

The Big Picture

Announcements

- Problem Set 9 due right now. We'll release solutions right after lecture.
 - ***Congratulations - you're done with CS103 problem sets!***
- Practice final exam is Monday, December 8 from 3:00PM – 6:00PM in Cemex Auditorium.
- **Please evaluate this course on Axxess!** Your feedback really does make a difference.

The Big Picture

The Big Picture

Imagine what it must have been like to discover all of the results in this class.

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?

Now, we need to learn how to prove things about processes that proceed step-by-step.

So let's learn induction.

We also should be sure we have some
rules about reasoning itself.

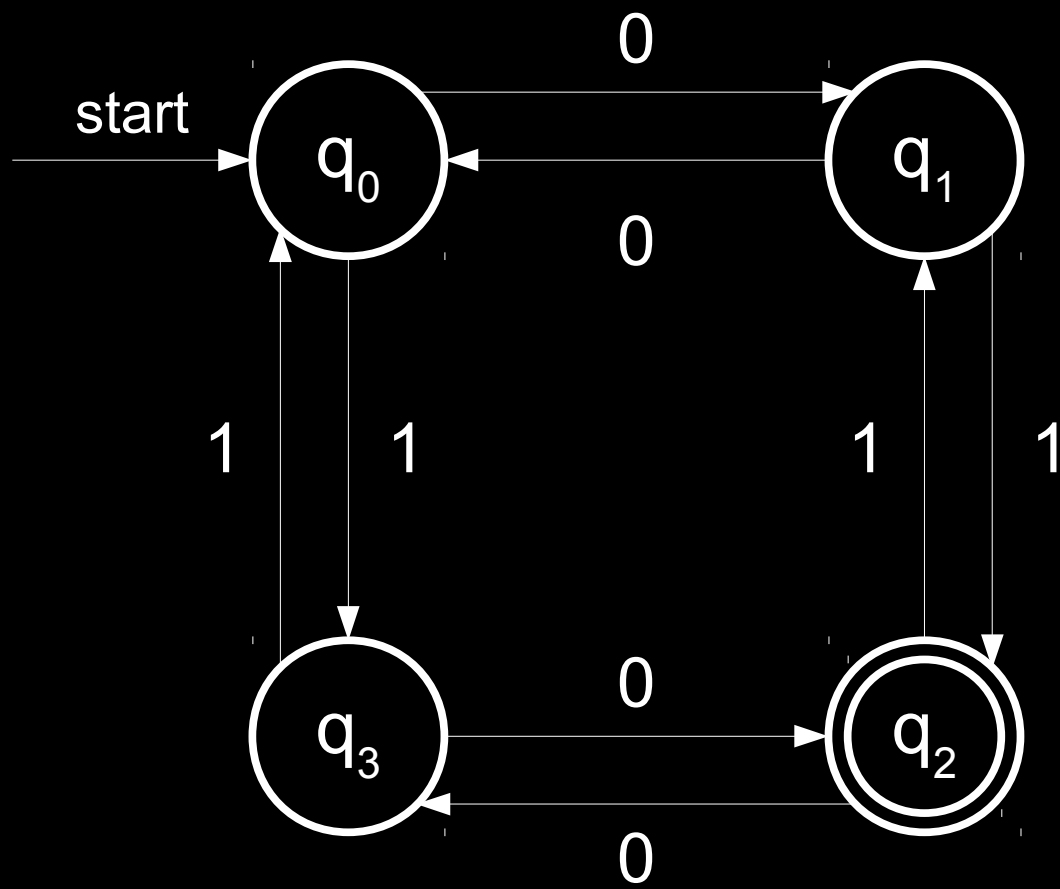
Let's add some logic into the mix.

Finally, let's study a few common discrete structures.

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

Okay! So now we're ready to go!
What problems are unsolvable?

Well, first we need a
definition of a computer!

