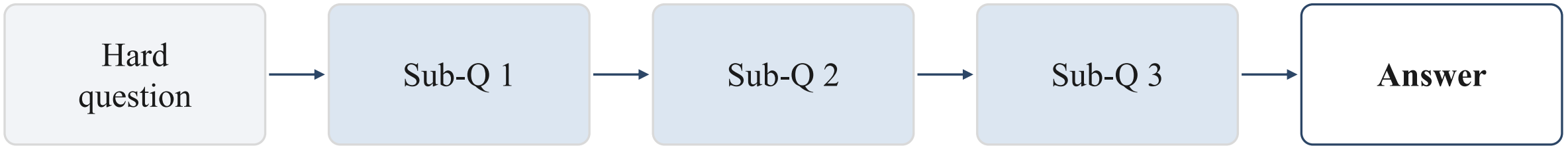
Prompt the model to **show its work**, and latent reasoning it already has comes out.



[Chain-of-thought prompting: Wei et al., 2022 · zero-shot "let's think step by step": Kojima et al., 2022 · figure after Wei et al., 2022]

Make the decomposition explicit

Least-to-most: solve easy sub-problems first, feed them forward — and it generalizes to harder instances.



Decomposition done in the prompt rather than left to chance — the C5/C6 pipeline, made operational.

[Least-to-most prompting: Zhou et al., 2023]

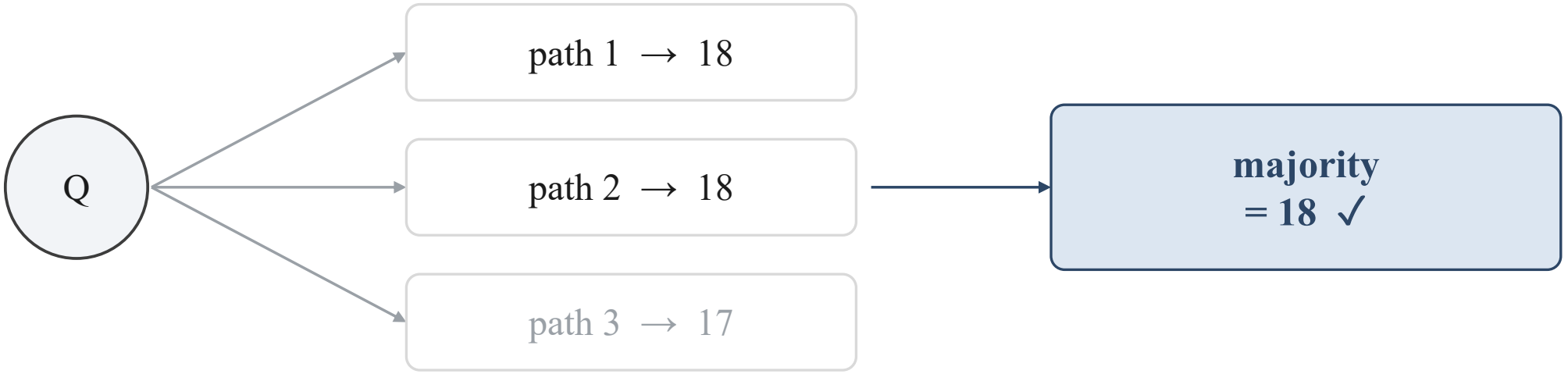
The same question, different answers

Generation is sampling: run it twice and you can get different outputs.
Fluent and confident ≠ correct.

A single sample is a coin flip. Hold this thought — it is exactly what "self-consistency" will exploit later.

Don’t trust one chain — sample many and vote

Self-consistency: sample several reasoning paths, then take the **majority answer.**

