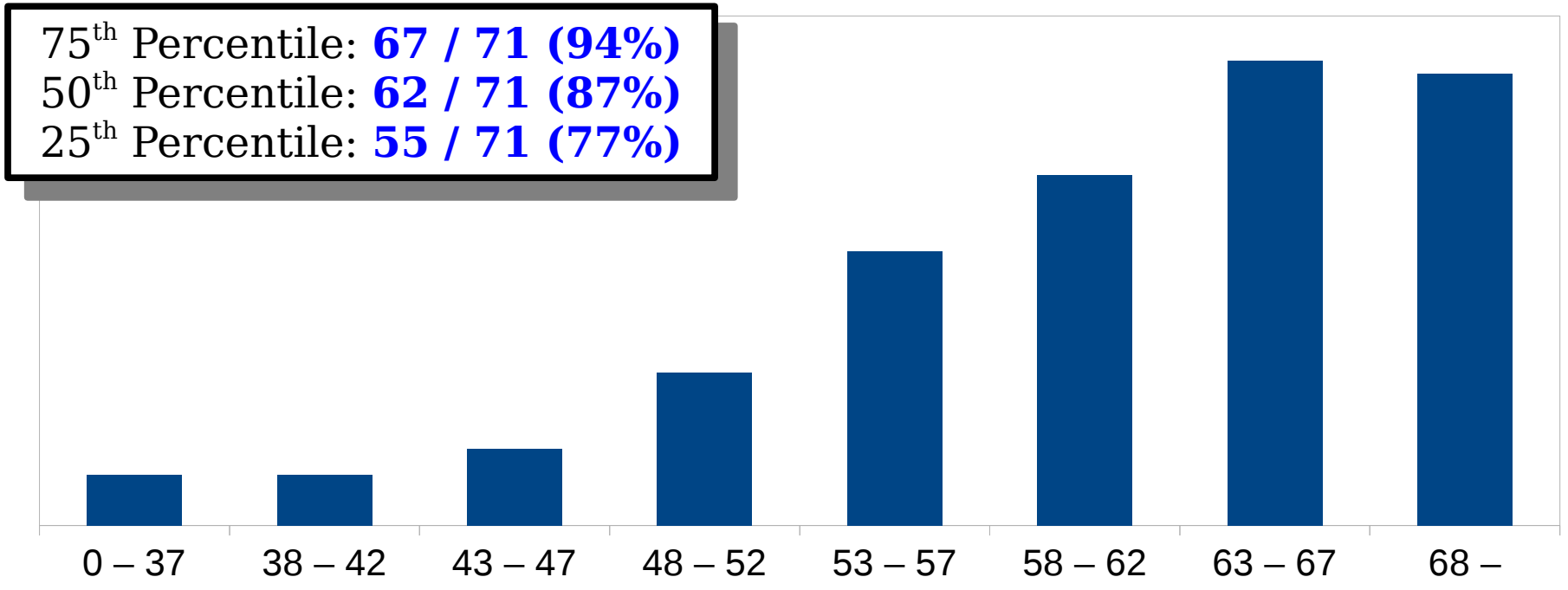
The Big Picture

Announcements

- Problem Set 9 was due thirty minutes ago. Solutions are available outside and also on the course website.
  - ***Congratulations – you're done with CS103 problem sets!***
- Problem Set 8 grade distribution:



Please evaluate this course on Axess.

Your feedback really makes a difference.

Final Exam Logistics

- Our final exam is this Monday, December 11th from 3:30PM – 6:30PM. Locations are divvied up by last (family) name:
  - Abb – Ngu: Go to ***Cubberley Auditorium***.
  - Ogr – Zwa: Go to ***Cemex Auditorium***.
- The final exam is cumulative and covers topics from PS1 – PS9 and all lectures. Topics from this week are fair game but will be deemphasized.
- The exam is closed-computer, closed-book, and limited-note. You may bring one double-sided 8.5” × 11” sheet of notes with you to the exam, decorated however you’d like.

Preparing for the Final

- On the course website you'll find
  - ***five*** practice final exams, which are all real exams with minor modifications, with solutions, and
  - a giant set of 45 practice problems (EPP3), with solutions.
- Our recommendation: Look back over the exams and problem sets and redo any problems that you didn't really get the first time around.
- Keep the TAs in the loop: stop by office hours to have them review your answers and offer feedback.

Outline for Today

- ***The Big Picture***
  - How does everything fit together?
- ***Where to Go from Here***
  - What's next in CS theory?
- ***A Final "Your Questions"***
  - You asked for it!
- ***Final Thoughts!***

The Big Picture

The Big Picture

Cantor's Theorem: $|S| < |\wp(S)|$

Corollary: Unsolvable problems exist.

What problems can
be solved by computers?

First, we need to learn how to prove results with certainty.

Otherwise, how can we know for sure that we're right about anything?

We also need a way to precisely pin down key terms and definitions.

Let's add some logic into the mix.

Let's study a few common discrete structures.

That way, we know how to model connected structures and relationships.

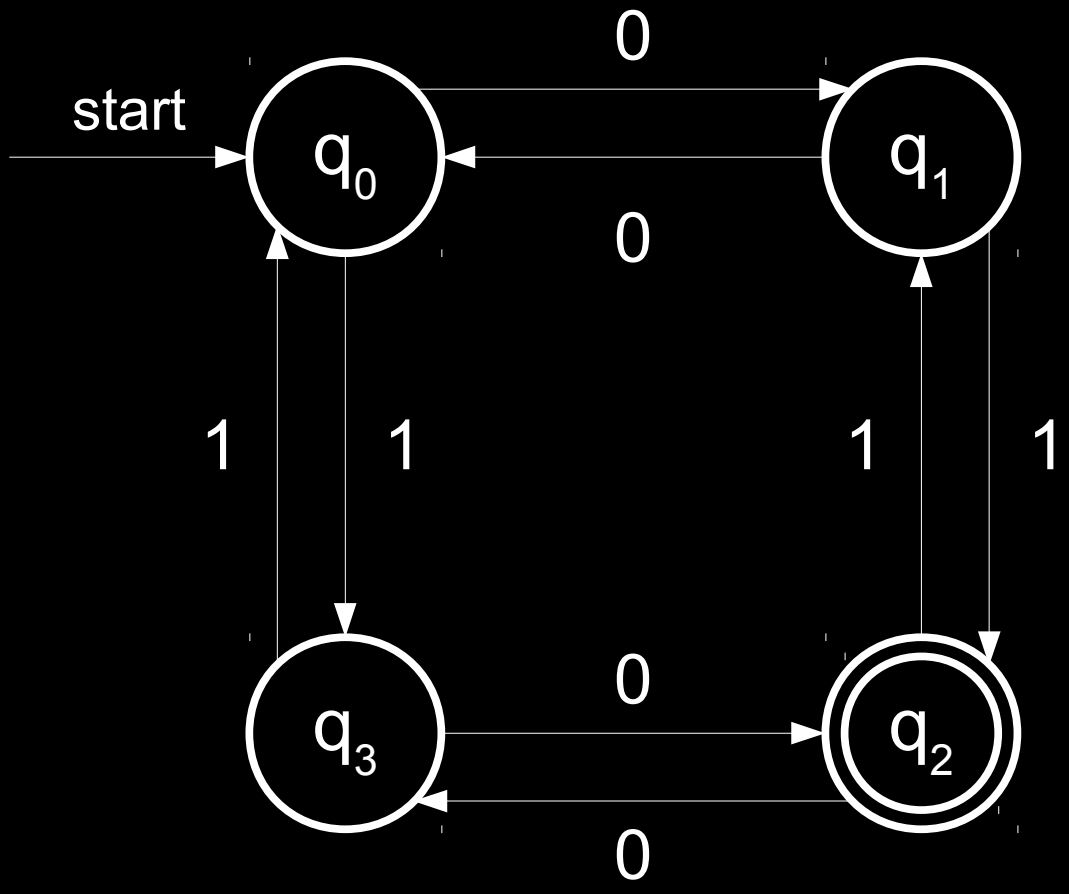
We also need to prove things about processes that proceed step-by-step.