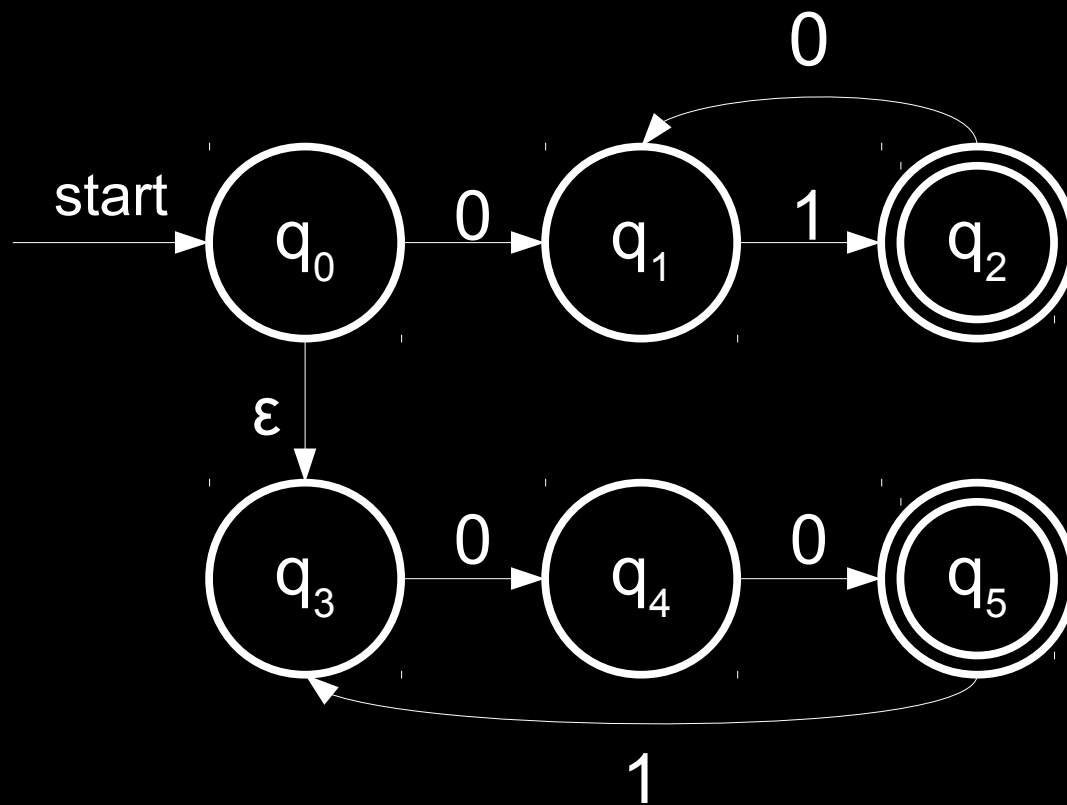


Cool! Now we have a model of a computer!

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

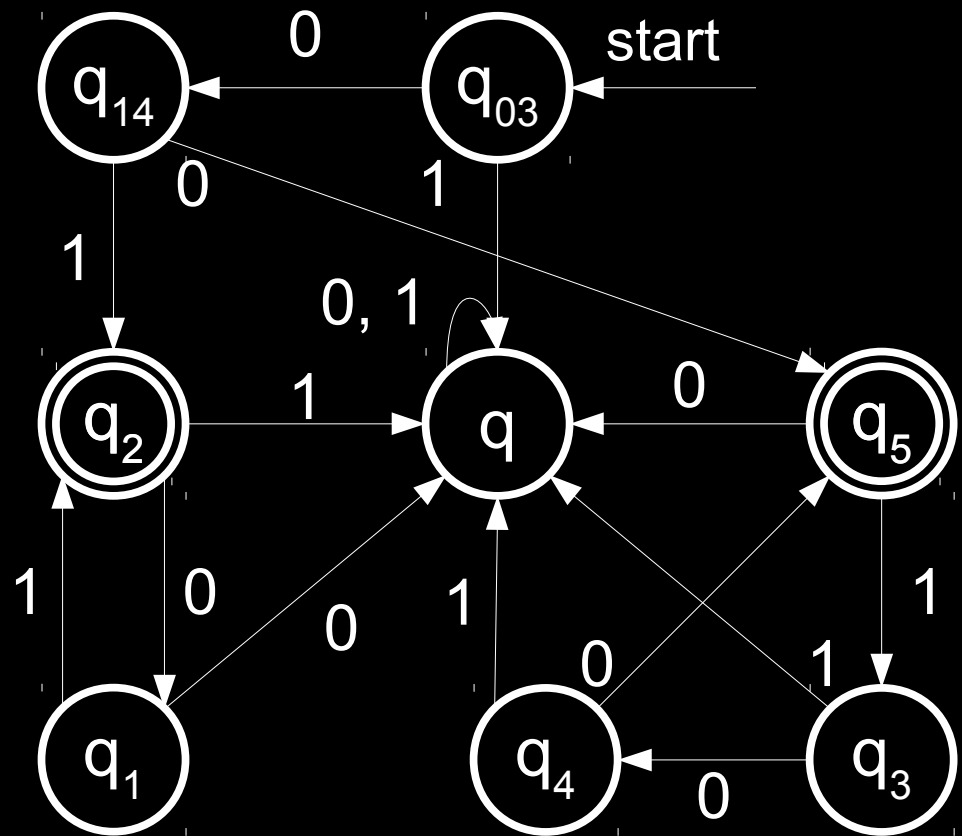
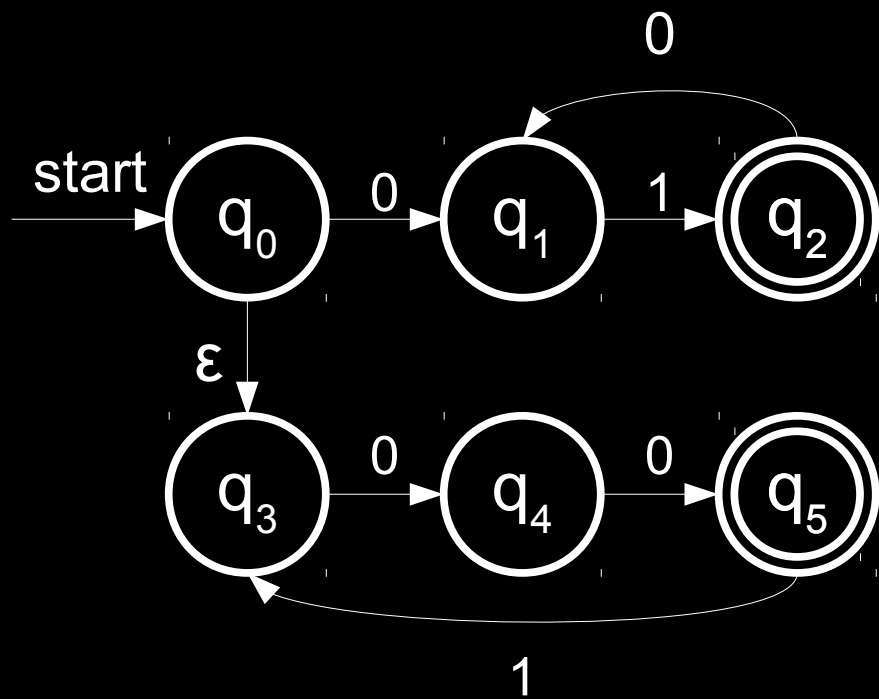
Let's call the languages we can capture this way the ***regular languages***.

