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!***
- ***Please evaluate this course on Axess!*** Your feedback really does make a difference.

Final Exam Logistics

- Final exam is Wednesday, March 16 from 3:30PM – 6:30PM, location TBA.
  - We'll announce locations over the weekend.
  - Cumulative exam – all topics are fair game for the exam – but the questions are calibrated based on the skills you've developed on the problem sets. The exam is balanced at around 50% topics from PS1 – PS6 and 50% topics from PS7 – PS9.
- Exam is closed-book, closed-computer, and limited-note. You can have a double-sided, 8.5” × 11” sheet of notes with you when you take the exam.

Preparing for the Exam

- By popular demand, we'll be holding a practice final exam on ***Saturday at 2PM in 320-105***. We'll use the “Pracitce Final Exam” handout up on the course website as the practice exam, so please don't look at it if you're thinking you're going to attend. ☺
- We will be holding limited office hours next week. Check the OH calendar for details.
- Please feel free to ask us questions on Piazza over the next couple of days. We want you to understand this material!
- EPP8 – EPP10 are cumulative review problems. Solutions are available in the Gates building.

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 should be sure we have some rules about reasoning itself.

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.

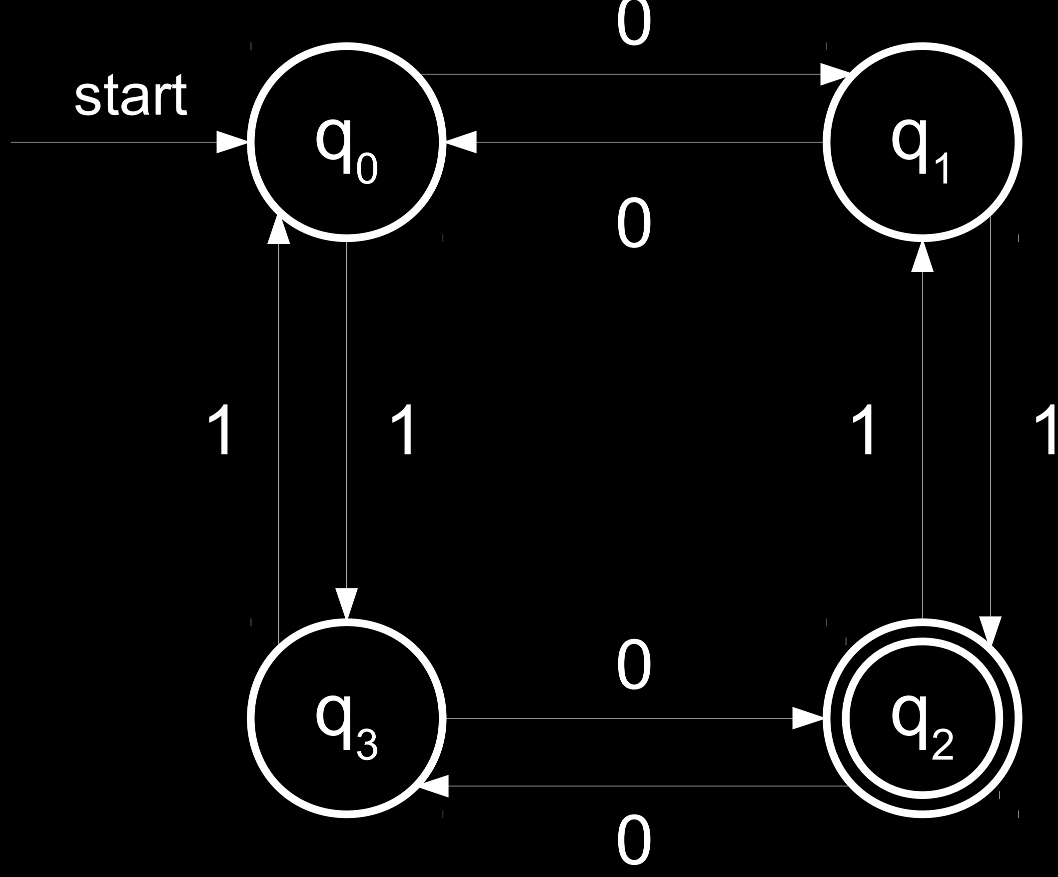
We also need to prove things about processes that proceed step-by-step.