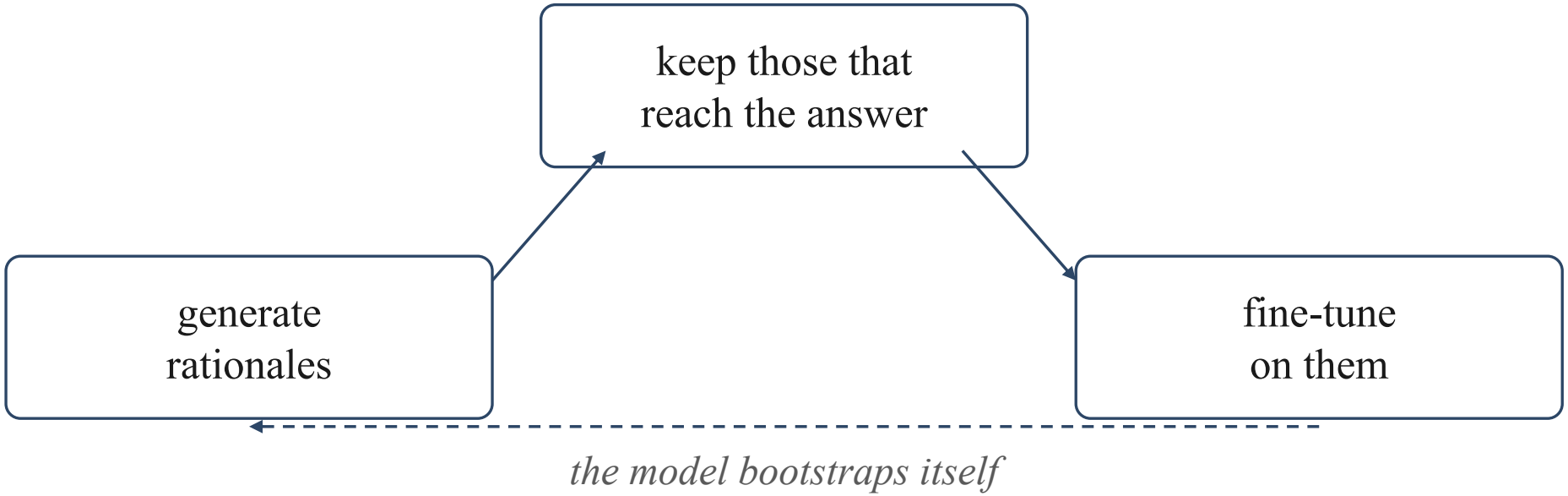


Reasoning paths that reach the truth agree; wrong ones scatter. Voting turns a coin-flip into a reliable answer.

[Self-consistency: Wang et al., 2023]

You can train reasoning in, not just prompt it

STaR: let the model generate rationales, **keep the ones that reach the right answer**, fine-tune on them, repeat.



A self-improvement loop — no human-written reasoning needed, only a correctness check at the end.

[STaR — self-taught reasoner: Zelikman et al., 2022]

Where prompting and data run out

These get a decent reasoner. But when a chain goes wrong, **we can't recover it** — and fixes are ad hoc.

one bad step derails everything after it (no eraser)

slow, ad hoc, and it doesn't scale

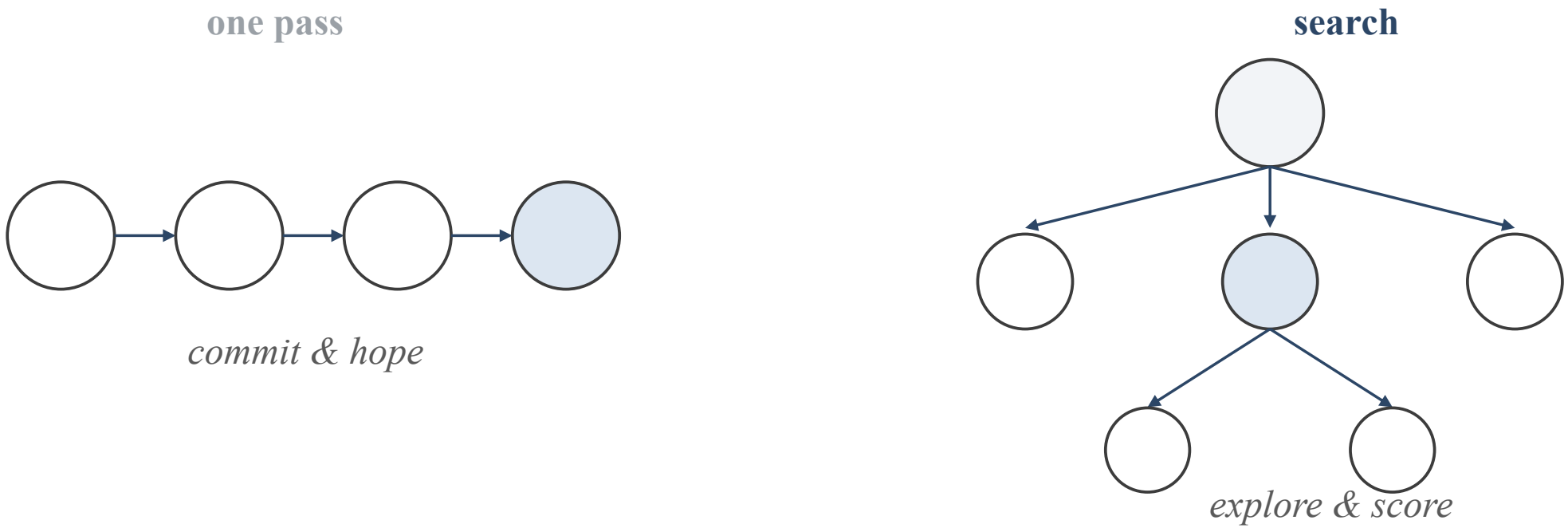
[e.g. error-analysis → targeted data collection; a manual loop]