I wonder what other
machines we can make?



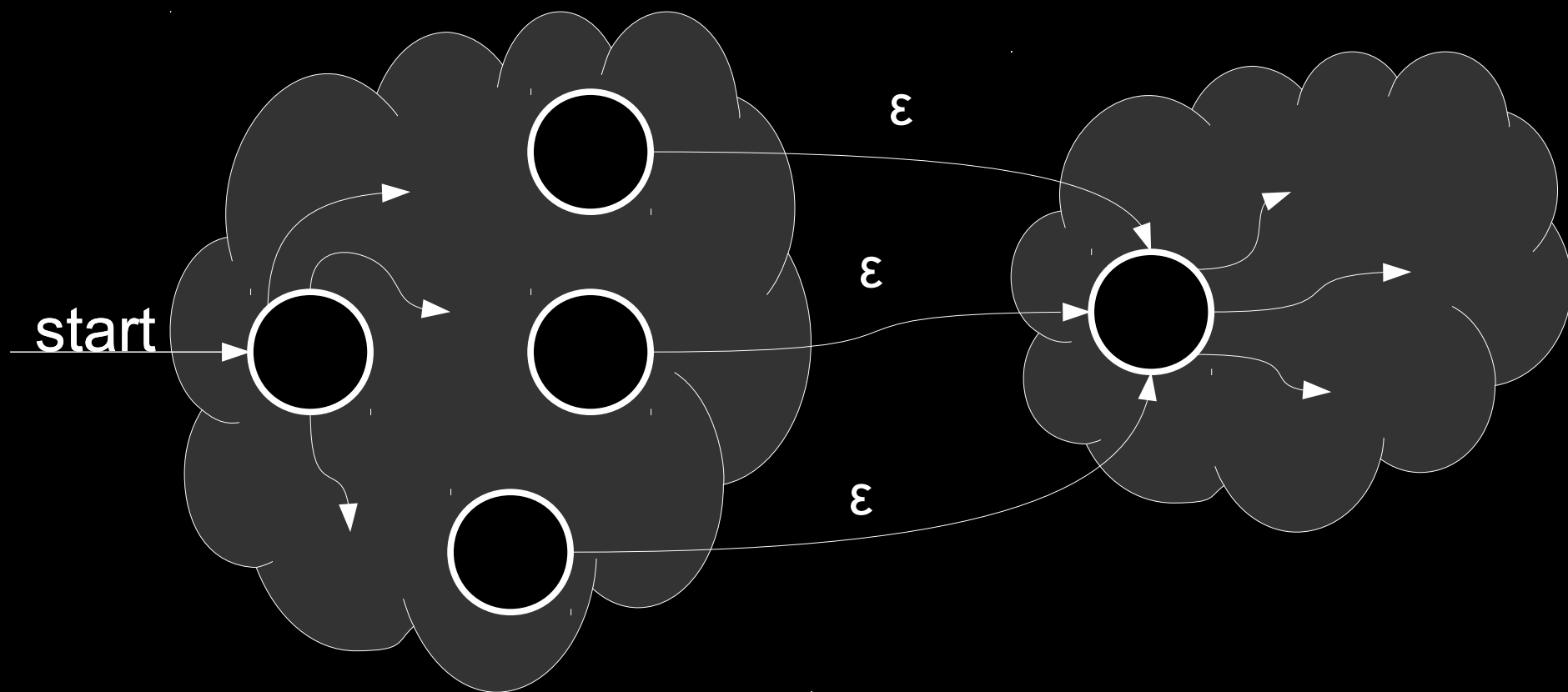
Wow! Those new machines are
way cooler than our old ones!