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!

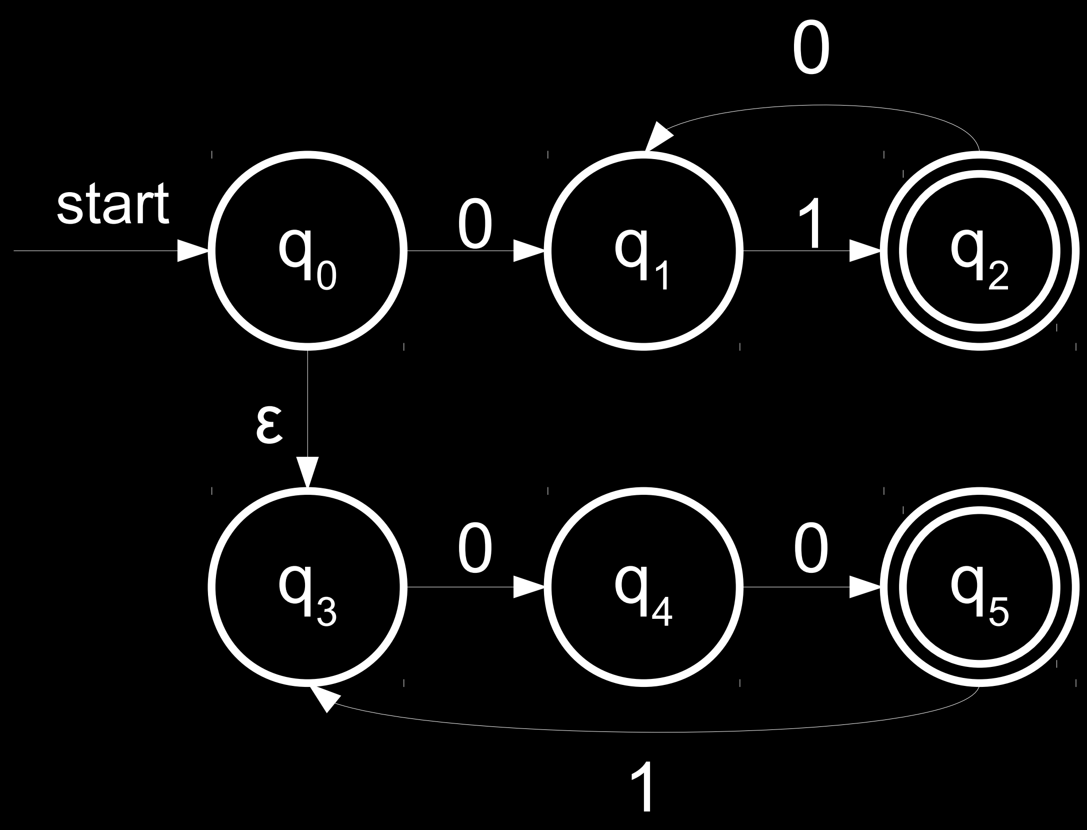

start
q0
q1
q2
q3
0
0
0
0
1
1
1
1

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***.

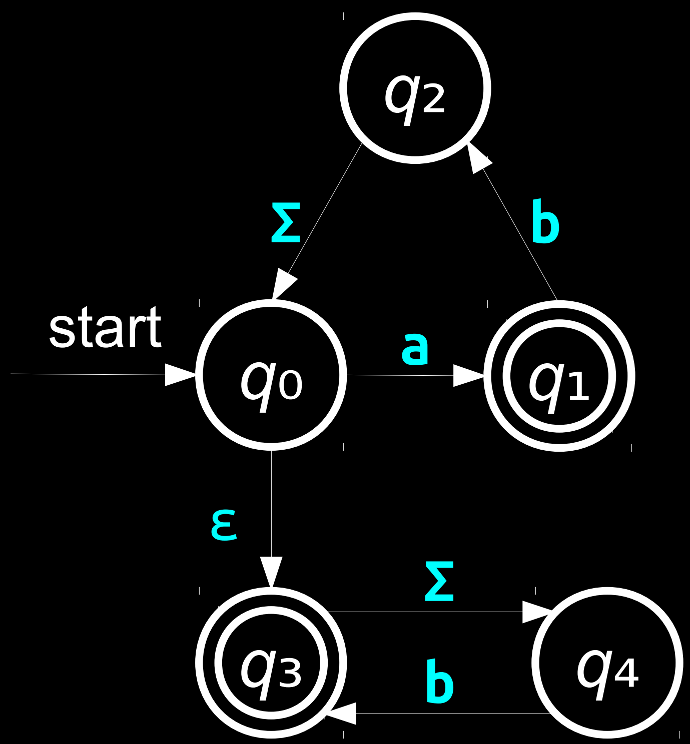
I wonder what other
machines we can make?



start
q0
q1
q2
q3
q4
q5
0
1
0
ε
0
0
1

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

I wonder if they're more powerful?



