AUTOMATING COMPUTER SECURITY

Tyler Nighswander
@tylerni7

CHAPTER 1

WHY WE NEED COMPUTERS

MOTIVATION

- We have too much software
  - For “good” reasons we wrote a lot of it in C/C++
    - And it is all full of bugs
- Humans alone scale poorly
- Scaling is one of very few things computers do well
  - But can they “automate” security well?

CYBER GRAND CHALLENGE

- Offensive/defensive tests on stripped binaries
- Objectively scored by 3rd party referees
- Best public demo of state-of-the-art in automated analysis

THE BEGINNINGS

- Defense Advanced Research Projects Agency (DARPA)
  - Helped create Internet, GPS, stealth, onion routing, etc.
  - Famously hosts “challenges”

CMU Sandstorm

Google Self Driving Car

THE BEGINNINGS

- Idea: host one of these challenges for computer security
  - Model after Capture the Flag competition
  - Offer USD$750,000 to top 7 teams to qualify
  - Offer USD$2,000,000 to top team in final contest
  - No intellectual property given up by competitors
  - Cyber Grand Challenge

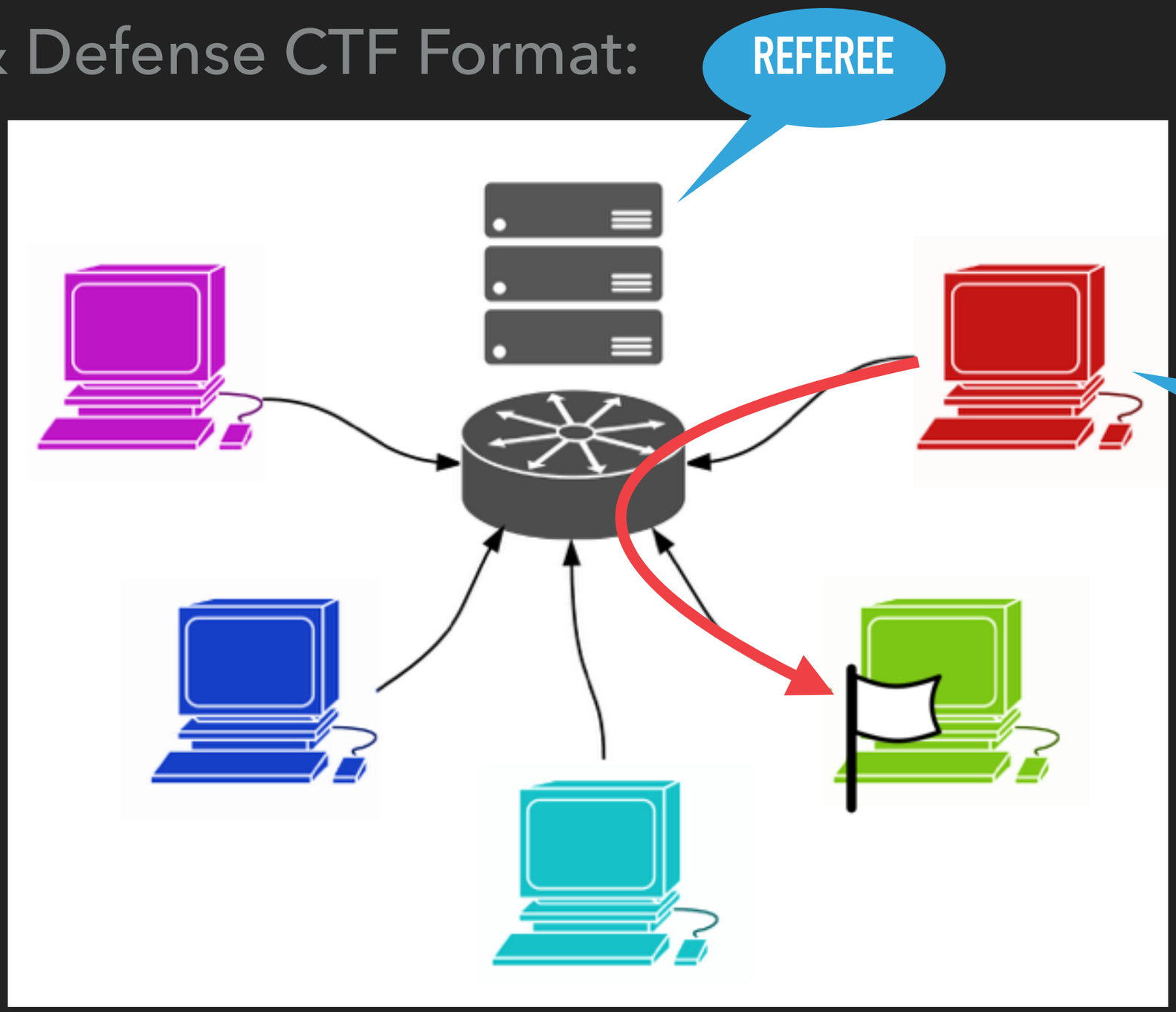
ABOUT ME

- Playing in CTFs for about 7 years with PPP
  - Top ranked CTF team in the world (sometimes) !
- Graduated from CMU
  - Winner/finalist in several DARPA Challenges
- Researcher at ForAllSecure
  - Based on a decade of research in automated security

Perfect fit !