Making models reason · 2

Search

The missing ingredient: explore, then evaluate

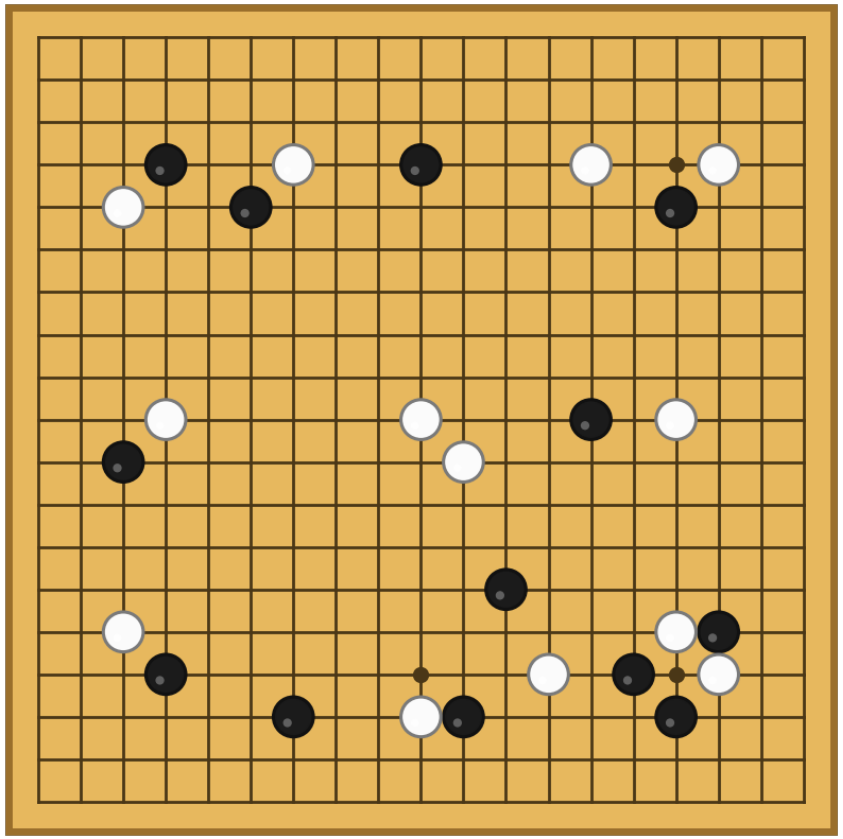
One greedy pass commits to a single chain. We want to **try alternatives and judge them** — that's search.



Same model, but we spend inference compute exploring a tree instead of one line.

Why Go needs smart search

The game tree is astronomically large — you **cannot enumerate it**. You must search selectively.



$$\approx 10^{170}$$

legal positions — more than atoms in the observable universe

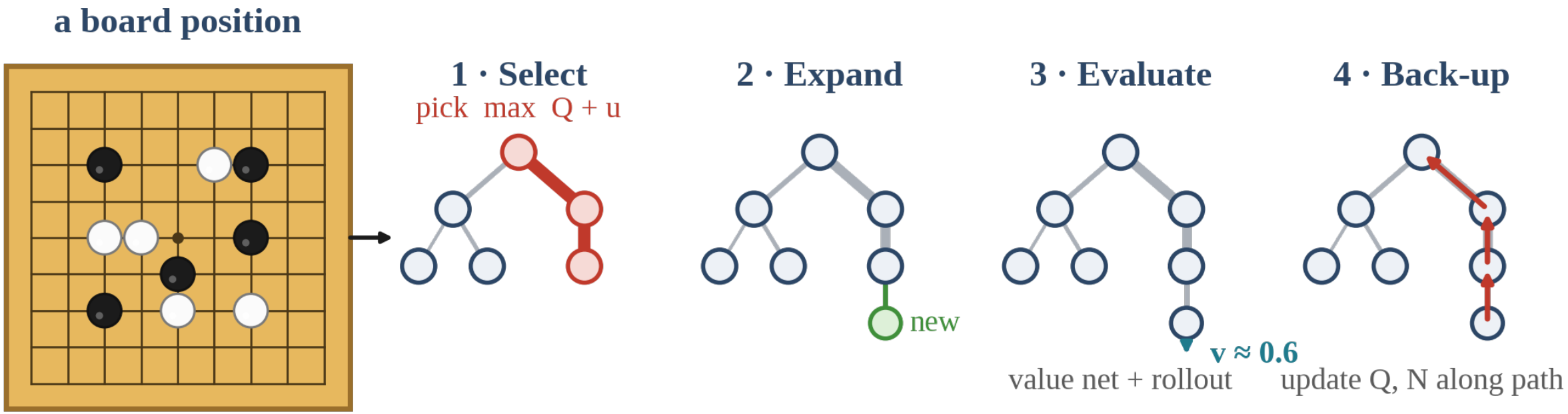
Brute-force lookahead is hopeless and a hand-written evaluator is weak — so how did a machine beat the world's best?

AlphaGo defeated Lee Sedol 4–1 · March 2016

Its engine — Monte-Carlo Tree Search — is the tool we now point at reasoning.

Monte-Carlo Tree Search, in four repeated steps

Spend a fixed budget wisely: grow the tree only where it looks promising.