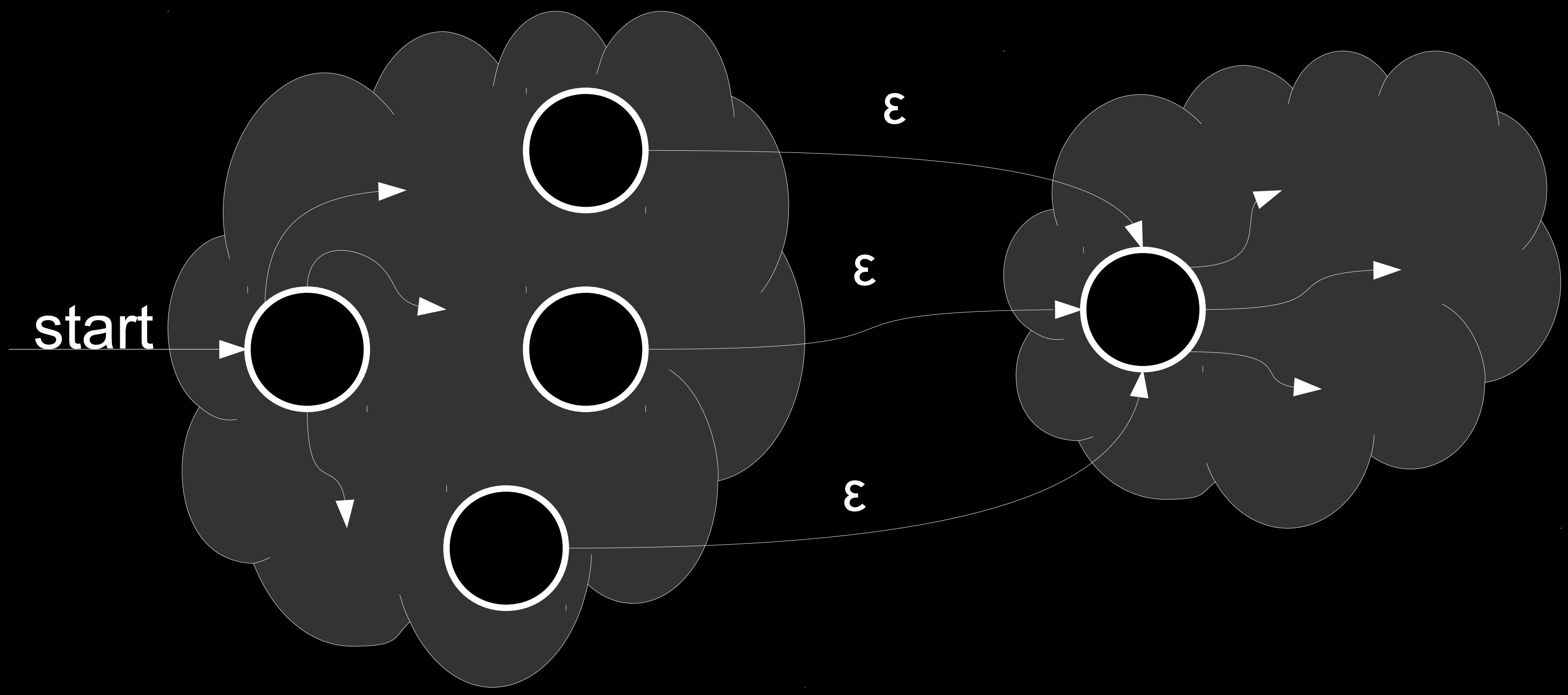
	a	b
*$\{q_0, q_3\}$	$\{q_1, q_4\}$	$\{q_4\}$
*$\{q_1, q_4\}$	$\emptyset$	$\{q_2, q_3\}$
$\{q_4\}$	$\emptyset$	$\{q_3\}$
*$\{q_2, q_3\}$	$\{q_0, q_3, q_4\}$	$\{q_0, q_3, q_4\}$
*$\{q_3\}$	$\{q_4\}$	$\{q_4\}$
*$\{q_0, q_3, q_4\}$	$\{q_1, q_4\}$	$\{q_3, q_4\}$
*$\{q_3, q_4\}$	$\{q_4\}$	$\{q_3, q_4\}$
$\emptyset$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