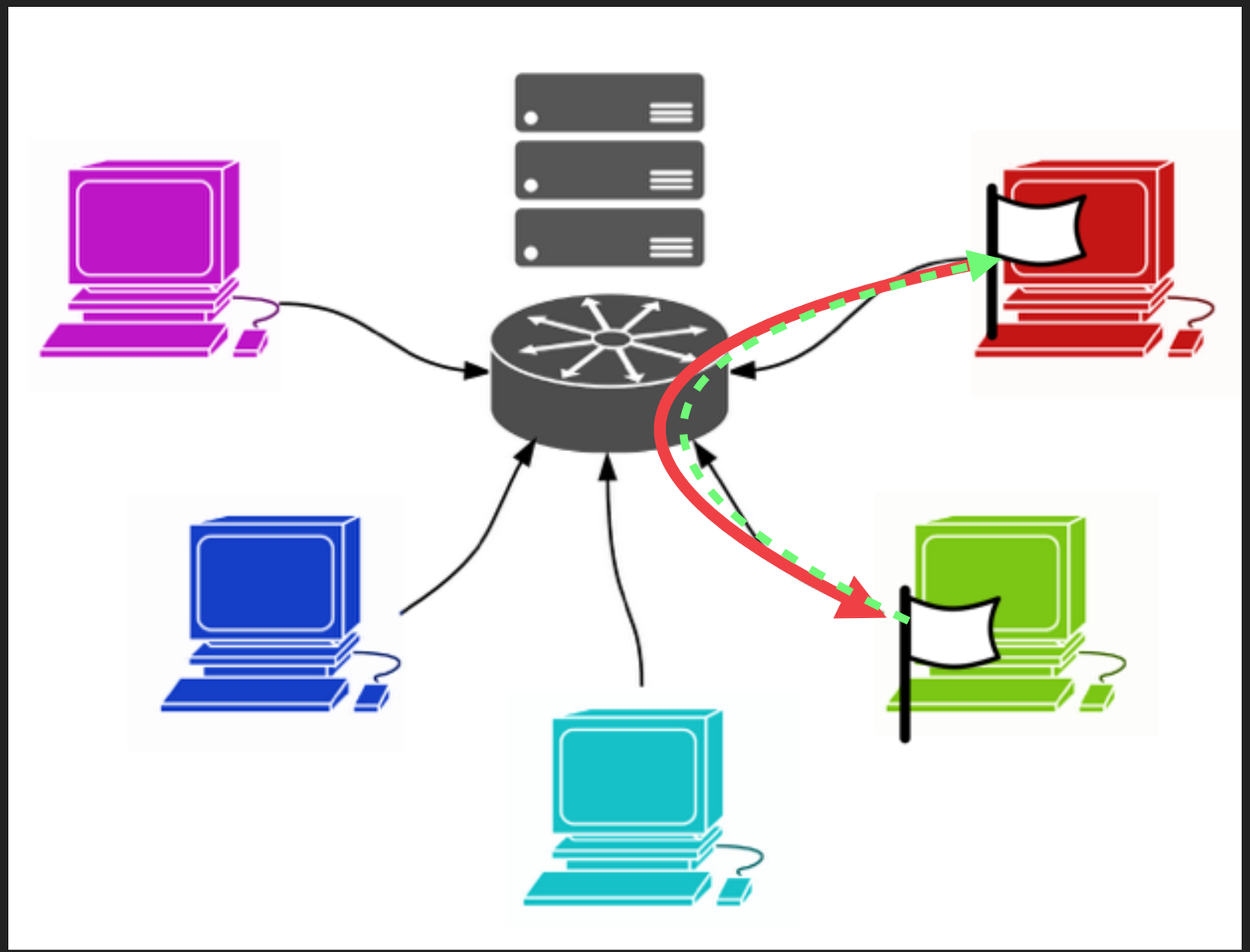
ABOUT CGC

- Attack & Defense CTF Format:



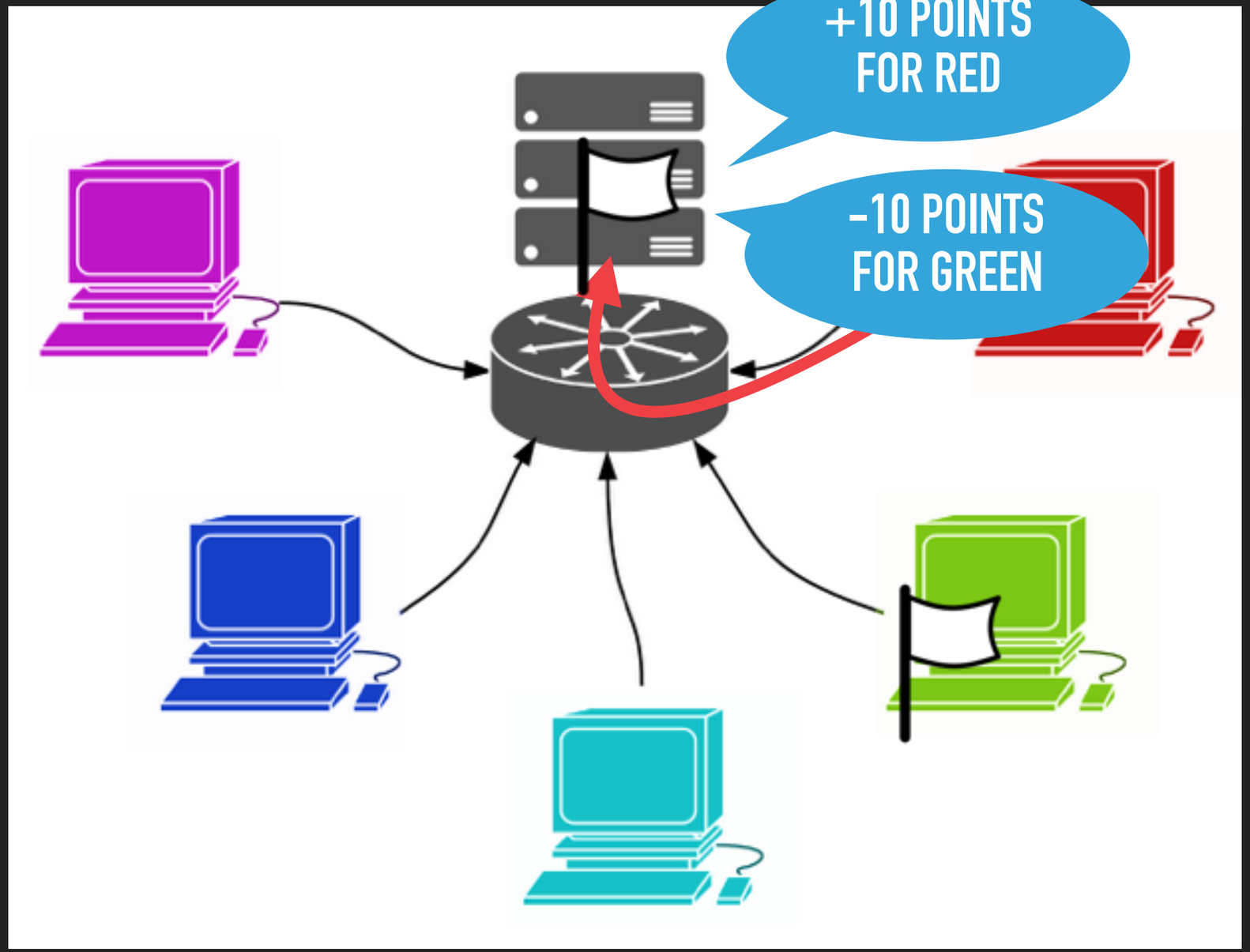
ABOUT CGC

- Attack & Defense CTF Format:



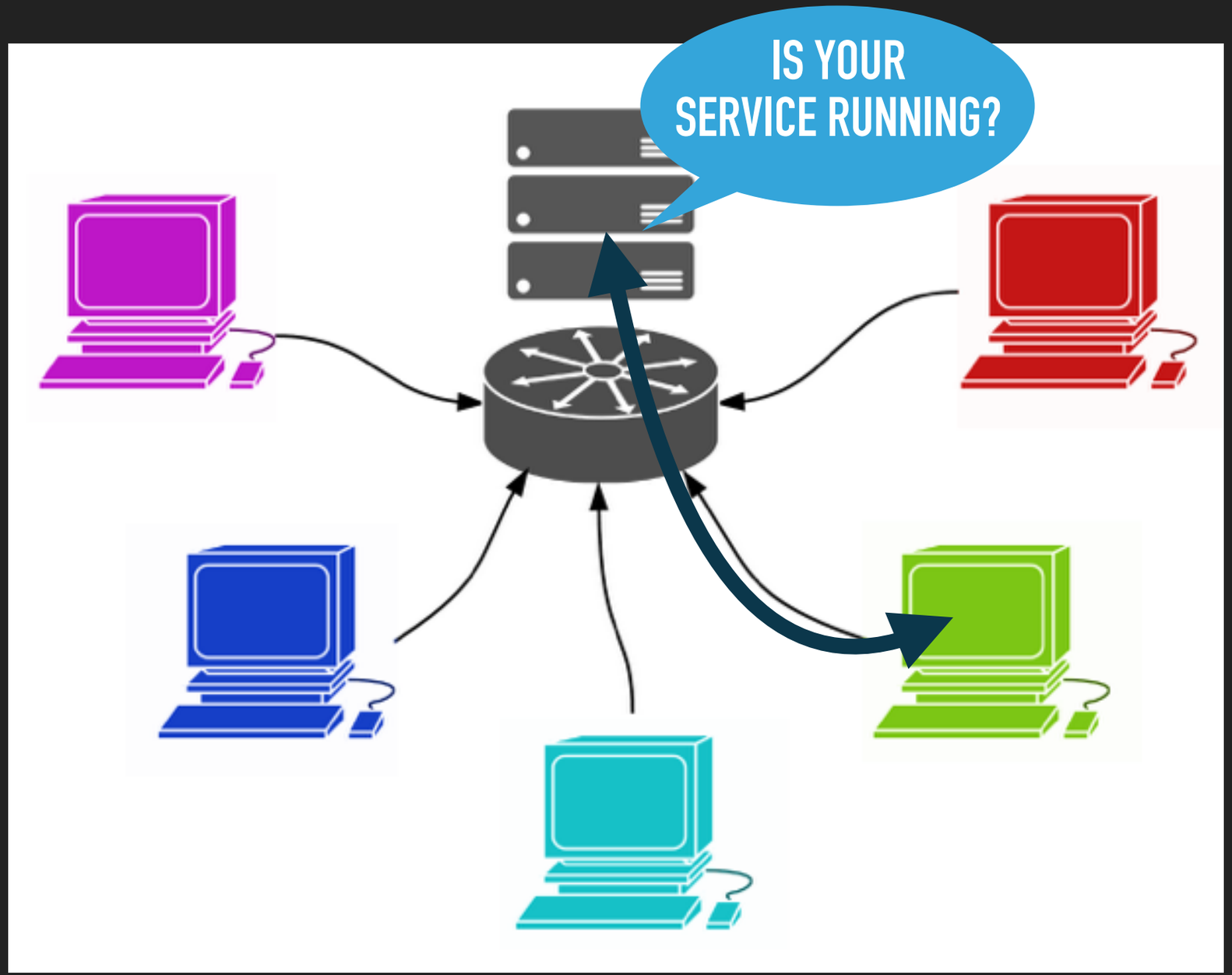
ABOUT CGC

- Attack & Defense CTF Format:



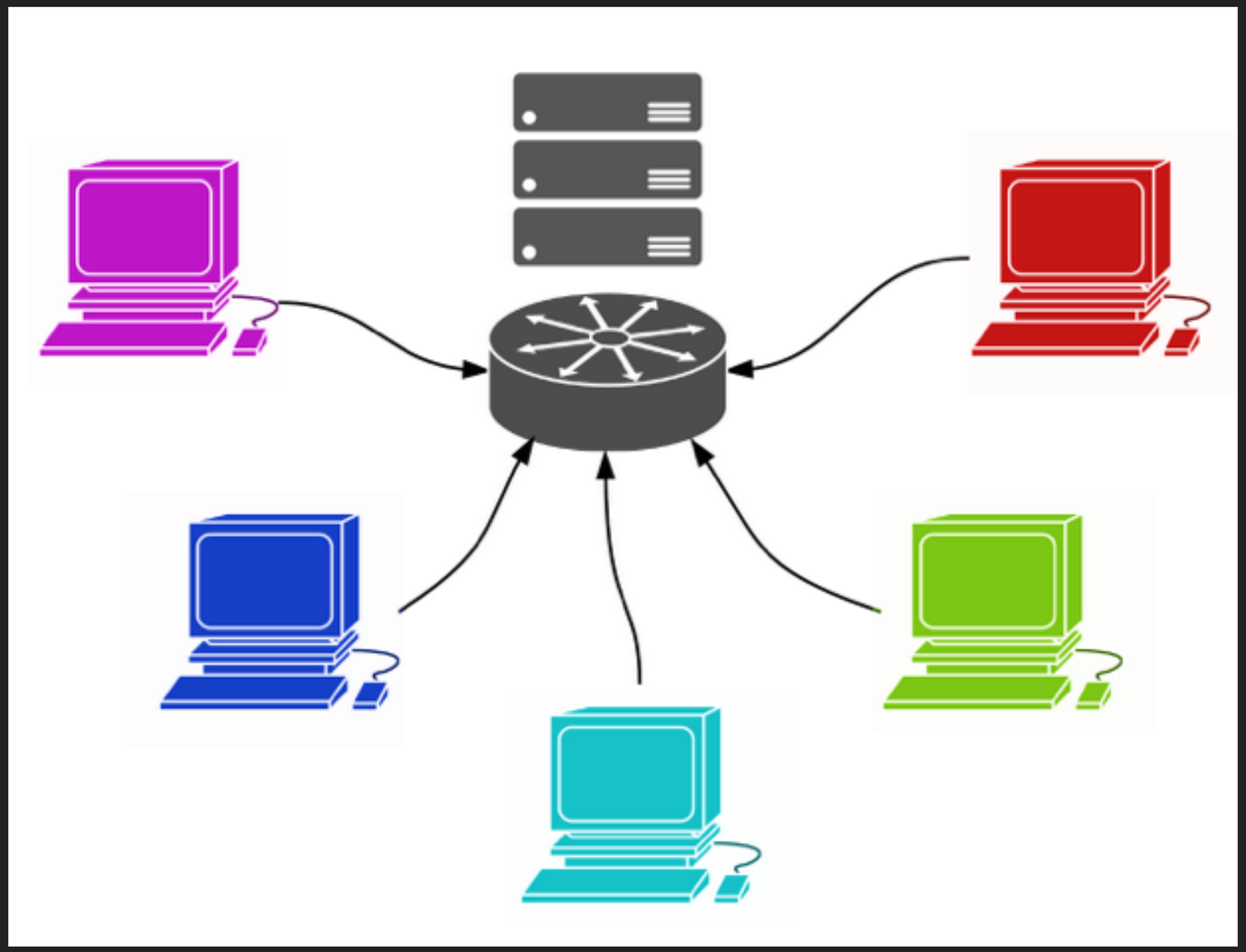
ABOUT CGC

- Attack & Defense CTF Format:



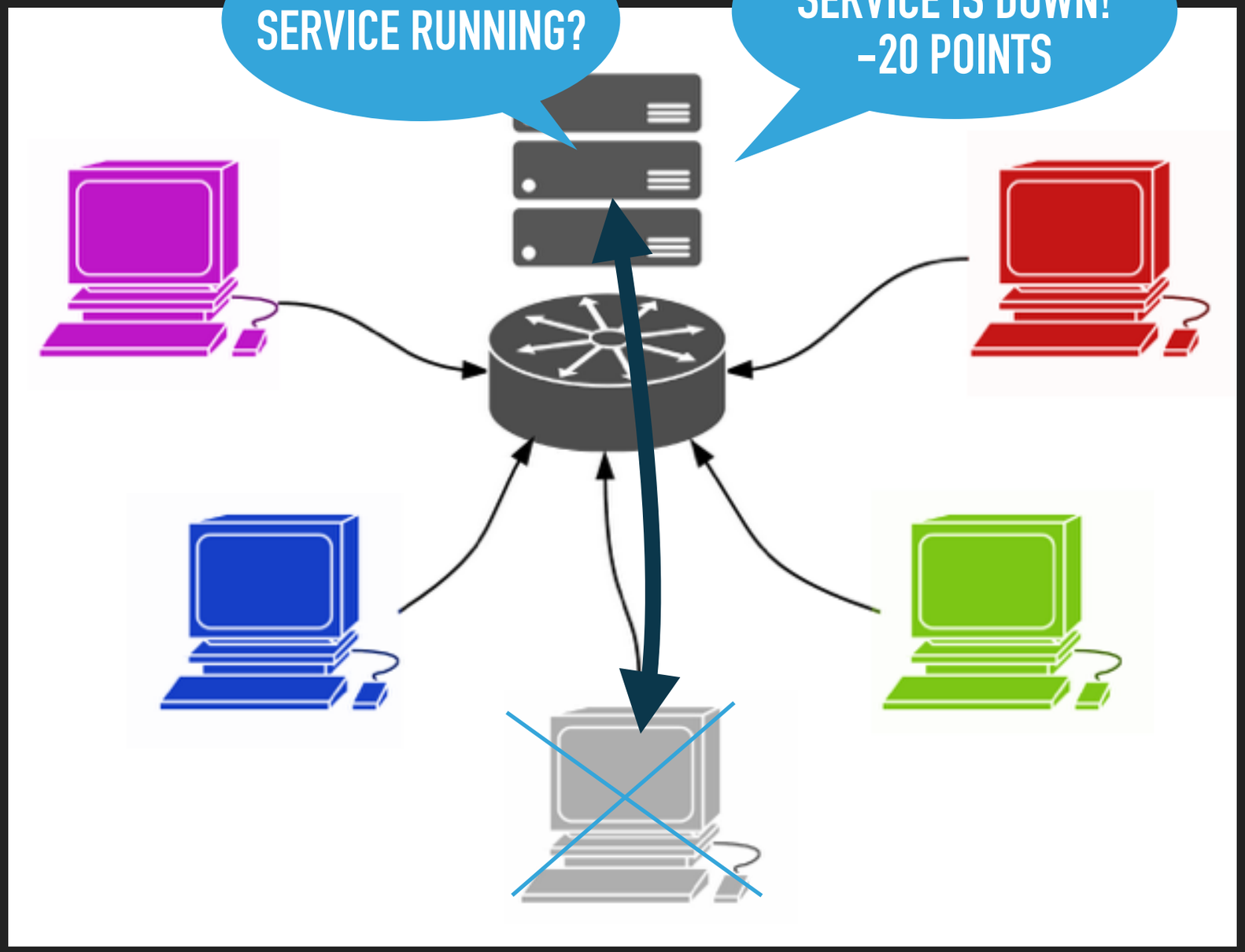
ABOUT CGC

- Attack & Defense CTF Format:



ABOUT CGC

- Attack & Defense CTF Format:

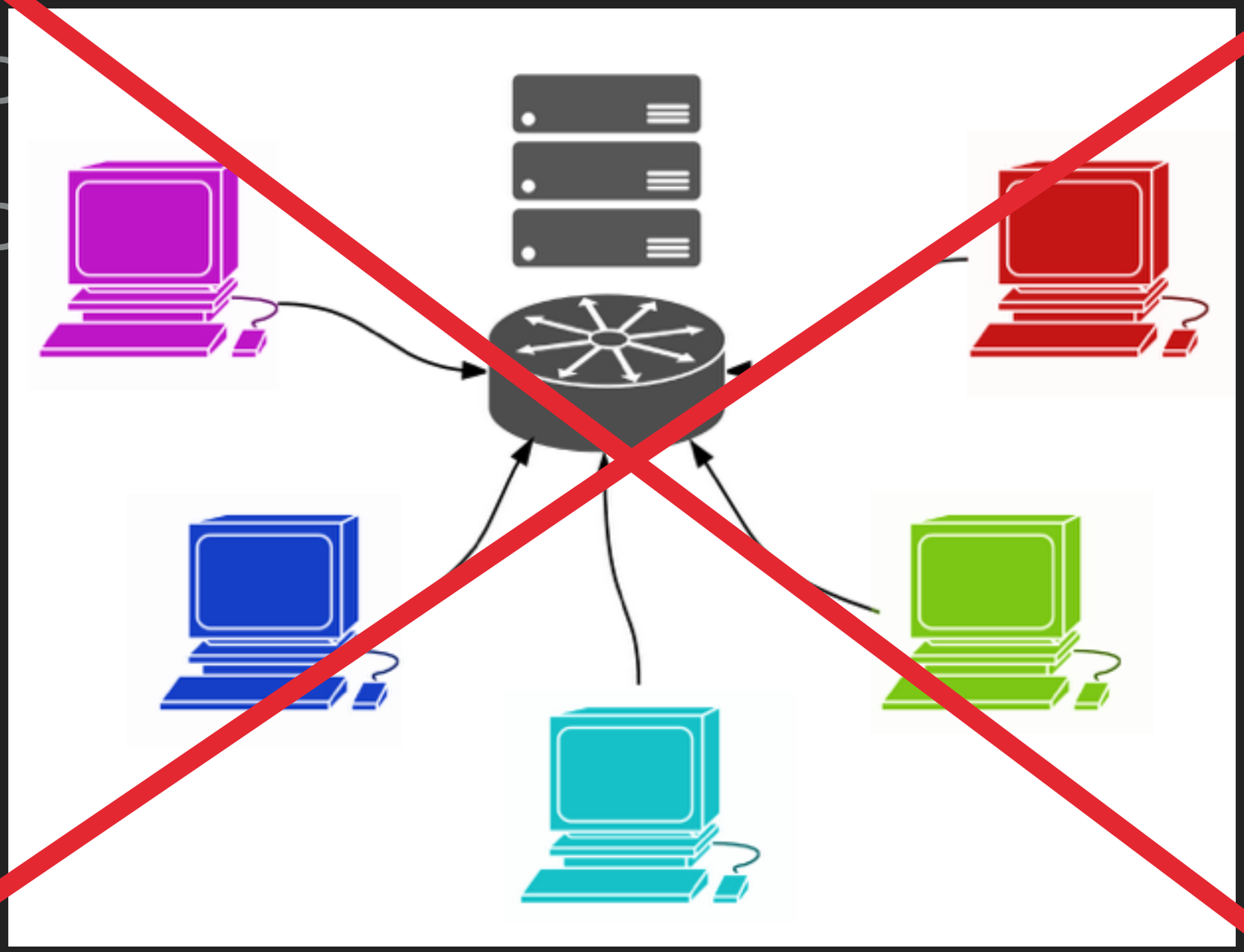


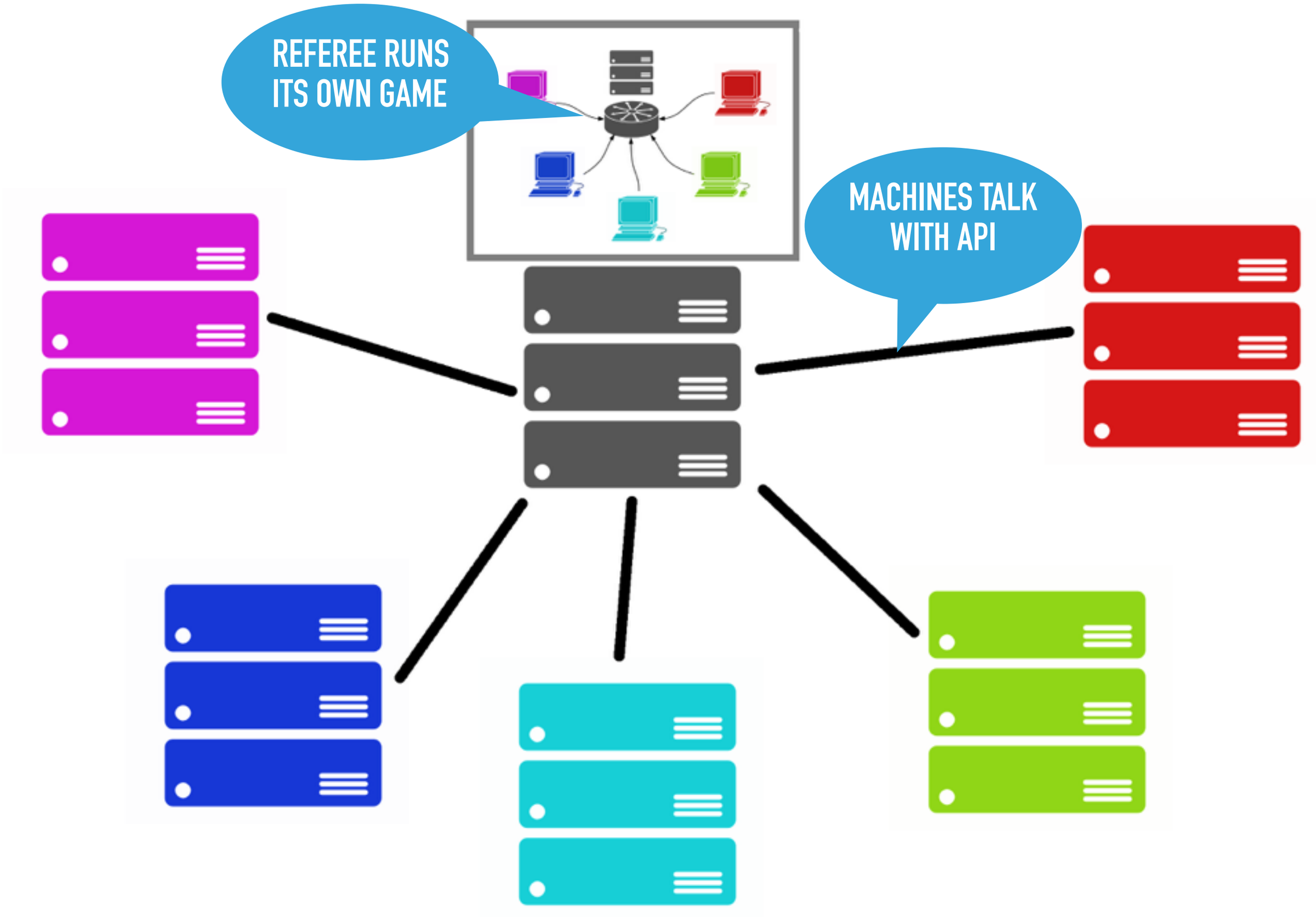
ABOUT CGC

- Scores based on:
  - Security:
    - Lose points if anyone exploits your system
  - Offense:
    - Gain points if you exploit other teams
  - Availability:
    - Lose points if performance decreased
    - Lose points if functionality is decreased

ABOUT CGC

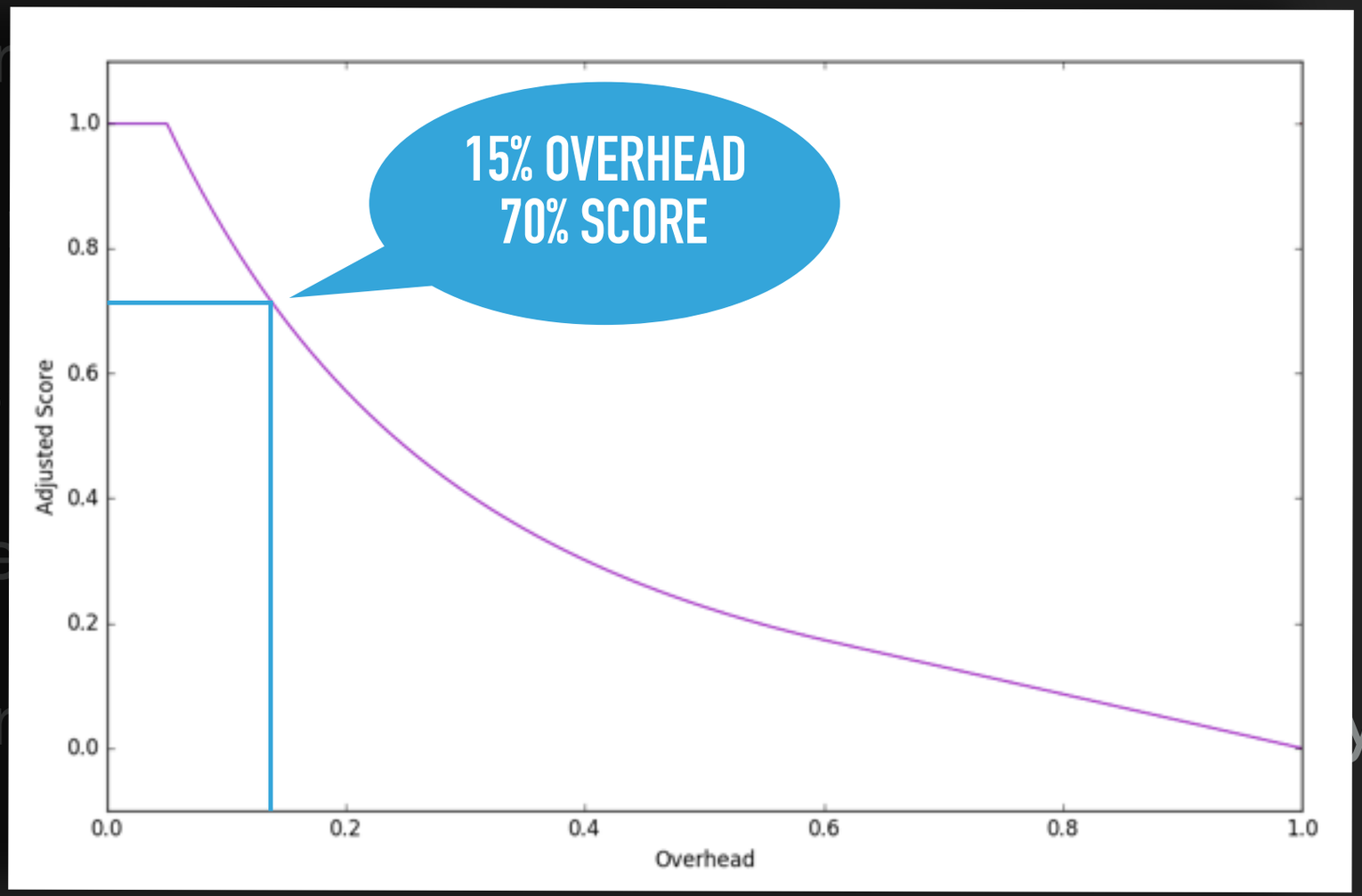
- To make scoring more fair+secure, DARPA runs everything
  - Give D
  - Give D