So let's learn induction.

Okay! So now we're ready to go!

What problems are unsolvable?

Well, first we need a
definition of a computer!

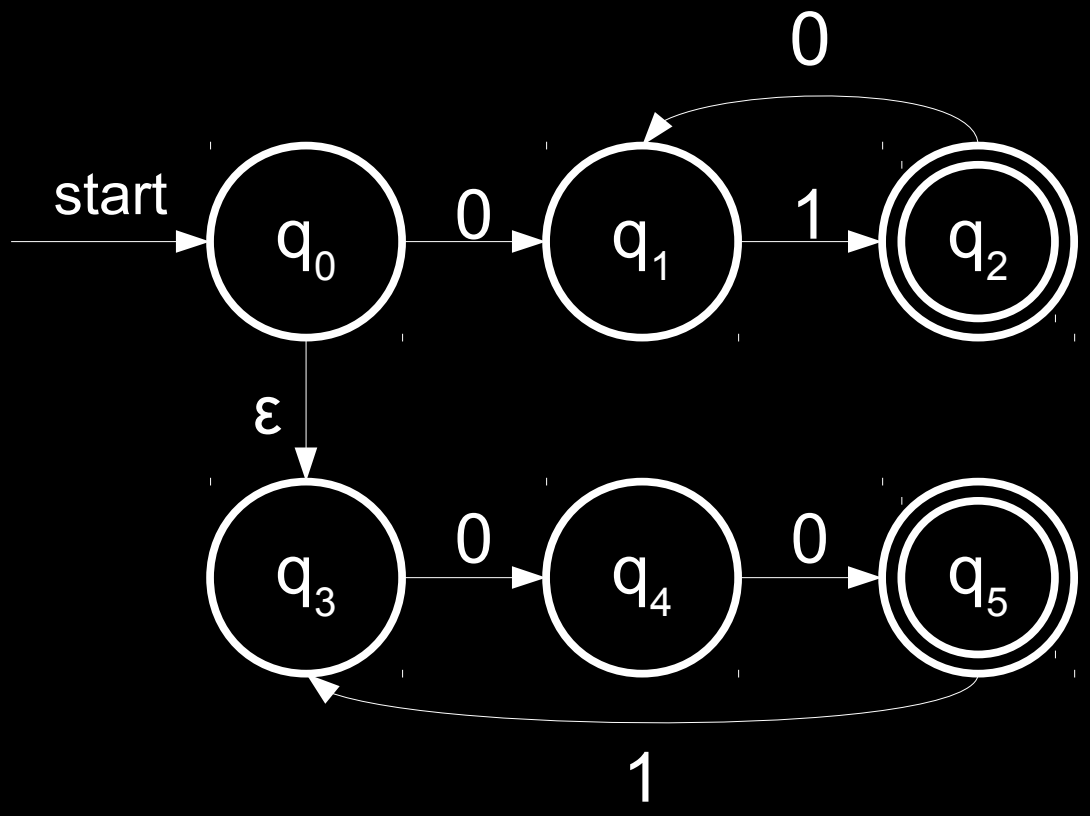

start
q_0
q_1
q_2
q_3
0
0
1
1
1
1
0
0

Cool! Now we have a model of a computer!

We're not quite sure what we can solve at this point, but that's okay for now.

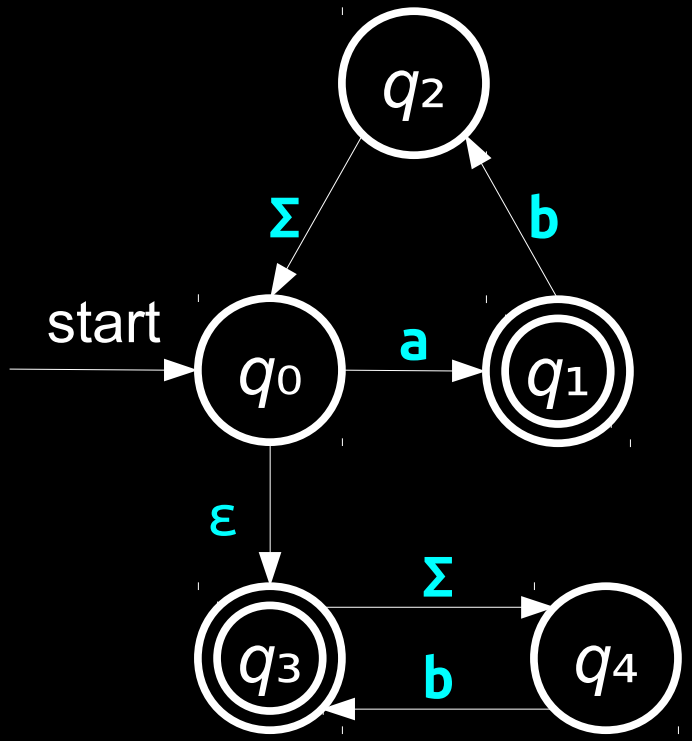
Let's call the languages we can capture this way the ***regular languages***.

I wonder what other
machines we can make?


start
q_0
q_1
q_2
q_3
q_4
q_5
0
0
1
ε
0
0
1

Wow! Those new machines are
way cooler than our old ones!

I wonder if they're more powerful?