Each node is a board position; thousands of these cycles concentrate compute on the strongest lines — no full enumeration. (edge width ∝ visit count)

Selection balances exploit vs. explore

Prefer high value, but keep probing the **under-tried** — that is the UCB rule.

$$\text{score} = \textbf{average value} + c \cdot \sqrt{(\ln N / n)}$$

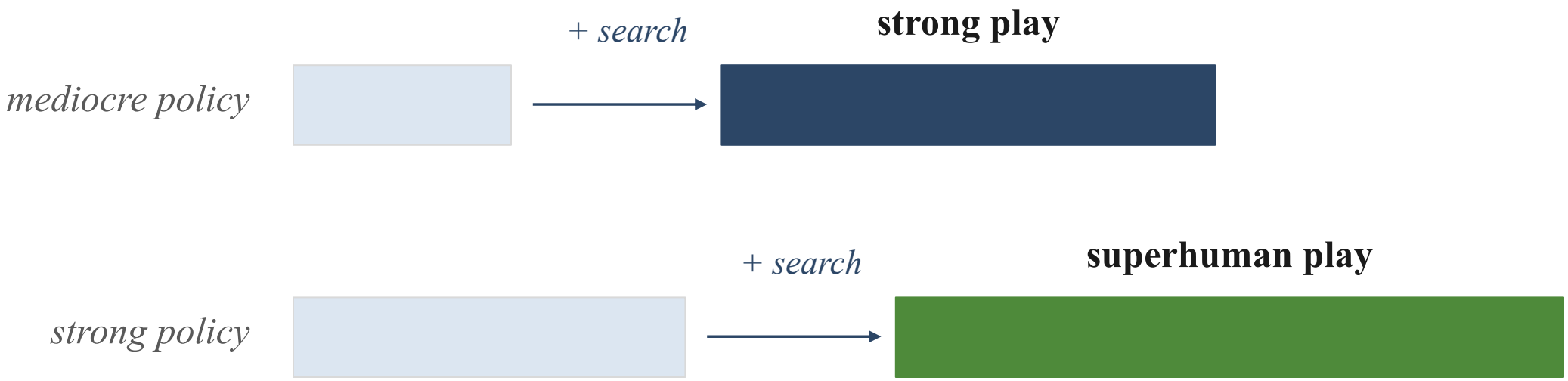
exploit what looks good — *explore the rarely-visited*

exploit picks this — *explore won't ignore this*

N = visits to parent n = visits to this child c = exploration weight

Why search matters: it amplifies the policy

Search turns a **mediocre policy into a strong one**, and a strong one into superhuman play.



Same network, more thinking at play-time → a categorically better player.

[AlphaGo — Silver et al., 2016]

Search over chains of thought

Memorize → Decompose → **Search** → RL — Part 2

Objection: Go is well-defined, language is not

Search needs states, moves, and a score. Go hands them to you; language seems not to.

	Go (well-defined)	Language (free-form)
State	a board position	*a half-finished answer?*
Moves	legal stones	*any next sentence?*
Score	win / loss	*is this reasoning good?*

If we can't define the game, we can't search it. So — can we define it?

Resolution: define the game with the model itself

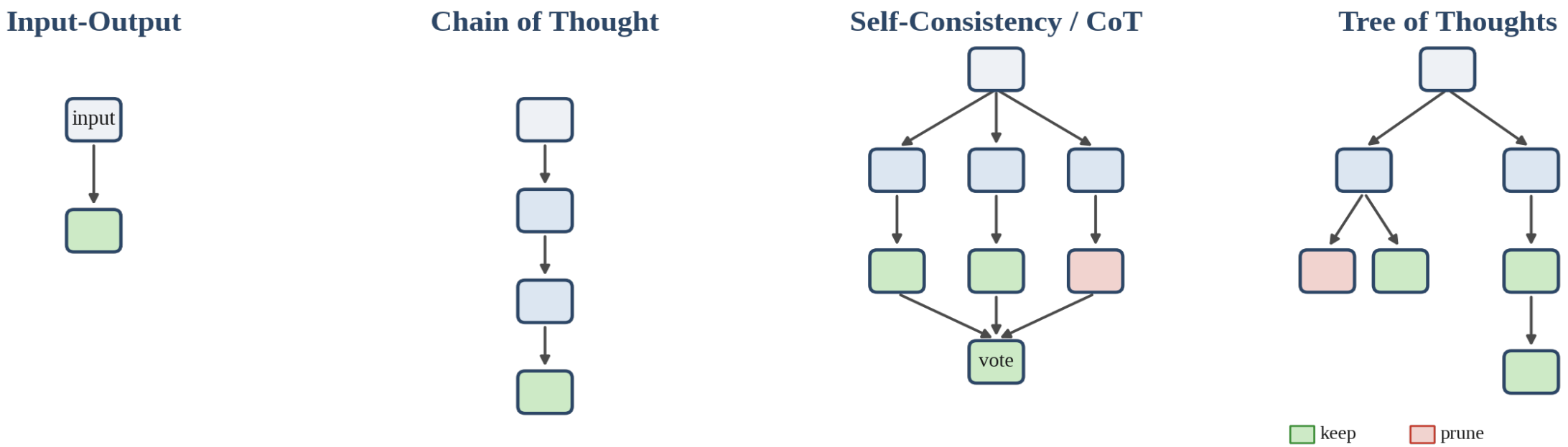
A **step** is a move; the **LLM proposes and scores**; a verifier judges the end.

a move	→	a reasoning step (a “thought”)
the policy	→	the LLM proposes candidate next steps
the value	→	the LLM / a verifier scores a partial solution
win / loss	→	final answer checked or scored

The model plays both sides — generating moves and evaluating them — so MCTS has everything it needs.