REFEREE RUNS
ITS OWN GAME
MACHINES TALK
WITH API

ABOUT CGC

- Availability scoring very strict!
- Lose poi
  - Measu programs
  - Pollers
  - No spe
- Lose poi y 5%



ABOUT CGC

- Create a new executable format, modeled after ELF
- Seven system calls:
  - exit
  - transmit
  - receive
  - fdwait
  - allocate
  - deallocate
  - random
- Makes running foreign code more safe
- Makes automated analysis easier vs Linux/Windows/OSX

ABOUT CGC

- No file system or fork or exec… what is an exploit?
- Type 1:
  - Control of register and instruction pointer
- Type 2:
  - Leak of sensitive data

Codenomicon/Synopsys

ABOUT CGC

- With 7 system calls, still plenty of complexity!
- Programs can have:
  - Several binaries communicating
  - User-space threads
  - Non-determinism, nonces, checksums
  - C or C++ code compiled to x86 assembly

ABOUT CGC

- Easily able to recreate several famous bugs:
  - Heartbleed
  - LNK exploit (Stuxnet infection vector)
  - Crackaddr bug
  - SQL Slammer
- All inside the more simple CGC framework

WHY WE NEED COMPUTERS

MAY

- S
  -
- C
- L

MAYHEM: BUG FINDING

- Symbolic execution
  - State of the art technique in academia
  - Represent program as symbolic expressions
  - Use SAT/SMT solvers to generate inputs

```
int main(int argc, char** argv) {
  if (argc < 3)
    return -1;
  int a = atoi(argv[1]);
  int b = atoi(argv[2]);
  if (a == 42) {
    if (b == 31337) {
      puts("You made it!");
    }
    retu
  }
  return 0;
}
```

TREAT SYMBOLICLY!

LEN(ARGV) >= 3

LEN(ARGV) >= 3

RETURN -1

LEN(ARGV) >= 3
NOT(42 == ATOI(ARGV[1])

PATH
AINTS

42 == ATOI(ARGV[1])

31337 == ATOI(ARGV[2])

PRINT

LET'S GO HER

LEN(ARGV) >= 3
NOT(42 == ATOI(ARGV[1])
-> ARGV={"A","A","A",0}

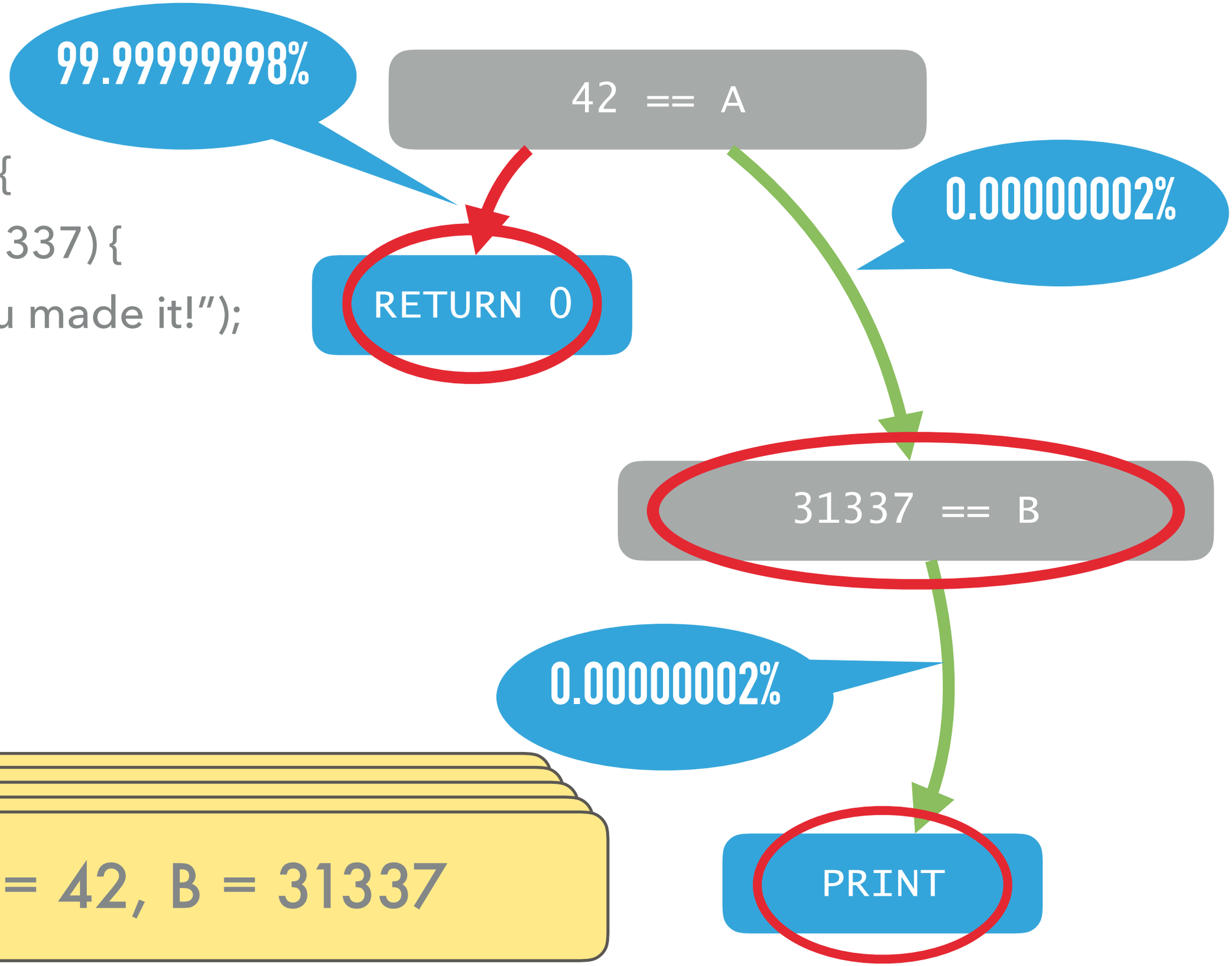
RETURN 0

RETURN 1

MAYHEM: BUG FINDING

- Fuzzing
  - Standard technique in industry
  - Send concrete values to a program and execute it
  - Can also “instrument” the program
    - Reveal internal state about what it did with inputs
    - Possible using a variety of techniques…

```
if (a == 42){
  if (b == 31337){
    puts(“You made it!”);
  }
  return 1;
}
return 0;
```



A = 42, B = 31337

MAYHEM: BUG FINDING

- Proper instrumenting can help fuzzing explore deeper
- Instrumentation for fuzzing:
  - Efficient ways to instrument are still an area of research
  - QEMU, PIN, DynInst, DynamoRIO, etc ?
    - All incur a 2-10x slow-down!
  - Making a fuzzer too “smart” only makes it slow!