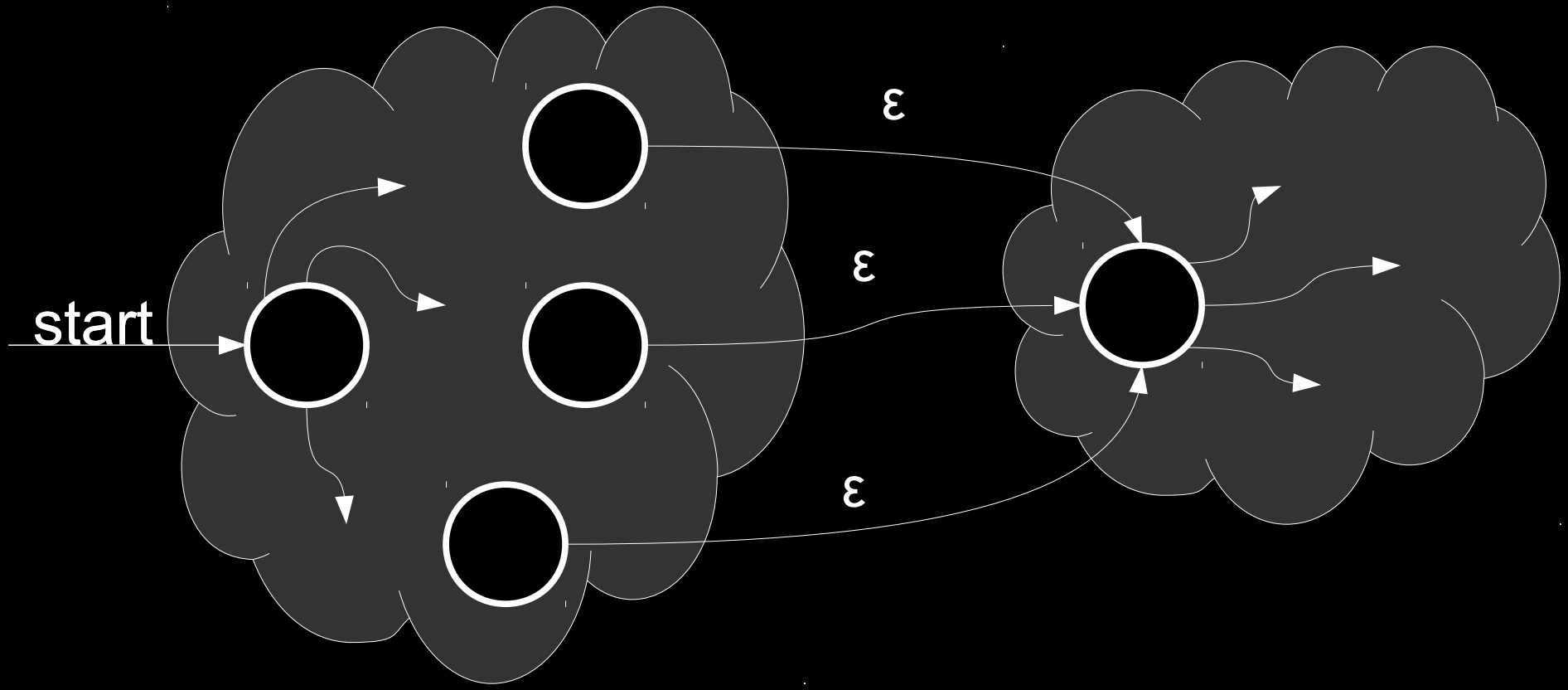
	a	b
*$\{q_0, q_3\}$	$\{q_1, q_4\}$	$\{q_4\}$
*$\{q_1, q_4\}$	$\emptyset$	$\{q_2, q_3\}$
$\{q_4\}$	$\emptyset$	$\{q_3\}$
*$\{q_2, q_3\}$	$\{q_0, q_3, q_4\}$	$\{q_0, q_3, q_4\}$
*$\{q_3\}$	$\{q_4\}$	$\{q_4\}$
*$\{q_0, q_3, q_4\}$	$\{q_1, q_4\}$	$\{q_3, q_4\}$
*$\{q_3, q_4\}$	$\{q_4\}$	$\{q_3, q_4\}$
$\emptyset$	$\emptyset$	$\emptyset$

Wow! I guess not. That's surprising!

So now we have a new way of modeling computers with finite memory!

However, we have just seen that ***computability*** (what problems can you solve?) is not the same as ***complexity*** (how efficiently can you solve the problem?)

I wonder how we can combine
these machines together?


start
ε
ε
ε

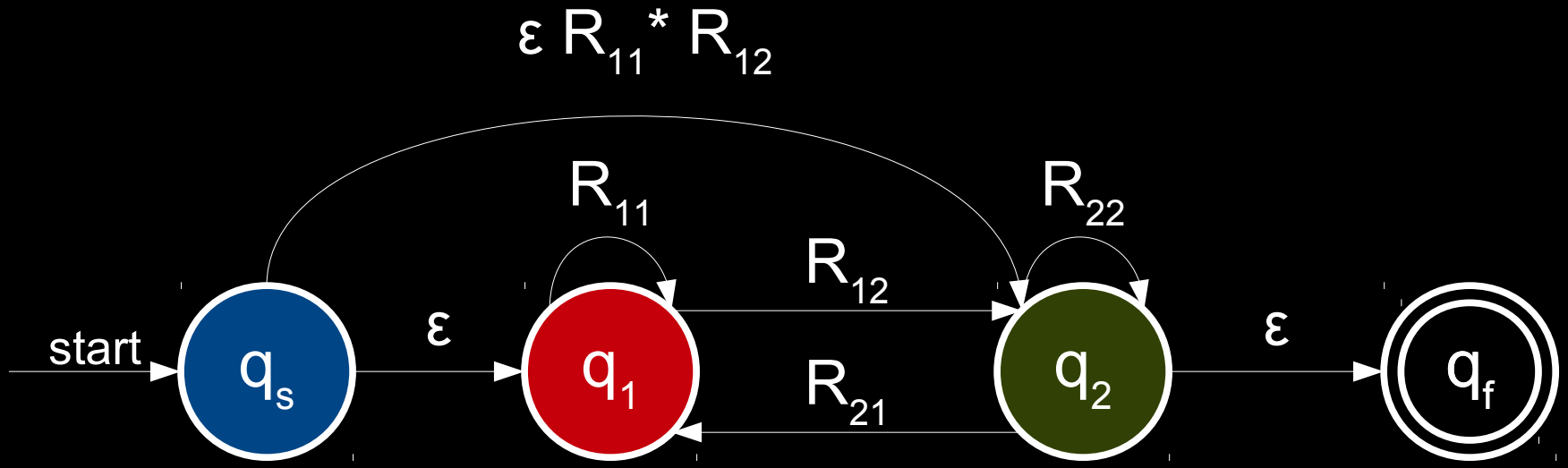
Cool! Since we can glue machines together, we can glue languages together as well.

How are we going to do that?

$$a^{+}(.a^{+})^{*}@a^{+}(.a^{+})^{+}$$

Wow! We've got a new way
of describing languages.