Tree of Thoughts: make the chain a searchable tree

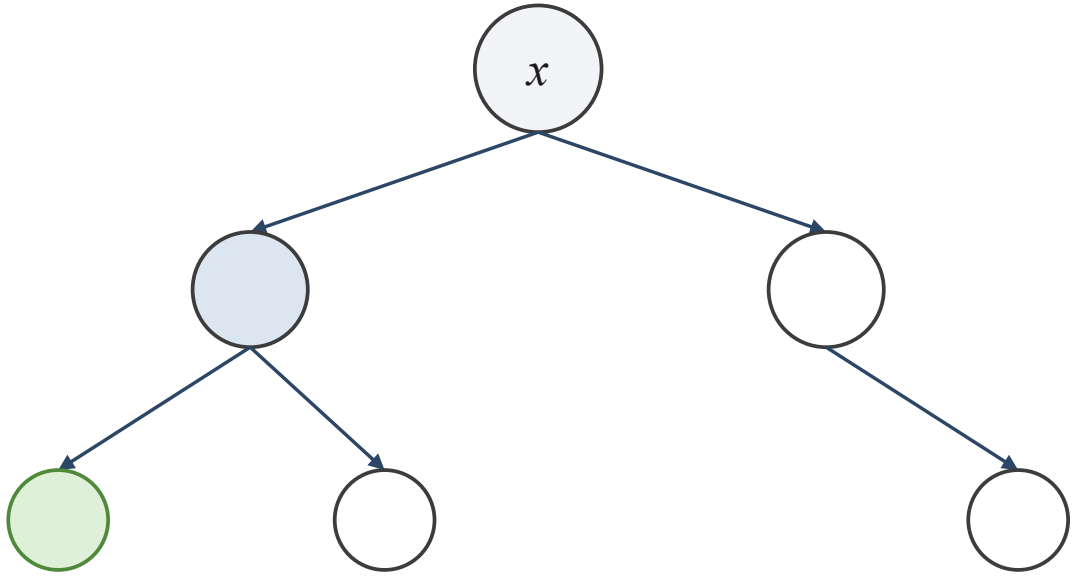
Instead of one chain, branch into **alternative thoughts**, evaluate them, and search — keeping the best.



[Progression of prompting/search schemes — after Tree of Thoughts: Yao et al., 2023]

Run MCTS over thoughts

The LLM is **both policy and evaluator**; MCTS grows the reasoning tree toward promising solutions.



value scores guide where to expand

LLM proposes next steps
(policy)

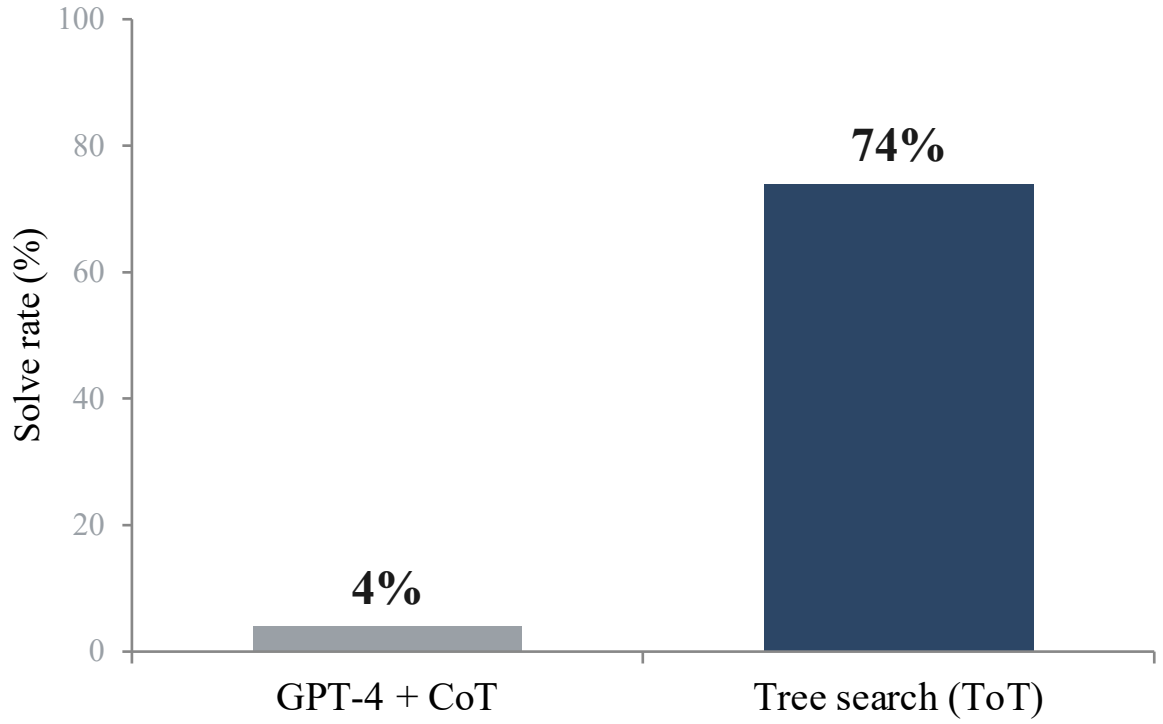
LLM / verifier scores
partial solutions (value)