I wonder if they're more powerful?



Wow! I guess not. That's surprising!
So now we have a new way of modeling
computers with finite memory!

I wonder how we can combine
these machines together?



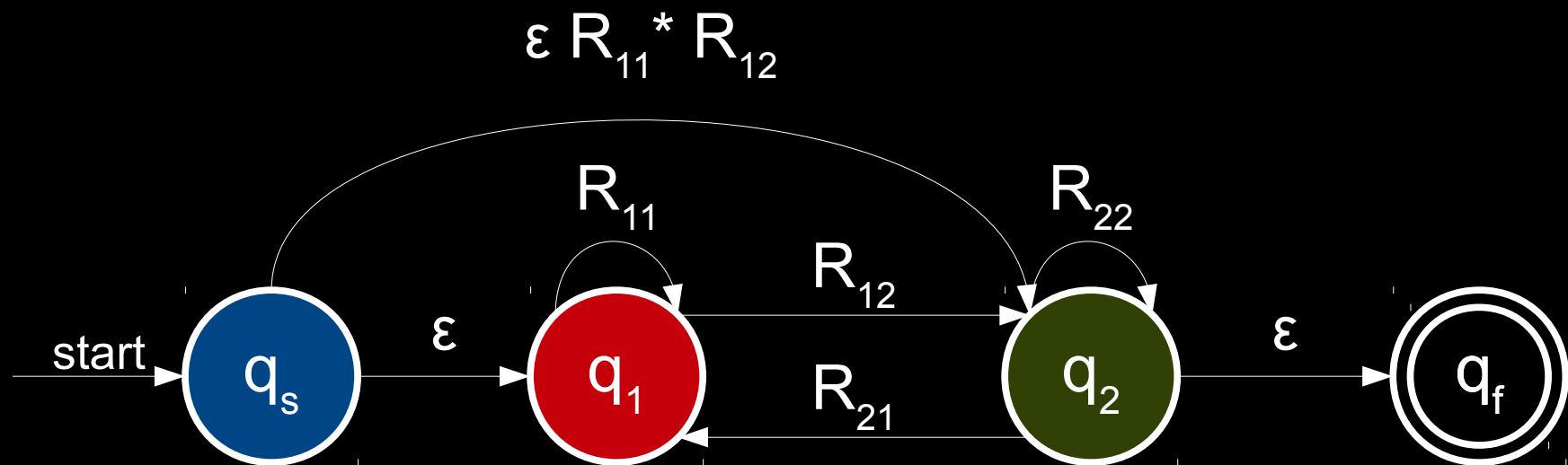
Cool! Since we can glue
machines together, we can glue
languages together as well.

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?



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?

$$xw \in L$$

$$yw \notin L$$

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?