MAYHEM: BUG FINDING

Fuzzing	Symbolic Execution
very easy to setup and run	usually very complex
10-10k executions/second	1-∞ seconds/“execution”
doesn’t “know” anything	“understands” the program
light on resources	very resource intensive
explore shallow paths fast	explore deep paths slowly
stalling	path explosion

MAYHEM: BUG FINDING

- Fuzzing and Symbolic execution
  - Combine the best of both worlds
    - Fuzz a program to uncover shallow paths
    - Use symbolic execution to find deep, complex paths
- To work together, simply share inputs
  - Keep track of which inputs did something “interesting”
  - Share the interesting inputs

MAYHEM: BUG EXPLOITING

- After you have a bug, need to turn it into an exploit!
- Again, use symbolic execution:
  - In addition to *path constraints* ask for EIP set to value
    - Can even use SMT solver in exploit script!
- A hybrid approach here works as well:
  - Use fuzzing and sym-ex on crashes to get new crashes
  - Use symbolic execution to generate exploits
- Can generate pretty complex exploits!

```
int copy( fileSystemType *fs, char *cmdline, unsigned int owner ) {
  char sourcefile[FILENAME_SIZE];
  char destfile[FILENAME_SIZE];
  int x;

  // skip over leading whitespace characters
  while ( *cmdline != 0 && isspace(*cmdline) )
    ++cmdline;

  // if we hit the end of the line there were no filenames specified
  if ( *cmdline == 0 ) {
    return ERROR_BAD_PARMS;
  }

  x = 0;
  while ( *cmdline != 0 && !isspace(*cmdline) ) {
    if ( x < FILENAME_SIZE ) {
      sourcefile[x] = *cmdline;
    }

    ++cmdline;
    ++x;
  }

  sourcefile[x] = 0;
  ...
}
```

```
int main(void) {
  char command[1024];
  while (1) {
    bzero(command, 1024);
    getline(command, 1024);

    i = 0;
    while (command[i] != ' ' && i < strlen(command)) {
      ++i;
    }
    command[i] = 0;

    if ( strcmp(command, "list") == 0 ) {
      ...
    }
    else if ( strcmp( command, "copy" ) == 0 ) {
      retcode = copy( currentFS, command+i+1, currentUser );
    }
    ...
  }
}
```

MAYHEM: BUG EXPLOITING

- Given ability to write a single null byte, generate exploit
- Basic exploit chain:
  - Overwrite lowest byte of saved EBP with 0
  - Stack frame above uses EBP-relative address for buffer
  - Future writes to that buffer will now clobber memory
  - Send a command to do another write, overflow buffer
  - Overwrite EIP, and win!
- Sound difficult?
  - Mayhem found and exploited this bug in 105 minutes

CGC: LESSONS LEARNED

- Computers *can be a lot* faster than any humans
  - (This should not be very surprising)
- Actually relevant to “real” systems
  - Mayhem is perfectly capable of running on Linux ELFs
- Computers can find more bugs than might be expected
  - This includes lots of things that humans might overlook

WHY WE REALLY NEED COMPUTERS SOON

- IoT : aka “who needs security? just put it on the internet”
  - Light-weight and cheap, so lots of C/C++
  - Only way to keep up with this will be automation
- Fuzzing is doing it’s job—slowly weeding out shallow bugs
  - Bug finders will need to step up their game!
- Fuzzing still finds tons of bugs…
  - Triaging them currently is not very good
- Automated systems can help with all of these things!

HUZZAH! COMPUTERS WILL SOLVE ALL OUR PROBLEMS!

Everyone

CHAPTER 2

WHY COMPUTERS NEED US

CGC: LESSONS LEARNED

- Computers *can be a lot* faster than any humans
  - (This should not be very surprising)
- Actually relevant to “real” systems
  - Mayhem is perfectly capable of running on Linux ELFs
- Computers can find more bugs than might be expected
  - This includes lots of things that humans might overlook
  - …But they’re still not magic, and miss lots
- Computers can’t exploit many tricky bugs
- Computers have difficulty scaling to very large programs

AUTOMATIC EXPLOIT GENERATION

- As we saw, it is possible to generate non-trivial exploits
- Not-easy-but-not-science-fiction:
  - Stack canaries? SMT solvers can (kind of) handle this!
  - Weird input restrictions? That's fine too!
  - Randomization? Ehh.. might be doable?
- Let's consider something much harder…
  - *Weird machines*

CVE-2015-0090

Adobe Font Driver in Microsoft Windows Server 2003 SP2, Windows Vista SP2, Windows Server 2008 SP2 and R2 SP1, Windows 7 SP1, Windows 8, Windows 8.1, Windows Server 2012 Gold and R2, and Windows RT Gold and 8.1 allows remote attackers to execute arbitrary code via a crafted (1) web site or (2) file

- Very cool bug(s) and exploit by @j00ru
  - (check out his Recon 2015 talk)
- tl;dr: kernel font parsing has insecure mini language!
- Exploit: create mini “program” to write out a ROP chain

AUTOMATIC EXPLOIT GENERATION

- To do this automatically:
  - ~~Ask an SMT solver to think really hard?~~
  - Distill state machine into simpler representation
  - Ask SMT solver to write ROP payload
- But how do we solve this generally?

AUTOMATIC EXPLOIT GENERATION

- Proper exploit generation ≈ program synthesis

> **Program synthesis** is a special form of automatic programming that is most often paired with a technique for formal verification. The goal is to construct automatically a program that provably satisfies a given high-level specification. In contrast to other automatic programming techniques, the specifications are usually non-algorithmic statements of an appropriate logical calculus.[1]

- Is this doable?
  - Yes!
- Is this a solved problem?
  - Far from it…

AUTOMATIC EXPLOIT GENERATION

- Weird machines
  - Charstring font exploits
  - Javascript exploits
    - ROP
    - Lots of other (less obvious) things
- Recognizing and exploiting these requires “creativity”
  - Creativity is not a computer’s strong point…

WHAT IS A BUG?

BUGS YOU KNOW ABOUT

BUGS YOU KNOW YOU DON'T KNOW ABOUT

BUGS YOU DON'T KNOW YOU DON'T KNOW ABOUT

THIS MIGHT CRASH IF A USER PUTS AN EMOJI IN [illegible]E!

I DON'T KNOW IF I HANDLED ALL THE EDGE CASES IN THIS STRING PROCESSING, IT MIGHT HAVE MEMORY [illegible]

I NEVER THOUGHT TO CONSIDER DRAM ACCESSES MIGHT AFFECT ADJACENT, NON-ACCESSED MEMORY CELLS!

WHAT IS A BUG?

- All automated systems check code against a specification
  - By default programs should not: segfault, print uninitialized memory, call system on user-defined buffers, etc.
  - More advanced specifications written case-by-case
- How do you detect bugs you never even considered?
  - Have a security-conscious person think very hard…
  - ???

WHAT IS A BUG?