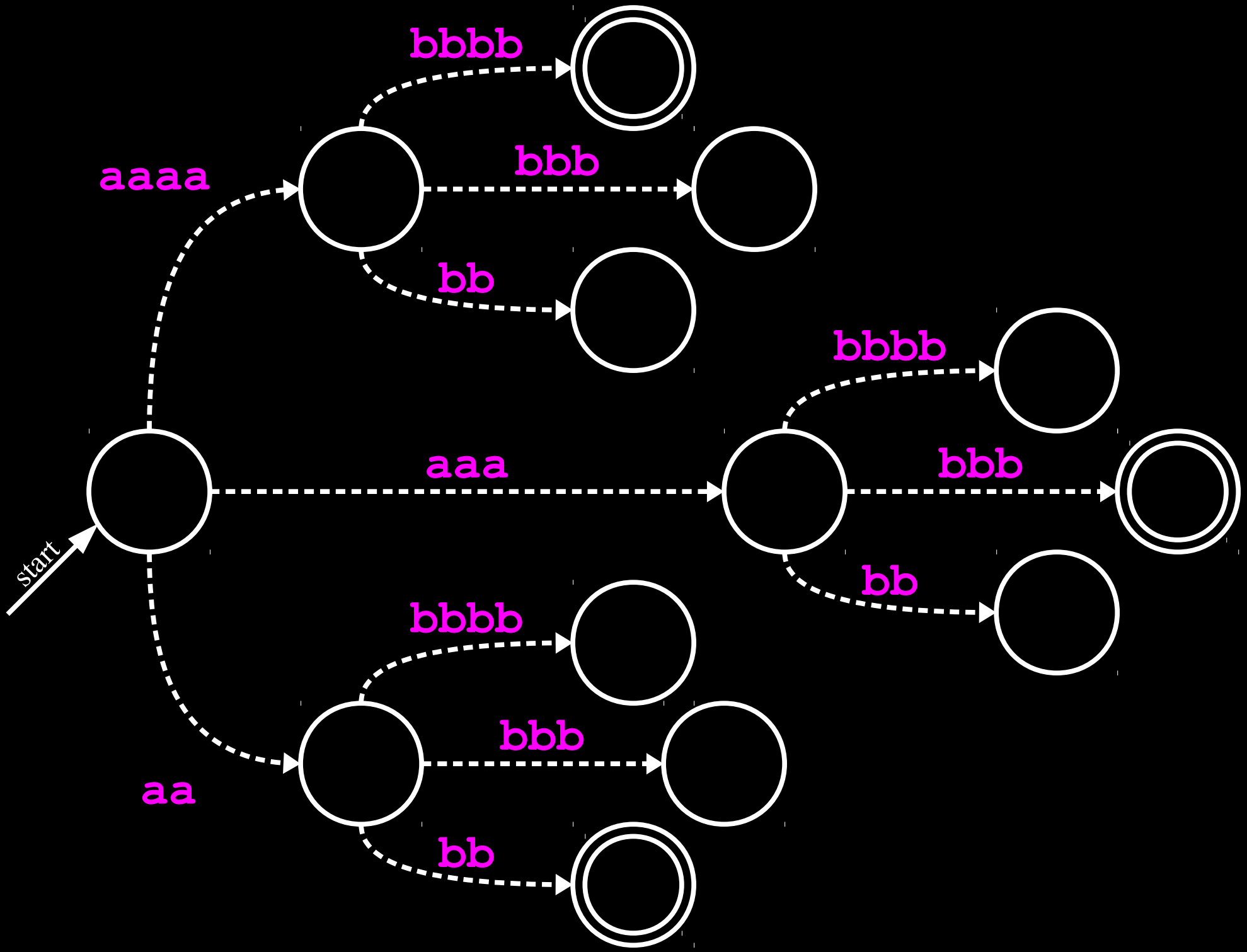
So what sorts of languages
can we describe this way?


ε R11* R12
R11
R22
R12
start
ε
ε
R21
qs
q1
q2
qf

Awesome! We got back the
exact same class of languages.

It seems like all our models give us the same power! Did we get every language?

$$xw \in L$$
$$yw \notin L$$


bbbb
aaaa
bbb
bb
bbbb
aaa
bbb
start
bb
bbbb
bbb
aa
bb

Wow, I guess not.

But we did learn something cool:

We have just explored what problems can be solved with finite memory.

So what else is out there?

Can we describe languages another way?

$$S \rightarrow aX$$
$$X \rightarrow b \mid C$$
$$C \rightarrow Cc \mid \varepsilon$$

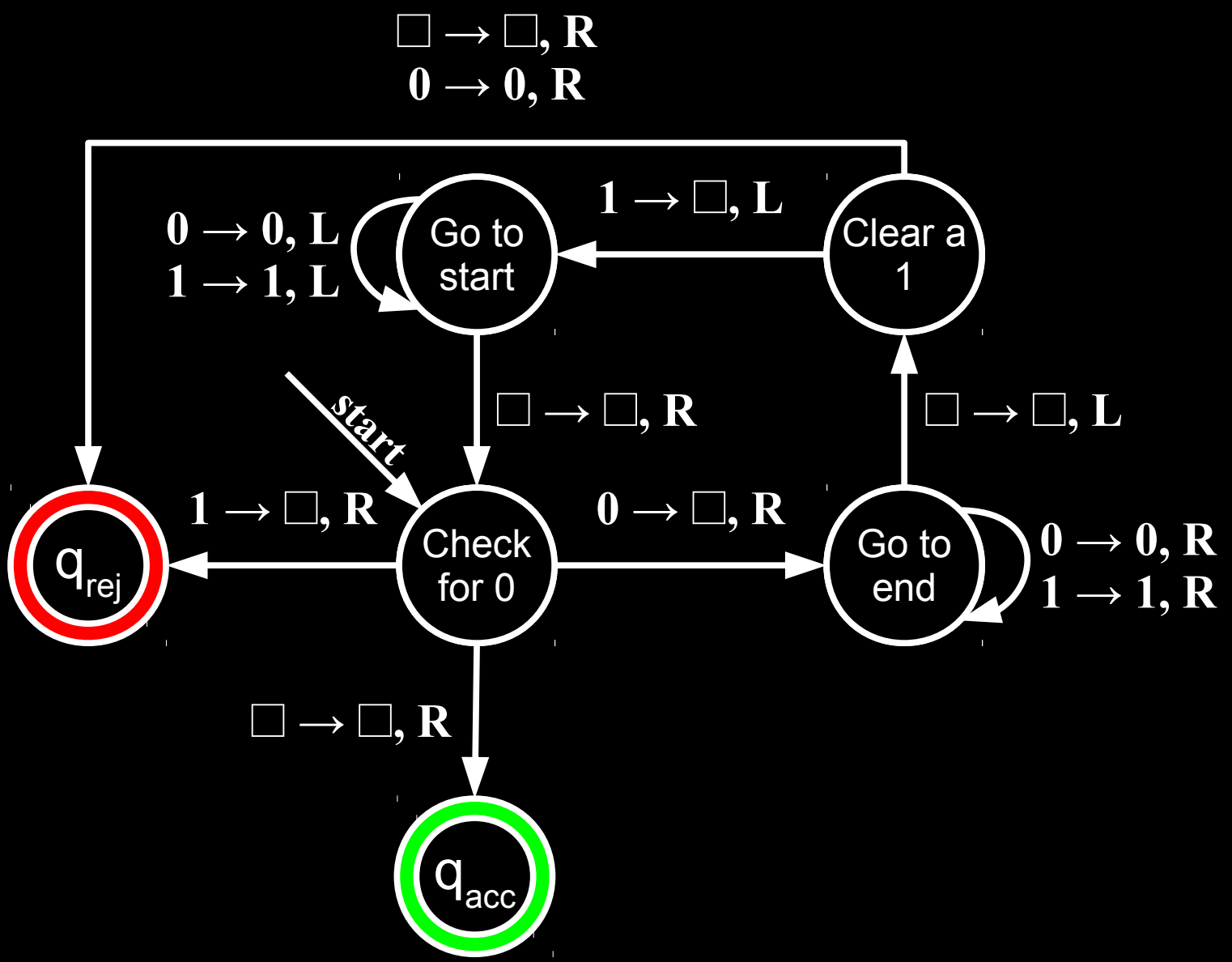
Awesome!

So, did we get every language yet?

$$|\Sigma^*| < |\wp(\Sigma^*)|$$

Hmmm... guess not.

So what if we make our memory a little better?