$$S \rightarrow aX$$

$$X \rightarrow b \mid C$$

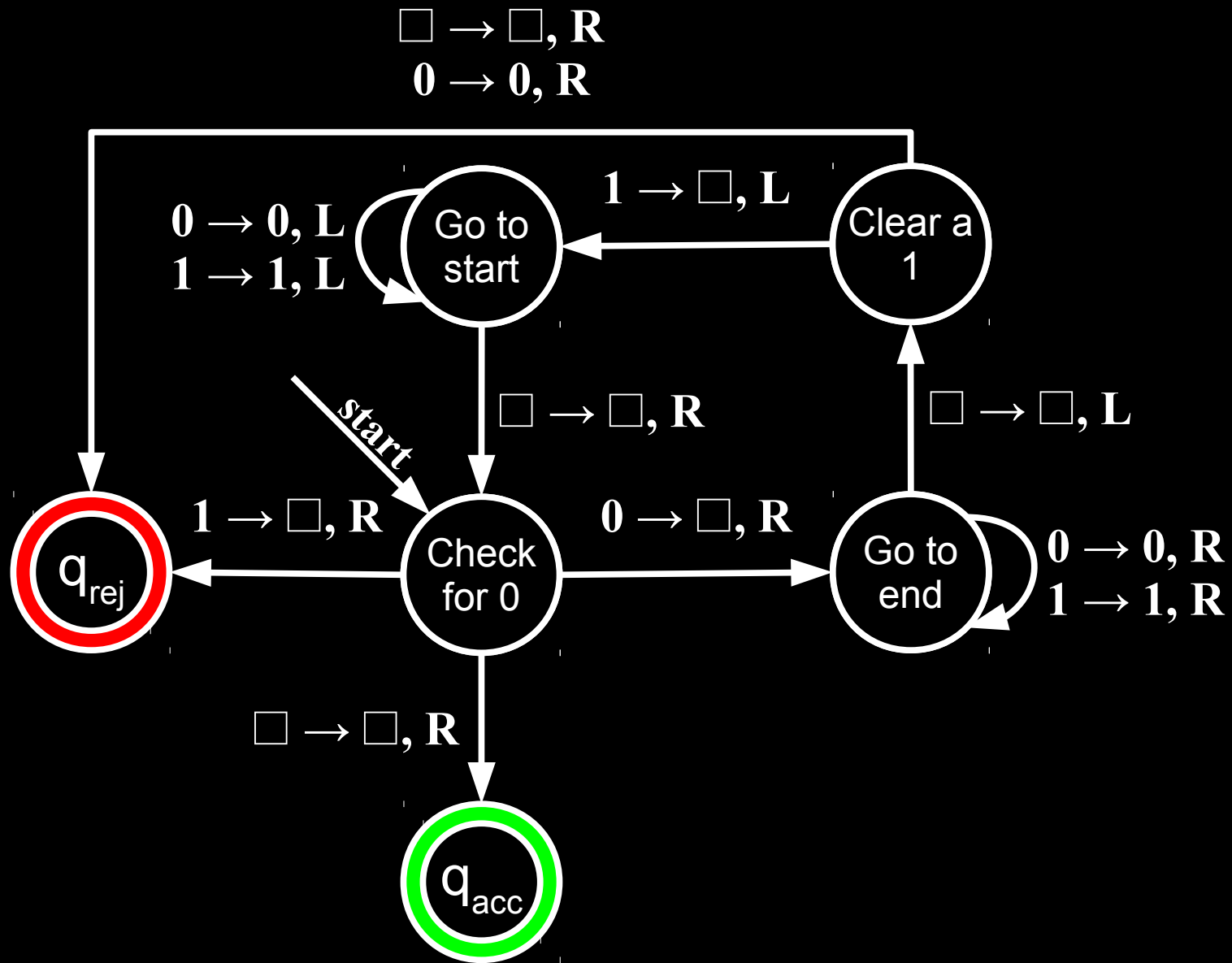
$$C \rightarrow Cc \mid \epsilon$$

Awesome!