search + evaluate = a much stronger reasoner on hard problems

[RAP (reasoning as planning): Hao et al., 2023 · ReST-MCTS: Zhang et al., 2024]*

And it works: search rescues problems CoT fails

Game of 24: GPT-4 with chain-of-thought solves **4%**; searching the tree of thoughts reaches **74%.**



[Tree of Thoughts, GPT-4, Game of 24: Yao et al., 2023]

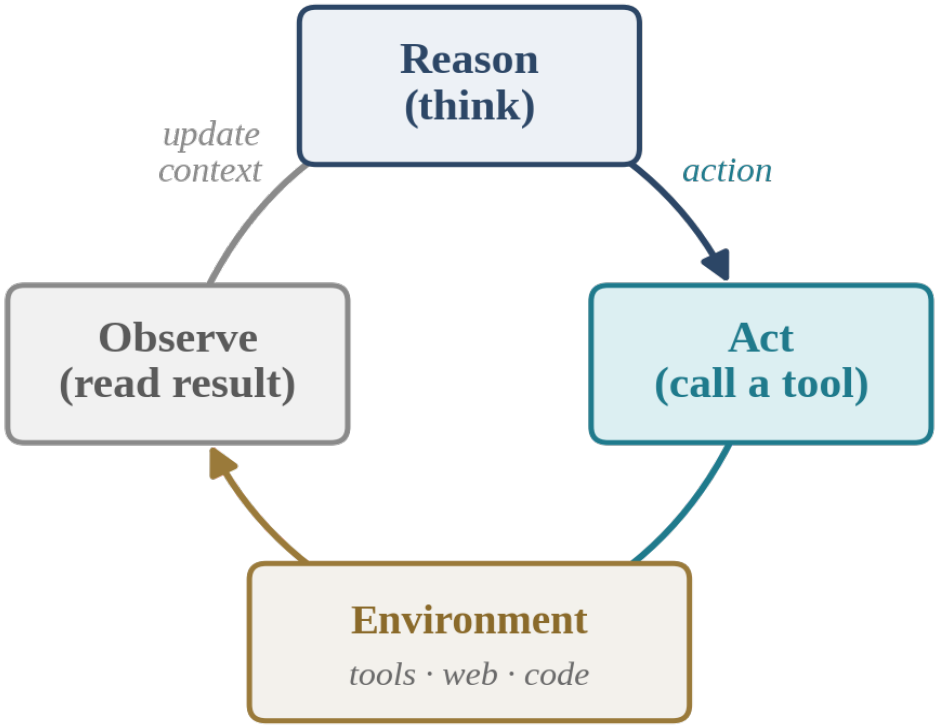
The topic: agentic AI

From reasoning to acting

so far the model only thinks — an agent also acts, in an environment

Reasoning was half the story — agents also act

Everything so far assumed a **closed question**. Real tasks need the model to **act and observe**, in a loop.



An agent = a reasoning LLM in a loop with:

Tools / actions — call a search engine, run code, hit an API

Observations — feed the results back into context

Memory — carry state across steps

Planning — break a goal into sub-tasks

The reasoning we built is the engine; acting in an environment is the new axis.

ReAct: interleave thinking and acting

Chain-of-thought, but with an environment in the loop: **Thought → Action → Observation**, repeated.