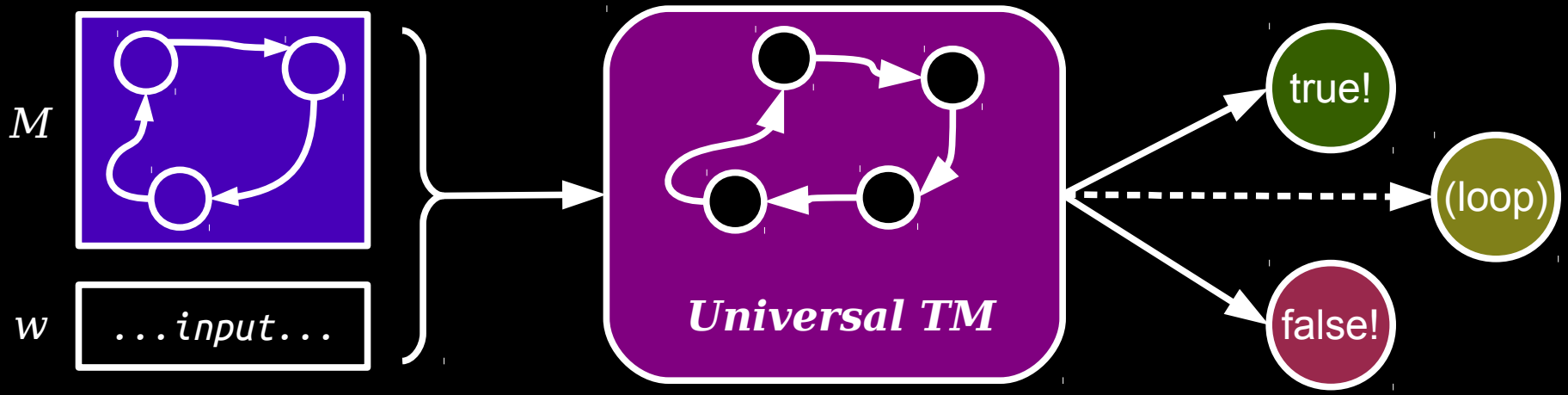

□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
1 → □, R
0 → □, R
Check for 0
Go to end
0 → 0, R
1 → 1, R
q_rej
□ → □, R
q_acc

Cool! Can we make these more powerful?

Wow! Looks like we can't get any more powerful.

(The ***Church-Turing thesis*** says that this is not a coincidence!)

So why is that?


M
w
...input...
Universal TM
true!
(loop)
false!

Wow! Our machines can simulate one another!

This is a theoretical justification for why all these models are equivalent to one another.

So... can we solve everything yet?

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
using namespace std;

const vector<string> kToPrint = {
    ...
};

void printProgramInQuotes() {
  for (string line: kToPrint) {
    cout << "  " << quoted(line) << "," << endl;
  }
}

int main() {
  for (string line: kToPrint) {
    if (line == "@") {
      printProgramInQuotes();
    } else {
      cout << line << endl;
    }
  }
}
```

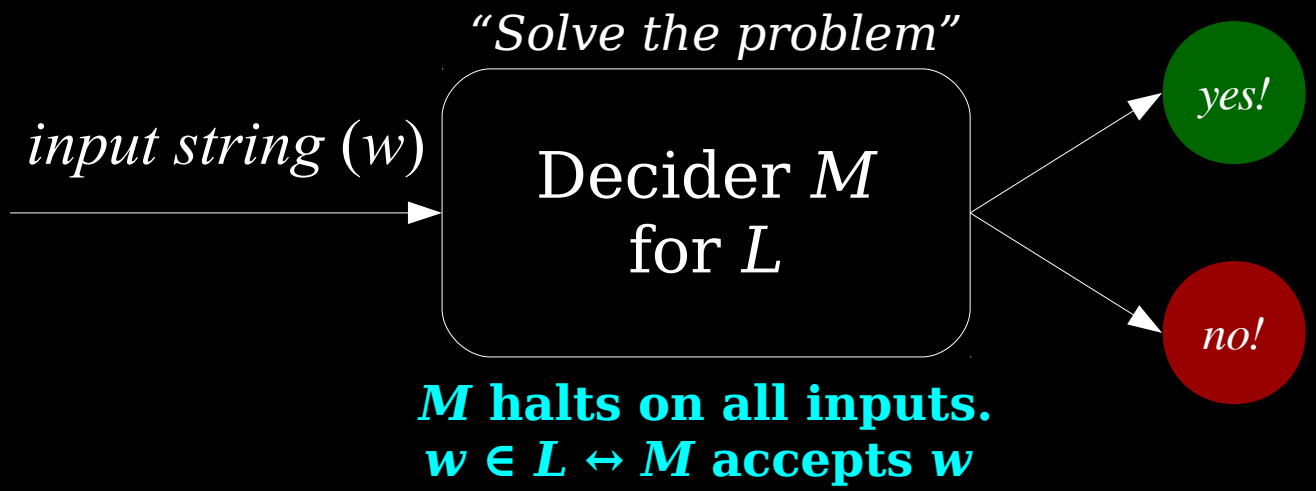
Weird! Programs can gain access
to their own source code!

Why does that matter?

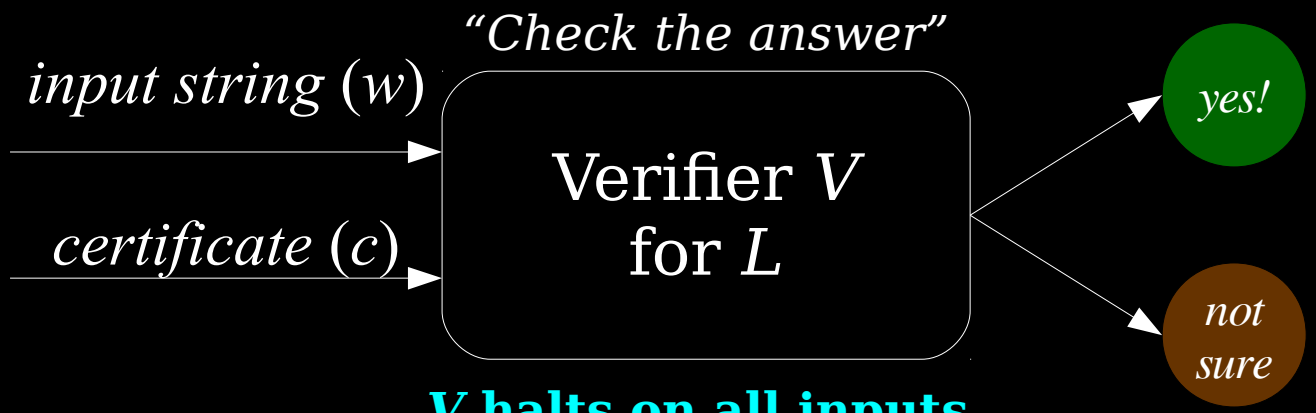
```
int main() {
   string me = mySource();
   string input = getInput();