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

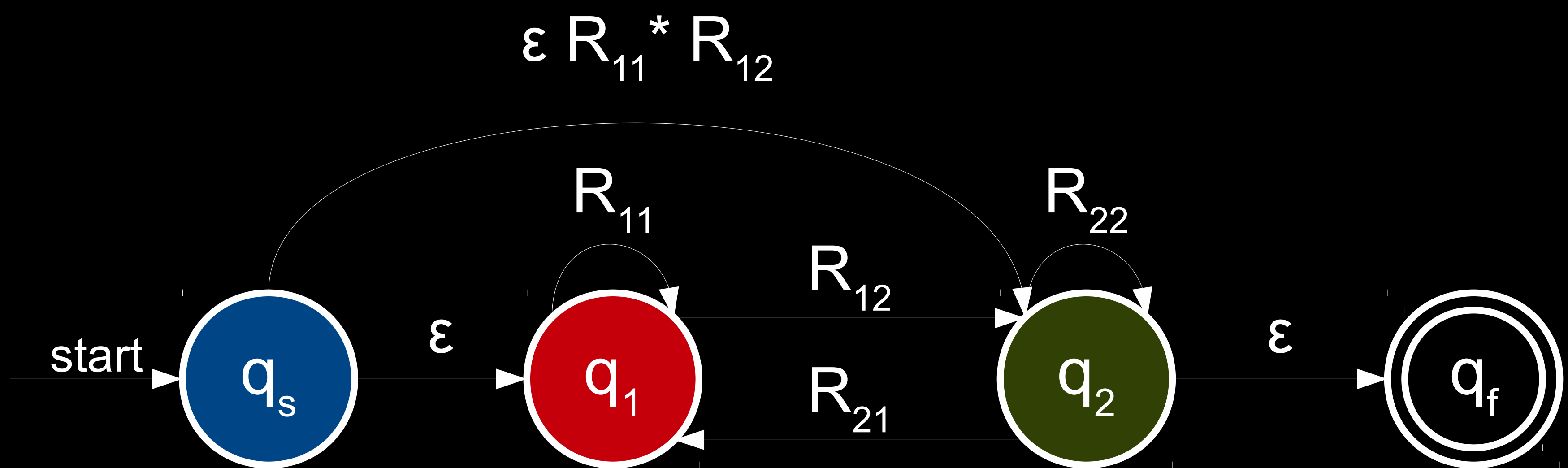
Cool! Since we can glue machines together, we can glue languages together as well.

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
qs
ε
q1
R21
q2
ε
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?

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

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?