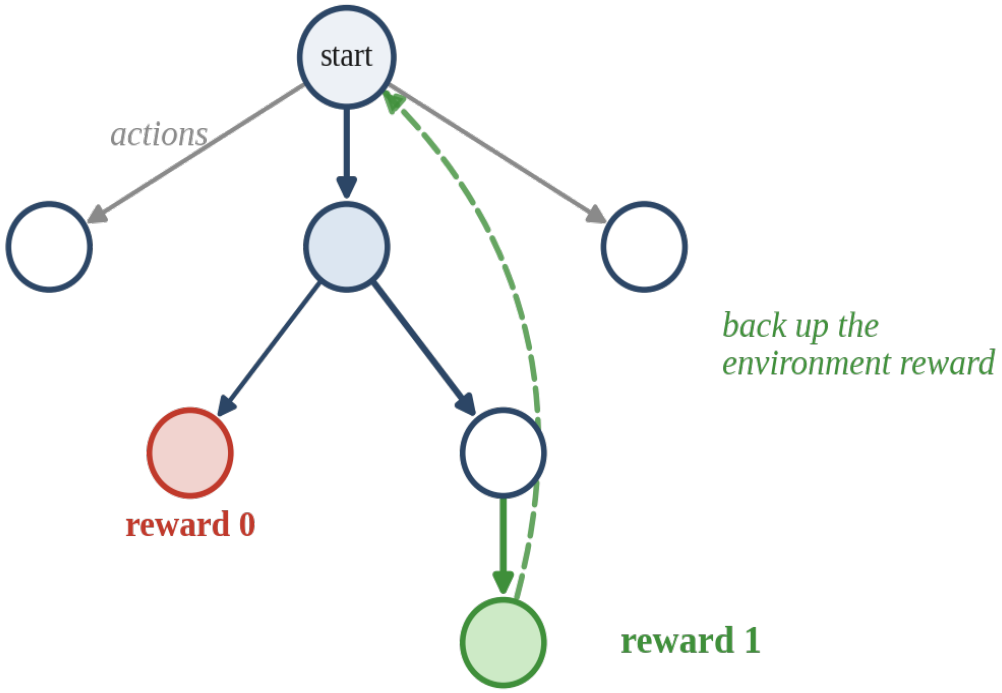
Q	Pens $3, notebooks $4. Buy 7 pens and 12 notebooks — total?
Thought	multiply each, then add — I'll use a calculator.
Action	`calc("7 * 3")`
Observation	21
Thought	now the notebooks, then the sum.
Action	`calc("12 * 4 + 21")`
Observation	69
Answer	**$69**

Same move as chain-of-thought — but each step can call a tool and read the result, so the model offloads what it can't reliably do in its head.

[ReAct: Yao et al., 2022 (ICLR 2023) — the same author as Tree of Thoughts]

Search over thoughts → search over trajectories

The same MCTS tree — but each node is a **(thought, action, observation)** step, and the value is a **real environment reward**.



a move → an action (a tool call)

the value → did the task succeed? (reward)

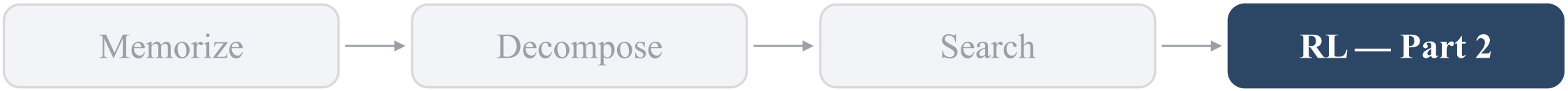
same loop → select · expand · evaluate · back-up

…and, like reasoning, you can amortize this search — RL over trajectories (next).

[LATS — Language Agent Tree Search: Zhou et al., 2024 (ICML) · RAP: Hao et al., 2023]

The catch

Search is powerful — but slow

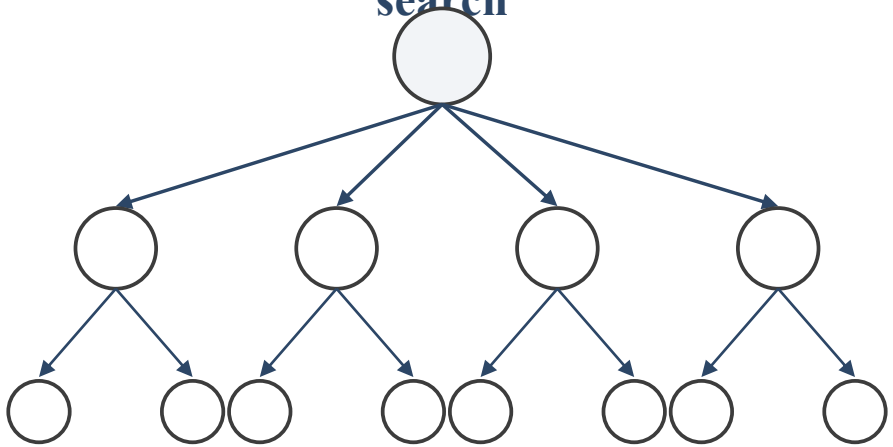


The catch: search is expensive

Every answer now costs **many model calls** — search is slow and pricey at inference time.