   if (willAccept(me, input)) {
      reject();
   } else {
      accept();
   }
}
```

Crazy! The power of self-reference immediately limits what TMs can do!

What if we think about solving problems in a different way?



M halts on all inputs.
$w \in L \leftrightarrow M$ accepts w



V halts on all inputs.
$w \in L \leftrightarrow \exists c \in \Sigma^*.\ V$ accepts $\langle w, c \rangle$

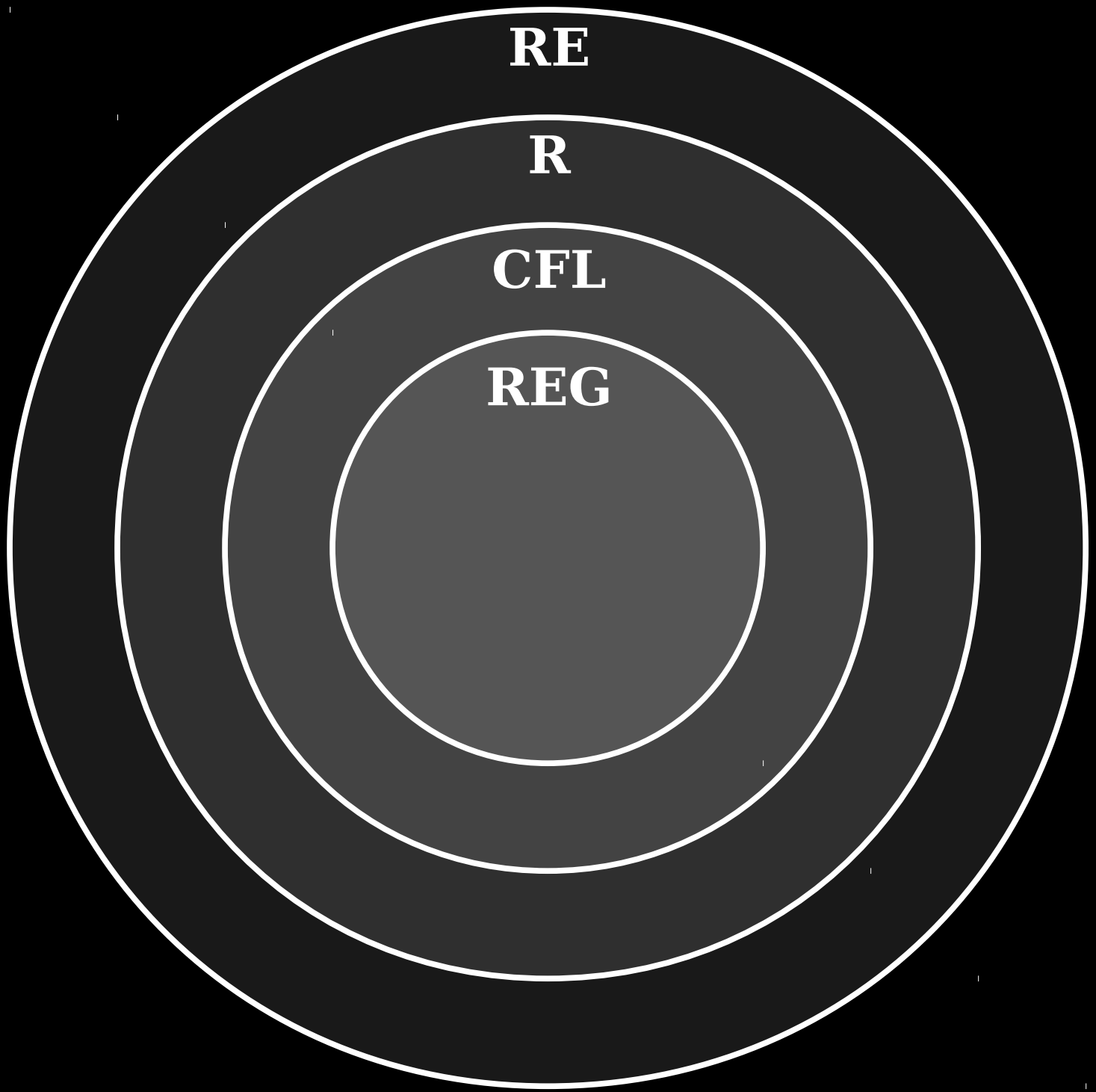
Crazy!

Can we at least verify everything?

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…
	No	No	No	Acc	No	Acc	…

Oh great. Some problems
are impossible to solve.

But look what we learned along the way!


RE
R
CFL
REG

Wow. That's pretty deep.

So... what can we do efficiently?

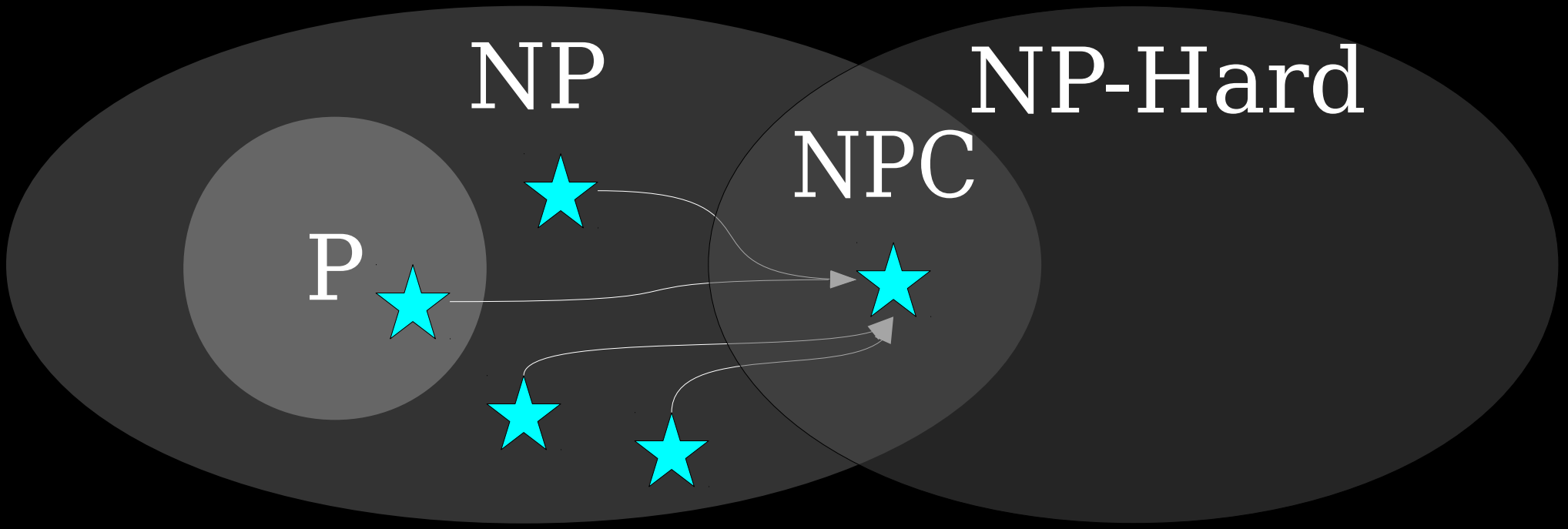
P


NP

So... how are you two related again?

No clue.

But what do we know about them?

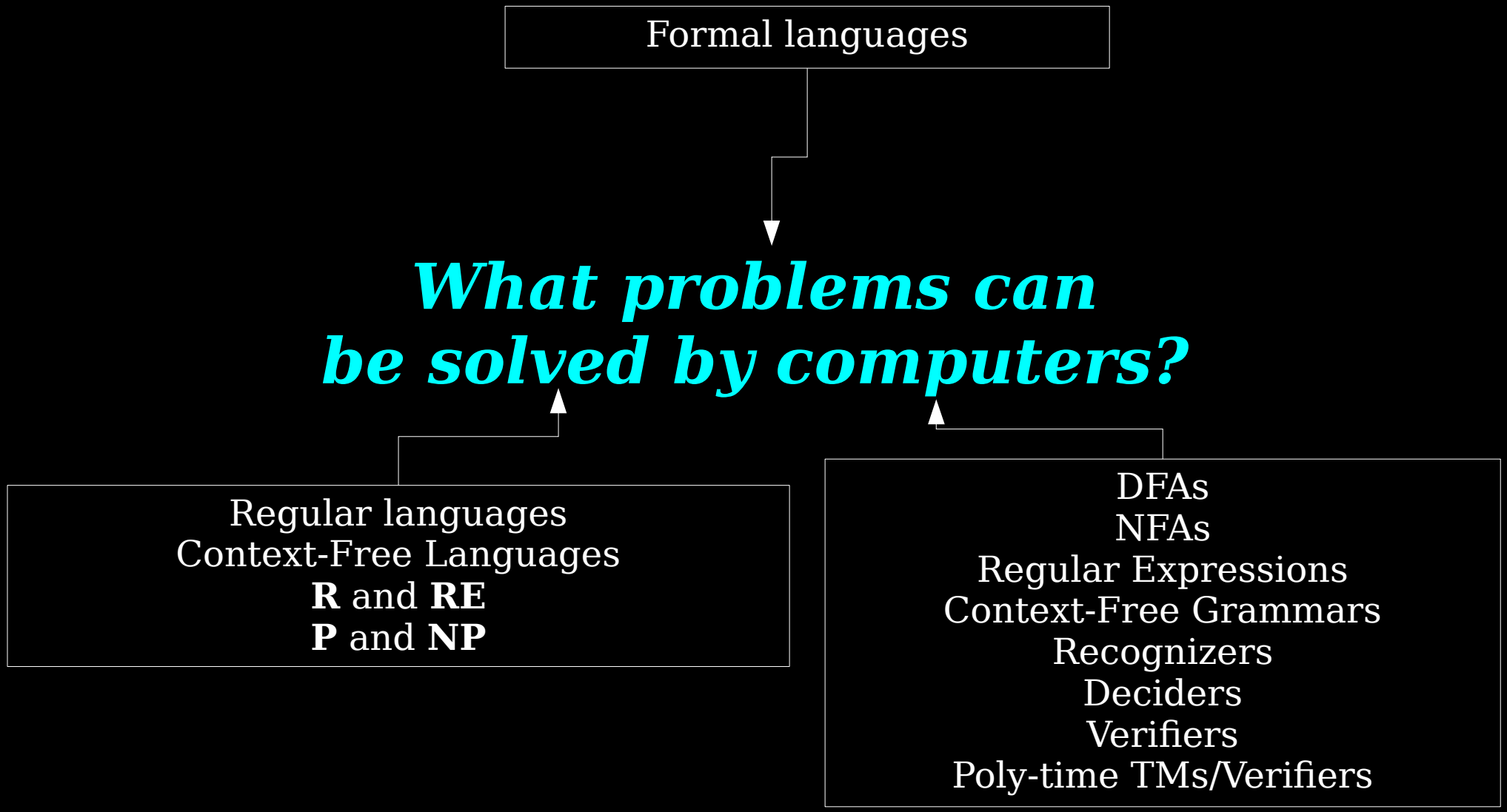

NP
NP-Hard
NPC
P

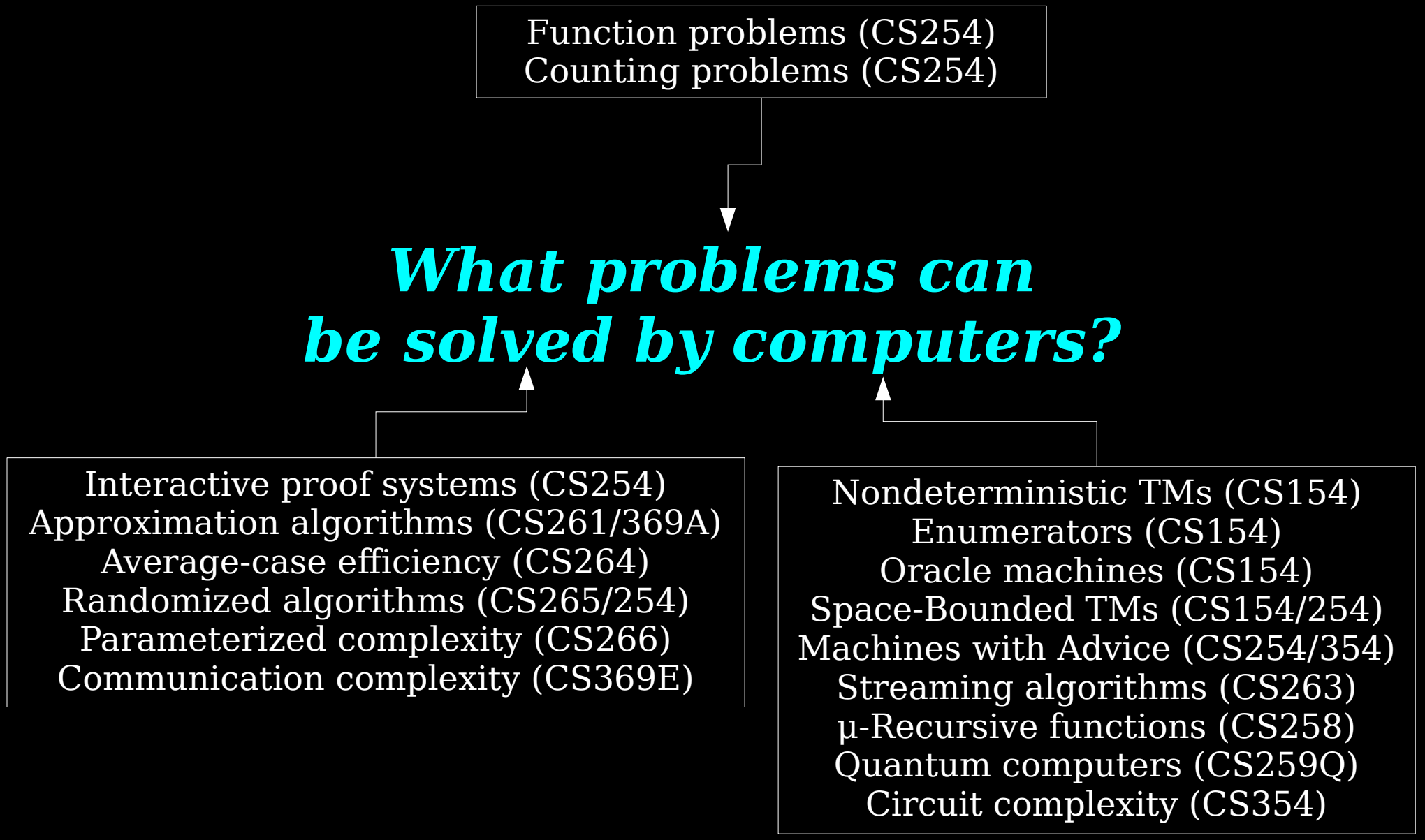
We've gone to the absolute limits of computing.

We've probed the limits of efficient computation.

Congratulations on making it this far!

What's next in CS theory?


Formal languages
What problems can be solved by computers?
Regular languages
Context-Free Languages
R and RE
P and NP
DFAs
NFAs
Regular Expressions
Context-Free Grammars
Recognizers
Deciders
Verifiers
Poly-time TMs/Verifiers


Function problems (CS254)
Counting problems (CS254)
What problems can be solved by computers?
Interactive proof systems (CS254)
Approximation algorithms (CS261/369A)
Average-case efficiency (CS264)
Randomized algorithms (CS265/254)
Parameterized complexity (CS266)
Communication complexity (CS369E)
Nondeterministic TMs (CS154)
Enumerators (CS154)
Oracle machines (CS154)
Space-Bounded TMs (CS154/254)
Machines with Advice (CS254/354)
Streaming algorithms (CS263)
μ-Recursive functions (CS258)
Quantum computers (CS259Q)
Circuit complexity (CS354)

How do we actually get the computer to effectively solve problems?

DFA design intuitions
Guess-and-check
Massive parallelism
Myhill-Nerode lower bounds
Verification
Polynomial-time reductions

How do we actually get the computer to effectively solve problems?

Algorithm design (CS161)
Efficient data structures (CS166)
Modern algorithmic techniques (CS168)
Approximation algorithms (CS261/CS369A)
Average-case efficient algorithms (CS264)
Randomized algorithms (CS265)
Parameterized algorithms (CS266)
Geometric algorithms (CS268)
Game-theoretic algorithms (CS364A/B)

What mathematical structures arise in computer science?

Sets
Propositional and First-Order Logic
Equivalence Relations
Strict Orders
Functions
Injections, Surjections, Bijections
Graphs
Planar and Bipartite Graphs
Polynomial-Time Reductions

What mathematical structures arise in computer science?

Groups, Rings, and Fields (Math 120, CS255)
Trees (Math 108, CS161)
Graphs (Math 107, Math 108)
Hash Functions (CS109, CS161, CS255)
Permutations (Math 120, CS255)