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

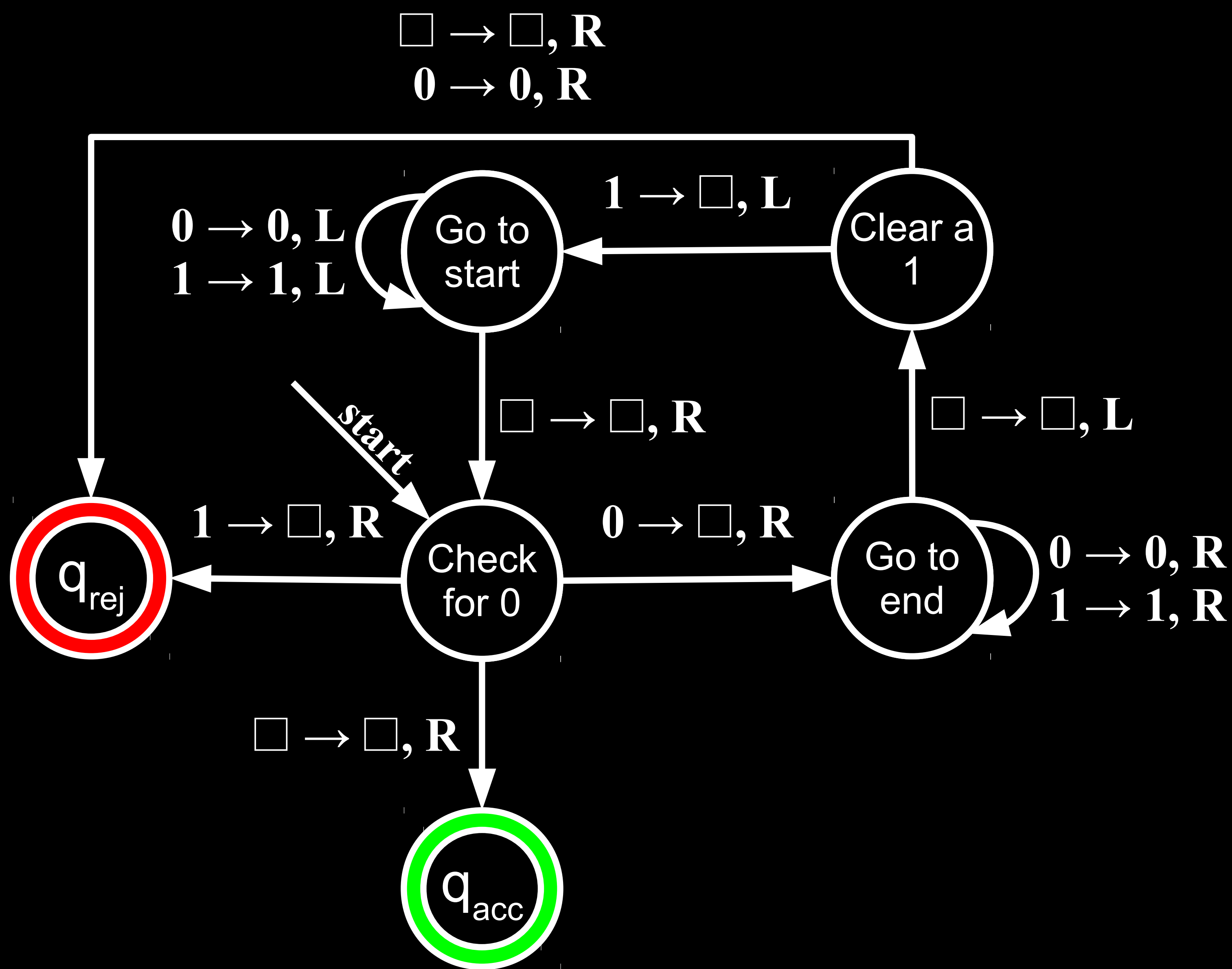
Awesome!

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
q_rej
Check for 0
Go to end
0 → 0, R
1 → 1, R
□ → □, 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?

U_{TM} = "On input ⟨*M*, *w*⟩, 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 ⟨*M*, *w*⟩.

If *M* rejected *w*, then U_{TM} rejects ⟨*M*, *w*⟩.

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?

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

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

string mySource() {
  string result;
  for (string line: kToPrint) {
    if (line == "@") {
      for (string inLine: kToPrint) {
        result += "  R\"(" + inLine + ")\",\n";
      }
    } else {
      result += line + '\n';
    }
  }
  return result;
}

int main() {
  cout << mySource() << 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?