direct

search

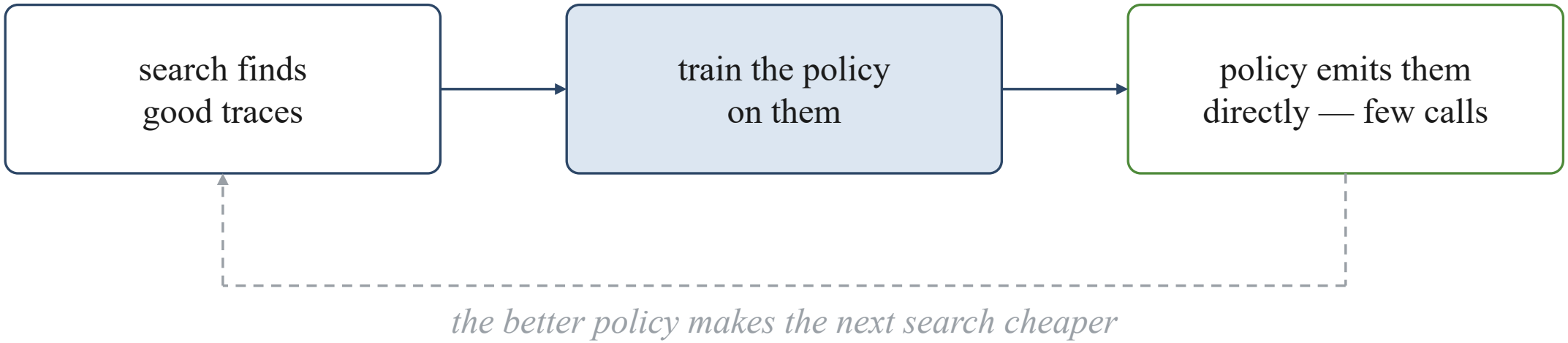


dozens–thousands of calls per answer

We paid compute at test time to get quality. Can we get the quality without paying every single time?

The idea: amortize the search

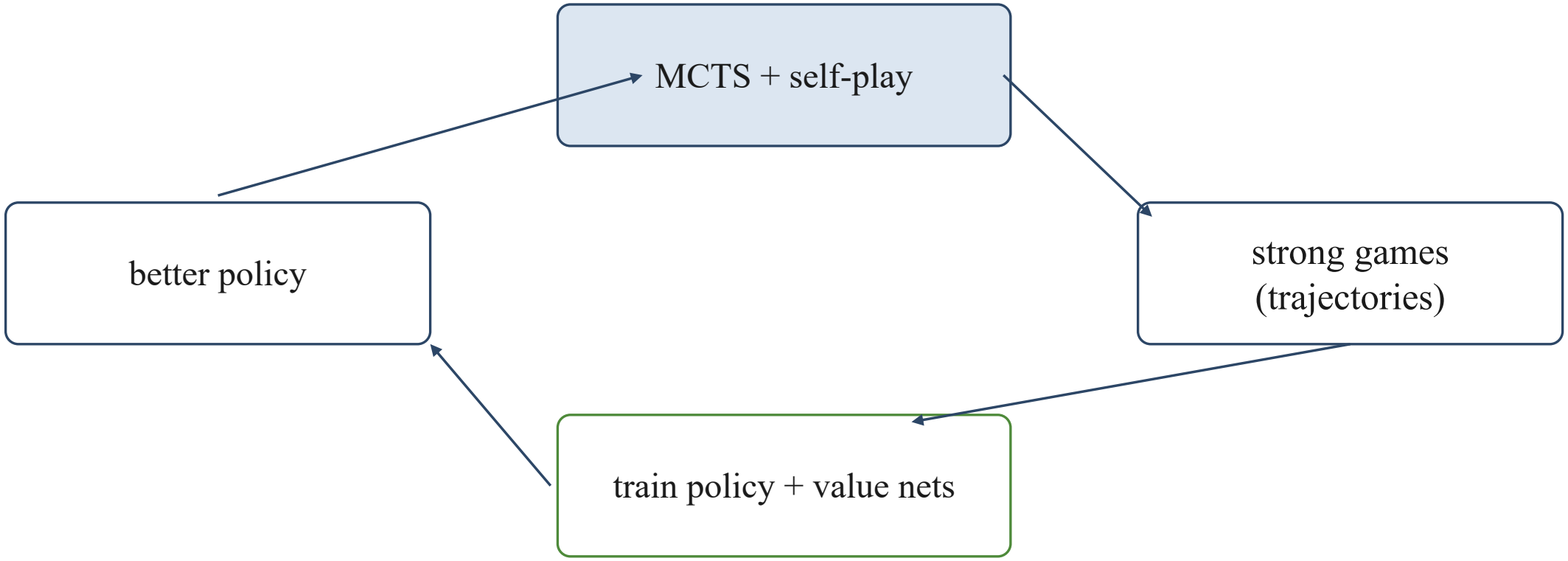
Don’t re-search every time. **Learn from the good trajectories search already found**, and fold them into the policy.



Pay the search cost once, at training time; keep the strength at inference for free.

We've seen this before: AlphaGo → AlphaZero

Use **search + self-play** to generate strong games, train the network on them, and **close the loop.**



each turn of the loop makes both the policy and the search stronger

[AlphaGo Zero / self-play: Silver et al., 2017]

Amortizing search into the policy

is Reinforcement Learning.

over reasoning traces today — and, for agents, over action trajectories

Part 2 — RL for reasoning **and agents**: Dehui and Chaoyi

Memorize → Decompose → Search → **RL — Part 2**

after the break