Monoids (CS149)
Lattices and Semilattices (CS143)
Control-Flow Graphs (CS143)
Vectors and Matrices (Math 113, EE103, CS205A)
Modal Logic (Phil 154, CS224M)
Mapping Reductions (CS154)

Where does CS theory meet CS practice?

Finite state machines
Regular expressions
CFGs and programming languages
Password-checking
Secure voting
Polynomial-time reducibility
NP-hardness and **NP**-completeness

Where does CS theory meet CS practice?

Compilers (CS143)
Computational logic (CS157)
Program optimization (CS243)
Data mining (CS246)
Cryptography (CS255)
Programming languages (CS258)
Network protocol analysis (CS259)
Techniques in big data (CS263)
Graph algorithms (CS267)
Computational geometry (CS268)
Algorithmic game theory (CS364)

A Whole World of Theory Awaits!

What's being done here at Stanford?

Algorithms ∩ Game theory
(Tim Roughgarden)

Learning patterns in randomness
(Greg Valiant)

Moving from secrecy to privacy
(Omer Reingold)

Approximating NP-hard problems
(Moses Charikar)

Optimizing programs... randomly
(Alex Aiken)

Computing on encrypted data

(Dan Boneh)

Interpreting structure from shape
(Leonidas Guibas)

Correcting errors automatically
(Mary Wootters)

So many options – what to do next?

Really enjoyed this class?

Give CS154 a try!

Interested in trying out CS?

Continue on to CS109!

Want to see this material come to life?

Check out CS143!

Want to tame infinity?

Dive into Math 161!

Like discrete structures?

Try Math 107 or Math 108!

Want to just go write code?

Take CS107!

Keep on exploring! There's so much more to learn!

A Final “Your Questions”

“What are some things you wish you could tell yourself as an undergrad?”

Be intentional. Be curious. Be circumspect. And have fun - everything is going to turn out okay.

“I've noticed that a lot of my Stanford friends "humble brag" a lot - how can I let them know when they're crossing over from confidence to arrogance?”

A great way to have this conversation is to avoid framing it as “you are X” and more as “when you do X I feel Y.” This is way more likely to lead to a constructive outcome. It also forces you to think through what it is about this that bothers you so that you can have a more honest conversation.

This advice applies more generally - if your goal is to shape behavior, avoid accusing someone of an intrinsic, incurable character flaw.

“I’m gay and I have a habit of liking guys that seem straight. How should I bring up the topic with them? Should I just give up?”

I honestly have no idea how to do this since I can’t say I’ve been in the same position. But I imagine that there are a lot of other people at the university who could give you some really good tips and pointers here.

“Any music you like listening to? :D”

I'm currently jamming out to the soundtrack to the 1984 concert film "Stop Making Sense." That movie is basically David Byrne and the Talking Heads having a ton of fun on stage and it really comes through in the music!