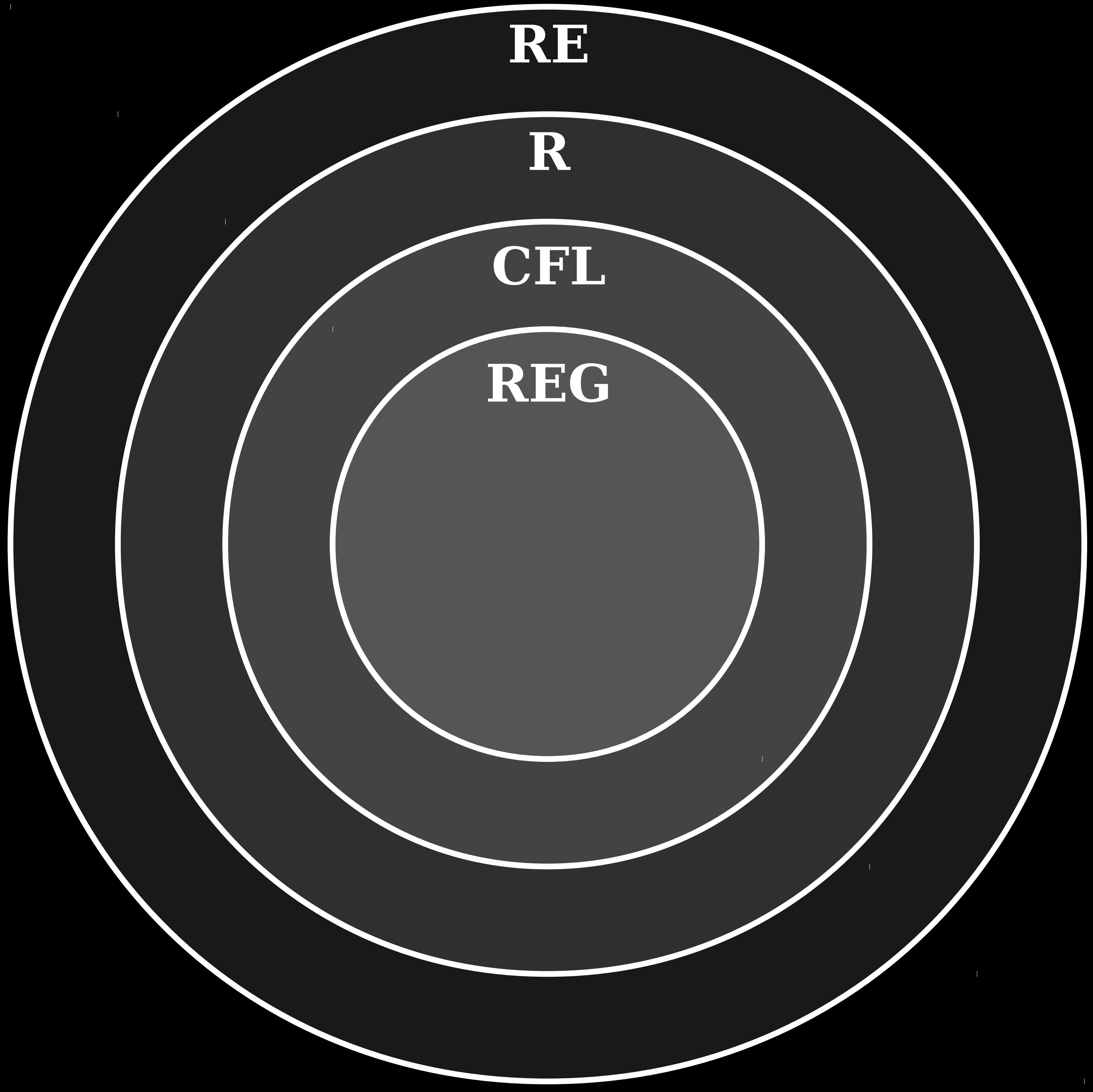
```
int main() {
   string me = mySource();
   string input = getInput();

   for (each string c) {
      if (imConvincedWillLoop(me, input, c) {
         accept();
      }
   }
}
```

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
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?