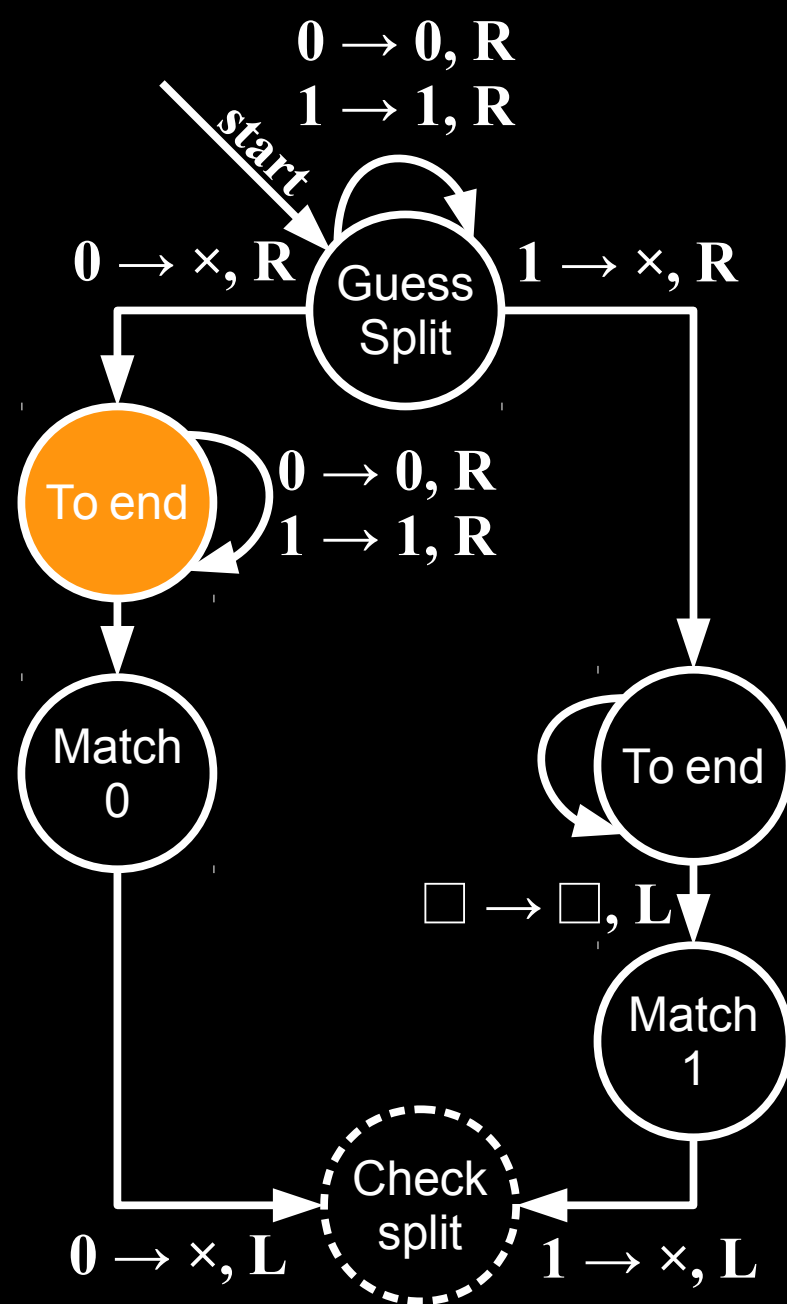
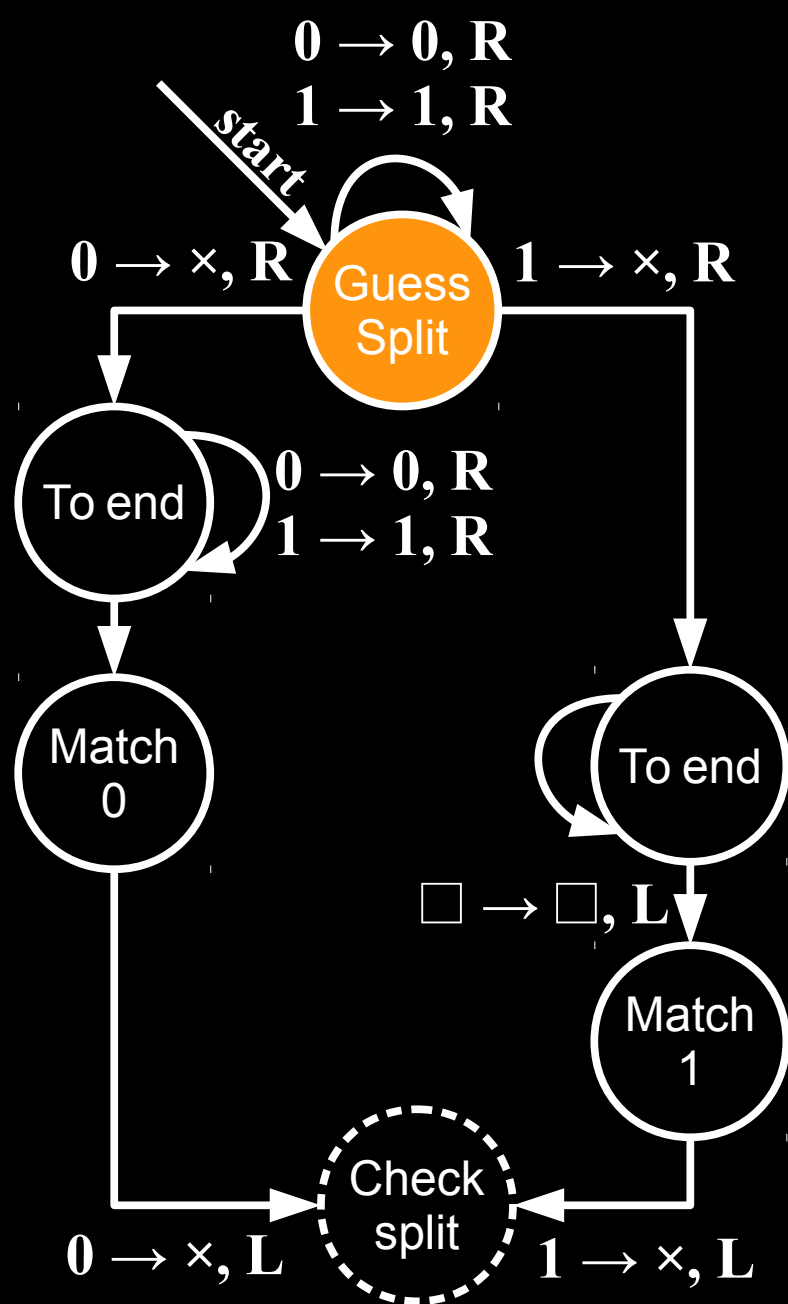
So, did we get every language yet?

Hmmm... guess not.

So what if we make our
memory a little better?



Cool! Can we make these
more powerful?



$V =$ “On input $\langle w, T \rangle$, where T is a sequence of transitions:

- Run N on w , following transitions in the order specified in T .
- If any of the transitions in T are invalid or can't be followed, reject.
- If after following the transitions N accepts w , accept; otherwise reject.

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?

U_{TM} = “On input $\langle M, w \rangle$, where M is a TM and $w \in \Sigma^*$:
Set up the initial configuration of M running on w .
while (true) {
 If M accepted w , then U_{TM} accepts $\langle M, w \rangle$.
 If M rejected w , then U_{TM} rejects $\langle M, w \rangle$.
 Otherwise, simulate one more step of M on w .
}”

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?

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$...
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 why exactly is that?

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
const vector<string> kToPrint = {
    /* ... */
};

void printProgramInQuotes() {
    for (string line: kToPrint) {
        cout << " \t";
        for (char ch: line) {
            if (ch == '\\') cout << "\\\"";
            else if (ch == '\\') cout << "\\\"";
            else cout << ch;
        }
        cout << "\t," << 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?