“A lot of students feel pressured to do CS so that they can 'be successful' down the line, foregoing true passions in the process. How can we combat this?”

Story time!

"Sometimes I feel like learning the low level programming stuff (i.e. pointers) is not as interesting as theory. How can I get more excited about that stuff?"

An interesting perspective to take: all of that low-level stuff, all the details of how the computer works, all the weirdnesses of assembly and pointers and threads - you're looking at something that someone made! Like, this didn't exist one hundred years ago! And it's astonishing how something that's changed things so much just boils down to a machine that picks up bytes from one place in memory and puts them down in another.

“Sometimes college seems like an unnecessarily stressful environment. Why do you think this has to be the case? Is there something wrong with our system?”

It’s worth seeing where this stress comes from. How much of the stress is identity-driven (“I’m smart and I’m supposed to be good at this, and it’s not working”), how much of it is market-driven (“We’re company X and we will tempt 18-year-olds with gazillions of dollars for Fun and Profit!”), how much of it is family-driven (“We’ve worked hard to get you here, and you’d better not disappoint.”), how much of it is social (“Everyone around me had opportunities X, Y, and Z and has no clue what A, B, and C feel like”), etc. These are real issues and there aren’t necessarily easy answers to them. Part of the challenge of being a student is learning how to navigate them - figuring out how you define yourself, determining what your values are, scoping your personal and family boundaries, and constructing your own set of cultural values.

“I really liked CS103 so I want to take more CS theory. However, I'm worried that I'll waste time learning useless things. What do you recommend?”

Theory is all about training yourself to think in new ways, to approach concepts you've seen from different perspectives, and to take a step back from what the task at hand to ask what it is that you're really doing. That's a valuable skill. Even if you never end up directly using any of the specifics of what you learn – which I think is incredibly unlikely – the approach to thinking you'll have developed will help you immensely along the way.

Anything else?

Final Thoughts

A Huge Round of Thanks!

There are more problems to solve than there are programs capable of solving them.

There is so much more to explore and so many big questions to ask – ***many of which haven't been asked yet!***


Theory
Practice

You now know what problems we can solve, what problems we can't solve, and what problems we believe we can't solve efficiently.

My questions to you:

What problems will you ***choose*** to solve?
Why do those problems matter to you?
And how are you going to solve them?