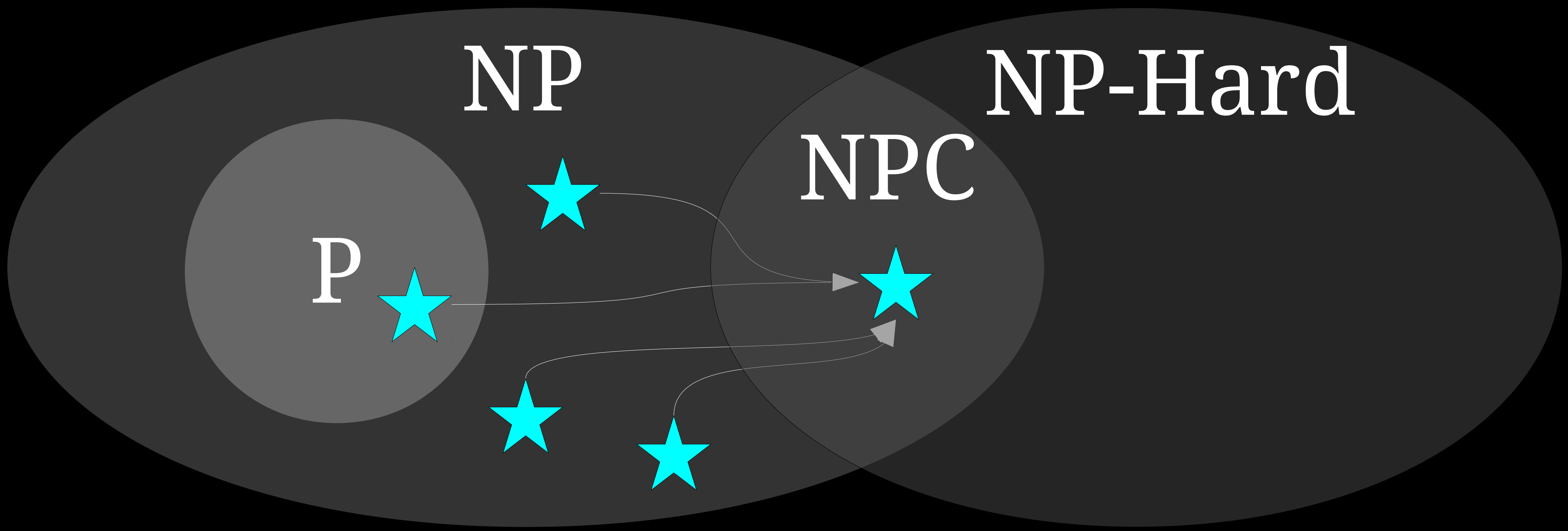
P

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)
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
"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 ∩ Game theory

(Tim Roughgarden)

Learning patterns in randomness

(Greg Valiant)

Approximating NP-Hard Problems

(Moses Charikar)

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?

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 108!

Want to just go write code?

Take CS107!

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

A Final “Your Questions”

“What's the wildest thing you did as an undergrad?”

Yeah… it's not repeatable.
Sorry about that!

“You seem like you do an excellent job of keeping your brain active and engaged post-school. I'm graduating and would like to do the same, any suggestions for how to keep learning?”

You'd be amazed how much you pick up and learn just purely by doing your job. Everyone I know in industry is constantly learning new things. If you find yourself in a job that isn't intellectually stimulating, see if you can find a better one.

Also, read a lot. Get a subscription to the New Yorker, The Atlantic, or The Economist and make time to read it. Listen to interesting podcasts. Take classes on Coursera or edX on topics you know nothing about. Find YouTube channels on things that interest you. Talk to people with different backgrounds and experiences. The world is an exciting place!

"How can we keep in touch with you? Realistically, how much capacity do you have for staying in convo with your students? What recommendations do you have for building and maintaining actually meaningful relationships with faculty?"

In a 385-person class there's just no way that I can stay in touch with everyone, but that shouldn't stop you from reaching out to me and keeping in touch! I'm still in touch with a bunch of my students from when I first started teaching. Stop on by and let me know how you're doing - I'd love to hear what you're up to!

“Any TV show recommendations for those of us looking for something to binge on over Spring Break?”

If you haven't watched “The Wire,” go watch it. It's phenomenal – I think it's the best TV show ever made. David Simon's new show “Show Me a Hero” is also fantastic. You can watch the whole thing in a day.

For really good but really dark television, watch “How to Get Away with Murder” or “Mr. Robot.” For something hilarious and irreverent, watch “One-Punch Man.” Or just go on YouTube and watch all the Steph Curry highlights.

OR GO OUTSIDE. It's Spring Break!

“Do you think computers have the capacity to be creative?”

I do. I don't think we've quite gotten there yet, but we're getting close. A lot of the new work done with deep neural nets (neural artwork, AlphaGo) is really interesting and I'm super excited to see where they take us.

"Hi Keith- I'm worried about retaining material from past classes (wow, I guess I forgot all of chem!) and I was wondering which you think is more important: information recall and easily acceding the material, or the general process of learning to learn."

Most people honestly forget most of what they learned as an undergrad. What's important is that you know what's out there and the key ideas. That makes it super easy to pick things back up later on and to keep exploring beyond what you learned the first time around.

Anything else?

Final Thoughts

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?