$M =$ “On input w :

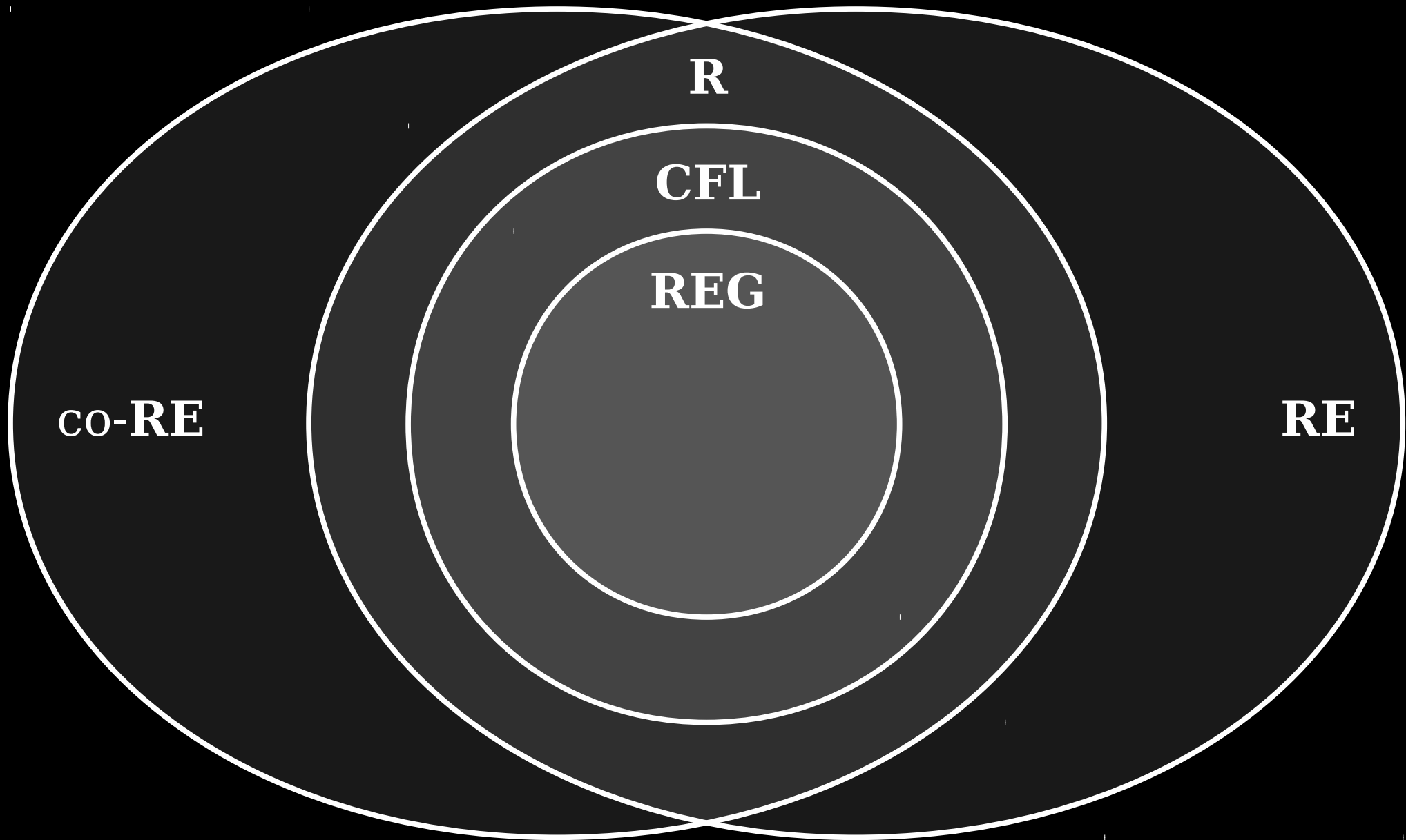
- Have M get its own description, $\langle M \rangle$.
- *Decide* whether M will accept w .
- If M will accept w , choose to reject w .
- If M will not accept w , choose to accept w .”

Okay... maybe we can't decide
or recognize everything.

Can we at least verify or refute everything?

$$\mathbf{EQ}_{\text{TM}} \notin \mathbf{RE}$$

$$\mathbf{EQ}_{\text{TM}} \notin \mathbf{co-RE}$$



Wow. That's pretty deep.

So... what can we do efficiently?