- Lots of behaviors are case-by-case issues:
  - Not encrypting UID numbers? Probably fine
  - Not encrypting National Identification Numbers? Bad
  - Use of SHA256 to hash data? No problem
  - Use of SHA256(secret || data) to authenticate data? Bad
  - Reading from uninitialized memory? Bad
  - Reading from uninitialized memory in crypto key generation? Terrible, but it might serve a purpose!

WHAT IS A BUG?

- Again, this is an area that requires *creativity*
  - Some bugs may not seem like bugs at first glance
  - “Features” may be chained to make bugs
- Discovering very “unique” bugs automatically is very hard!

INTUITION

- Great bug finders aren't always the "fastest"
  (at least, so it seems to me, though I am not a great bug finder)
- Great bug finders *know where to look*
  - This is *not* a "trick"
  - This is a very developed skill
  - This "knack" is hard to directly teach

INTUITION

- Most automated systems try to maximize code coverage
  - Works pretty well in practice
  - …But basically a “brute force” strategy
- Guiding fuzzing or symbolic execution is possible
  - For now, this is just basic heuristics
  - …but this is a great candidate for “deep learning”

BOO! COMPUTERS WON'T SOLVE ANY PROBLEMS!

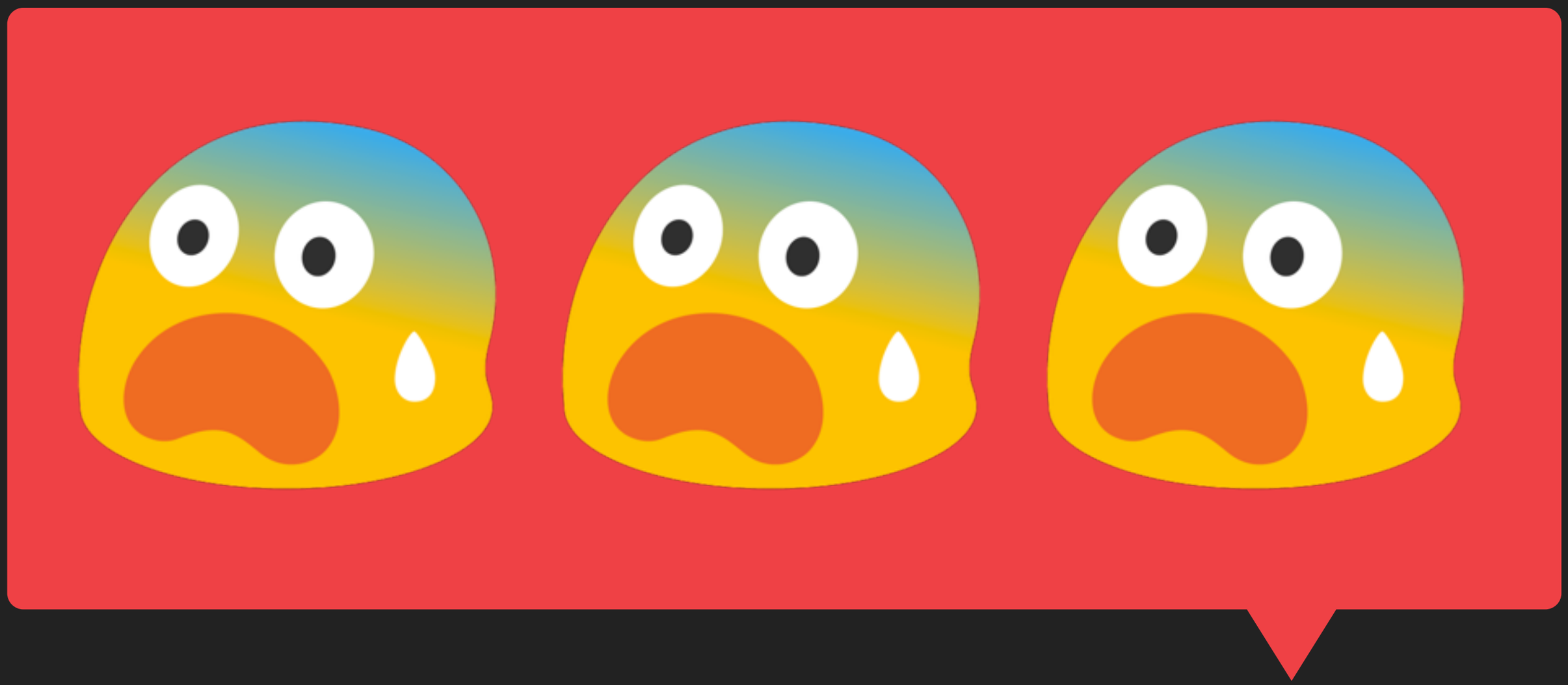
Everyone

CHAPTER 3

THE FUTURE

THE SINGULARITY

- Computer intelligence surpass that of humans
- All our security problems are solved
- ...But maybe just because we are all dead



Everyone

CHAPTER 3.5

THE (NEAR) FUTURE

HISTORICAL PERSPECTIVE: CHESS

- 1945 - Alan Turing theorizes computers could play chess
- 1959 - “Computer chess world champion in 10 years”
- 1970 - Concept of “Computer-Assisted Chess”
- 1978 - “Computer chess world champion in 10 years”
- 1989 - “Computer chess world champion in 3 years”
- 1997 - Deep Blue wins match against Gary Kasparov
- 1998 - First “Computer-Assisted Chess” tournament

RISE OF THE CENTAURS

HISTORICAL PERSPECTIVE: CHESS

- Today, these “centaurs” outperform humans or computers [1]
  - A top computer has an ELO of ~3400
  - The top humans have an ELO of ~2800
  - A human with ELO of < 2000 can still help a computer of ELO 3400!
    - That’s crazy!

[1] https://www.cse.buffalo.edu/~regan/chess/fidelity/FreestyleStudy.html

CENTAURS

- How are “centaurs” different from “people using tools”?
  - Centaurs sounds a lot fancier!
  - Two components that *could* work *individually*
- Humans right now pull most of the weight…
  - Disassembly/decompilation? computers
  - Fuzzing? computers (mostly)
  - ROP payload generation? somewhat mixed
  - Navigating complex code paths? humans
  - Triaging bugs? humans
  - Generating exploits? humans

CENTAURS

- A few possibilities:
  - “Focus on these sketchy areas of code”
  - “Use this input to get through that branch”
  - “Give me an input that sets this value here to X”
  - “Can any other thread of this code modify this?”
  - “Is it possible to call this function twice?”
  - “Make this exploit more reliable”
  - “Get me a large buffer I can write to at a known address”

SOME PREDICTIONS

- "Centaurs" will be a buzzword in the next 0-5 years
  - Maybe "Cyber-Centaurs"
- "Great bugfinders" will do more than fuzzing:
  - Maybe more taint analysis and SMT solving
- "Centaurs" will be more common, starting in CTFs
  - CTF problems generally more "bite-sized"
- Automated systems will get *a lot* better

CONCLUSIONS

- We (humans) need help finding bugs
- We are not “digging our own graves”
  - Some humans will be needed for quite a while
    - (AI might kill us all no matter what)
- Systems today are more magic than you might expect
- There is a lot of (very interesting) research still needed

QUESTIONS...?

Everyone