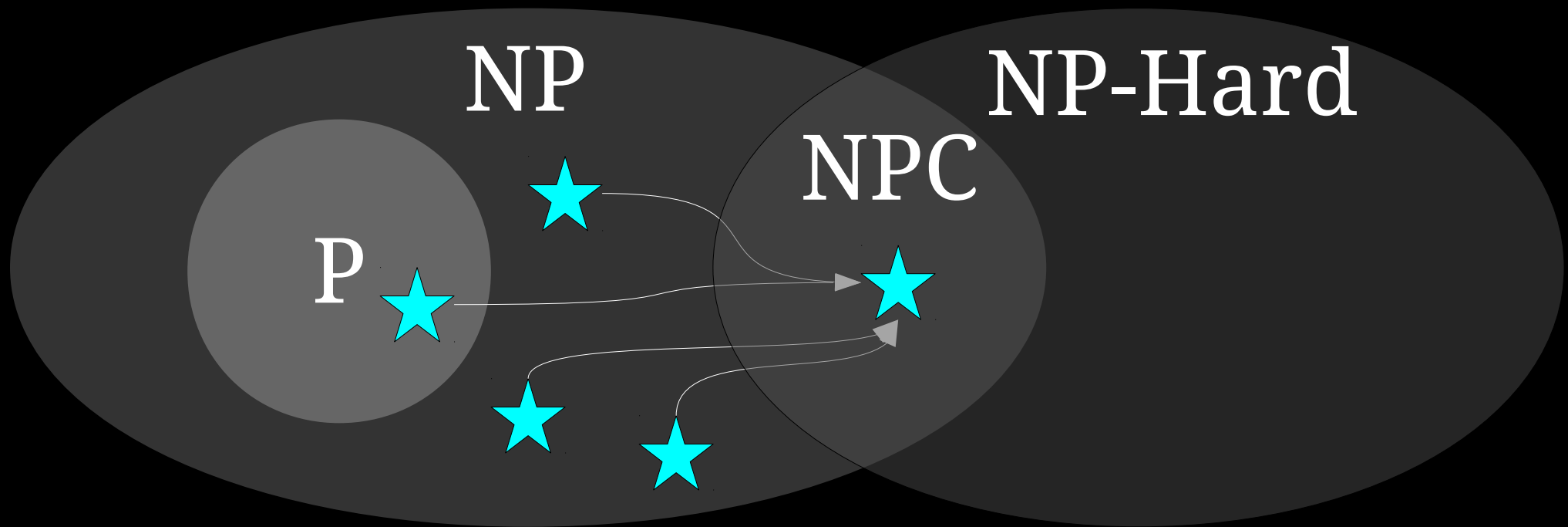
NP

The image features the letters 'N' and 'P' in a large, white, serif font against a solid black background. Behind the letter 'N' is a gray, stylized graphic of a human eye, with concentric circles representing the iris and pupil. Behind the letter 'P' is a gray, stylized graphic of a hand with fingers spread, positioned as if reaching towards the letter.

So... how are you two related again?

No clue.

But what do we know about them?



Congratulations on making it this far!

What's next in CS theory?

Formal languages

```
graph TD; A[Formal languages] --> B[What problems can be solved by computers?]; B --> C[Regular languages<br/>Context-Free Languages<br/>R, RE, and co-RE<br/>P and NP]; B --> D[DFAs<br/>NFAs<br/>Regular Expressions<br/>Context-Free Grammars<br/>Recognizers<br/>Deciders<br/>Verifiers<br/>NTMs<br/>Corecognizers<br/>Poly-time TMs/NTMs/Verifiers];
```

What problems can
be solved by computers?

Regular languages
Context-Free Languages
R, **RE**, and co-**RE**
P and **NP**

DFAs
NFAs
Regular Expressions
Context-Free Grammars
Recognizers
Deciders
Verifiers
NTMs
Corecognizers
Poly-time TMs/NTMs/Verifiers

Function problems (CS254)
Counting problems (CS254)

What problems can
be solved by computers?

Interactive proof systems (CS254)
Approximation algorithms (CS261/361)
Average-case efficiency (CS264)
Randomized algorithms (CS265/254)
Parameterized complexity (CS266)
Communication complexity (CS369E)

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)
- Average-case efficient algorithms (CS264)
- Randomized algorithms (CS265)
- Parameterized algorithms (CS266)
- Geometric algorithms (CS268)
- Game-theoretic algorithms (CS364A/B)

Where does CS theory meet CS practice?

Finite state machines
Regular expressions
CFGs and programming languages
Password-checking
Autograding
“This program is not responding”
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?

Hardness results for easy problems
(Virginia Williams)

Algorithms \cap Game theory
(Tim Roughgarden)

Learning patterns in randomness
(Greg Valiant)

Optimizing programs... randomly
(Alex Aiken)

Computing on encrypted data
(Dan Boneh)

Interpreting structure from shape
(Leonidas Guibas)

Lower bounds from upper bounds
(Ryan Williams)

So many options – what to do next?

Interested in trying out CS?
Continue on to CS109!

Really enjoyed this class?
Give CS154 a try!

Want to see this material come to life?
Check out CS143!

Want to just go write code?
Take CS107!

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

A Final “Your Questions”

CS theory is all about asking what's possible in computer science.

***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!***

What We've Covered

- Sets
- Proof Techniques
- Induction
- Graphs
- Logic
- Pigeonhole Principle
- Functions
- Relations
- DFAs
- NFAs
- Regular Expressions
- Closure Properties
- Nonregular Languages
- CFGs
- Turing Machines
- **R**, **RE**, and co-**RE**
- The Recursion Theorem
- NTMs and Verifiers
- Unsolvable Problems
- Reductions
- Time Complexity
- **P**
- **NP**
- **NP**-Completeness

Final Thoughts