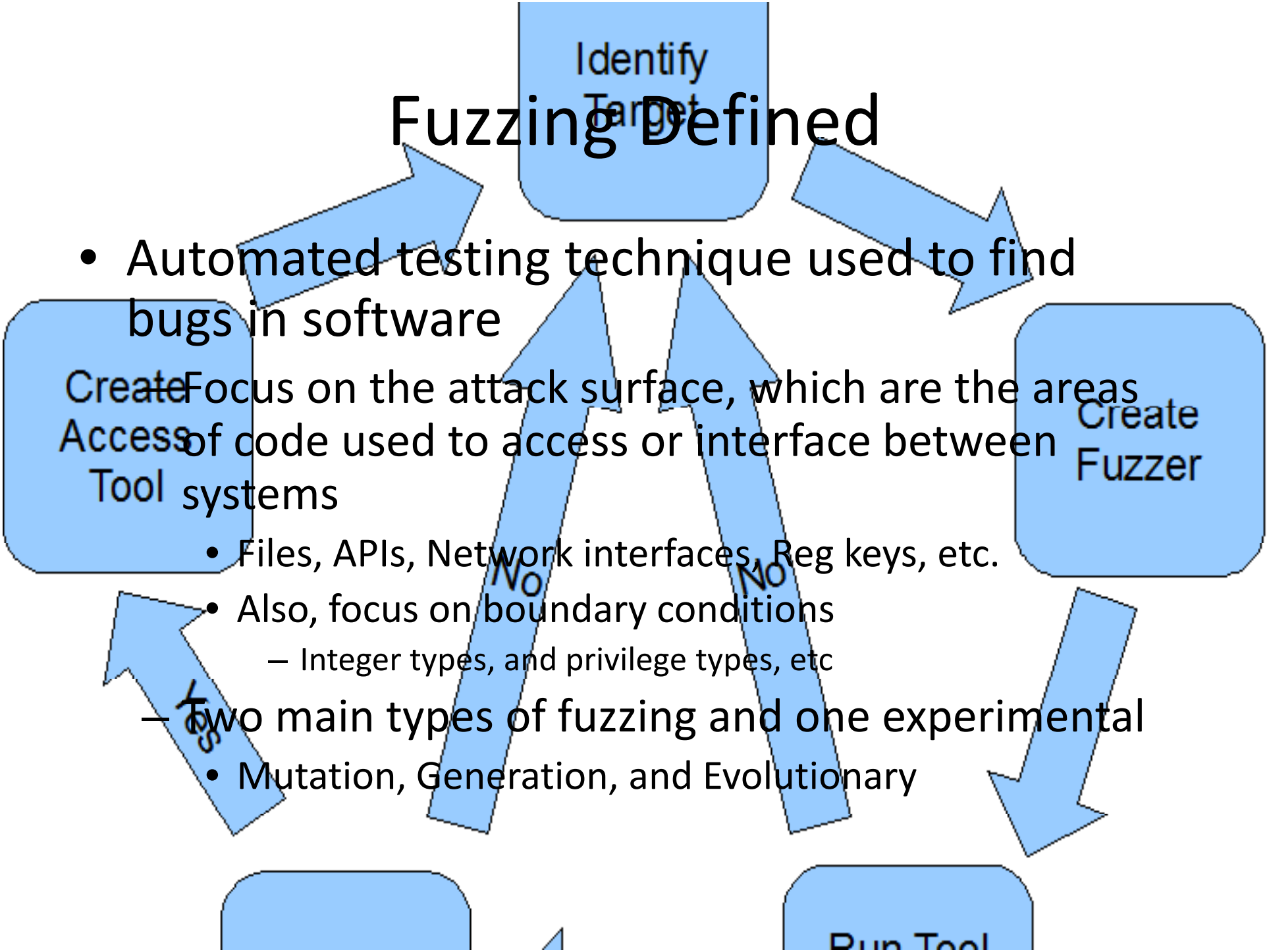
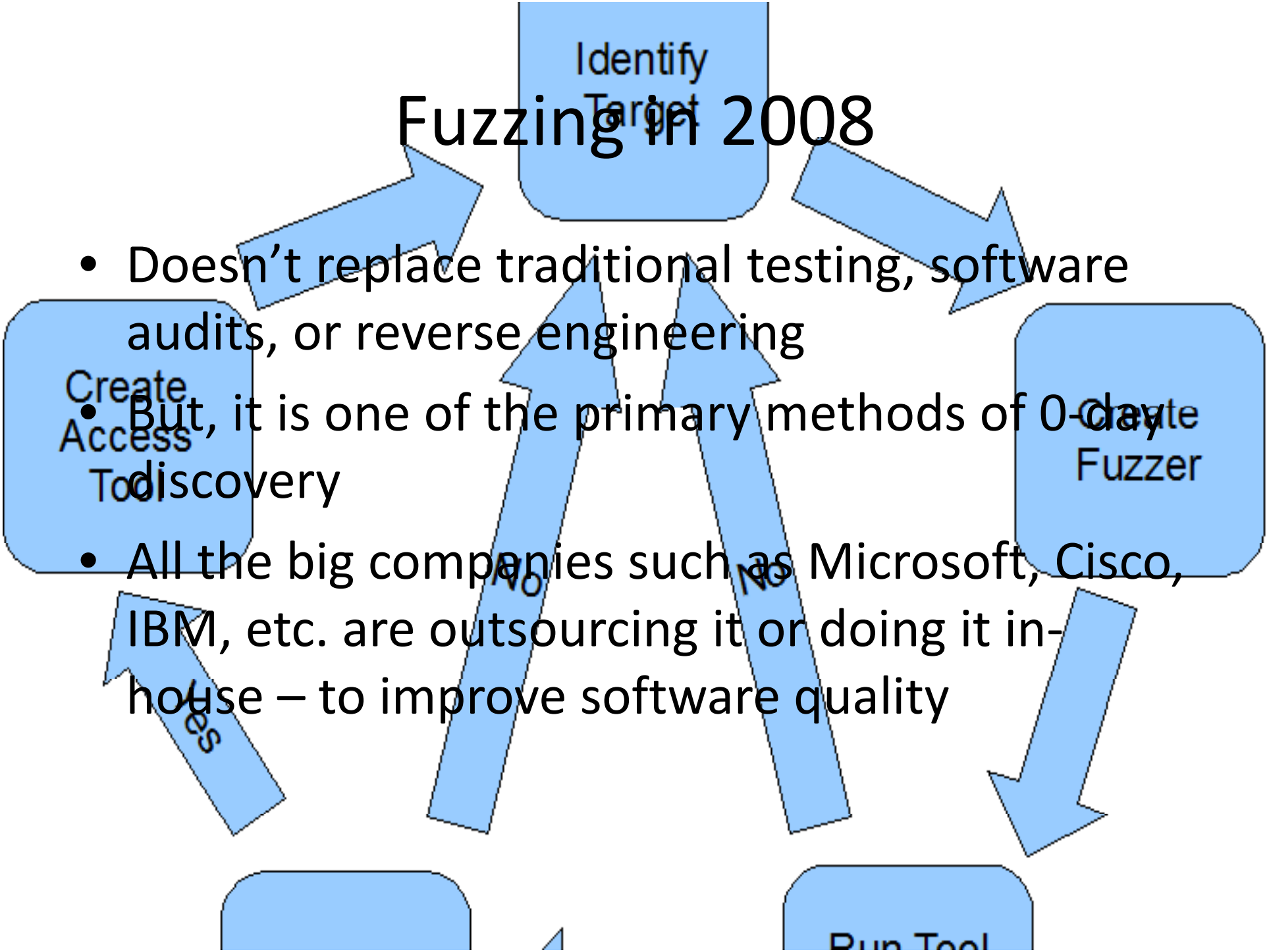
Fuzzing Defined

- Automated testing technique used to find bugs in software
  - Focus on the attack surface, which are the areas of code used to access or interface between systems
    - Files, APIs, Network interfaces, Reg keys, etc.
    - Also, focus on boundary conditions
      - Integer types, and privilege types, etc
  - Two main types of fuzzing and one experimental
    - Mutation, Generation, and Evolutionary

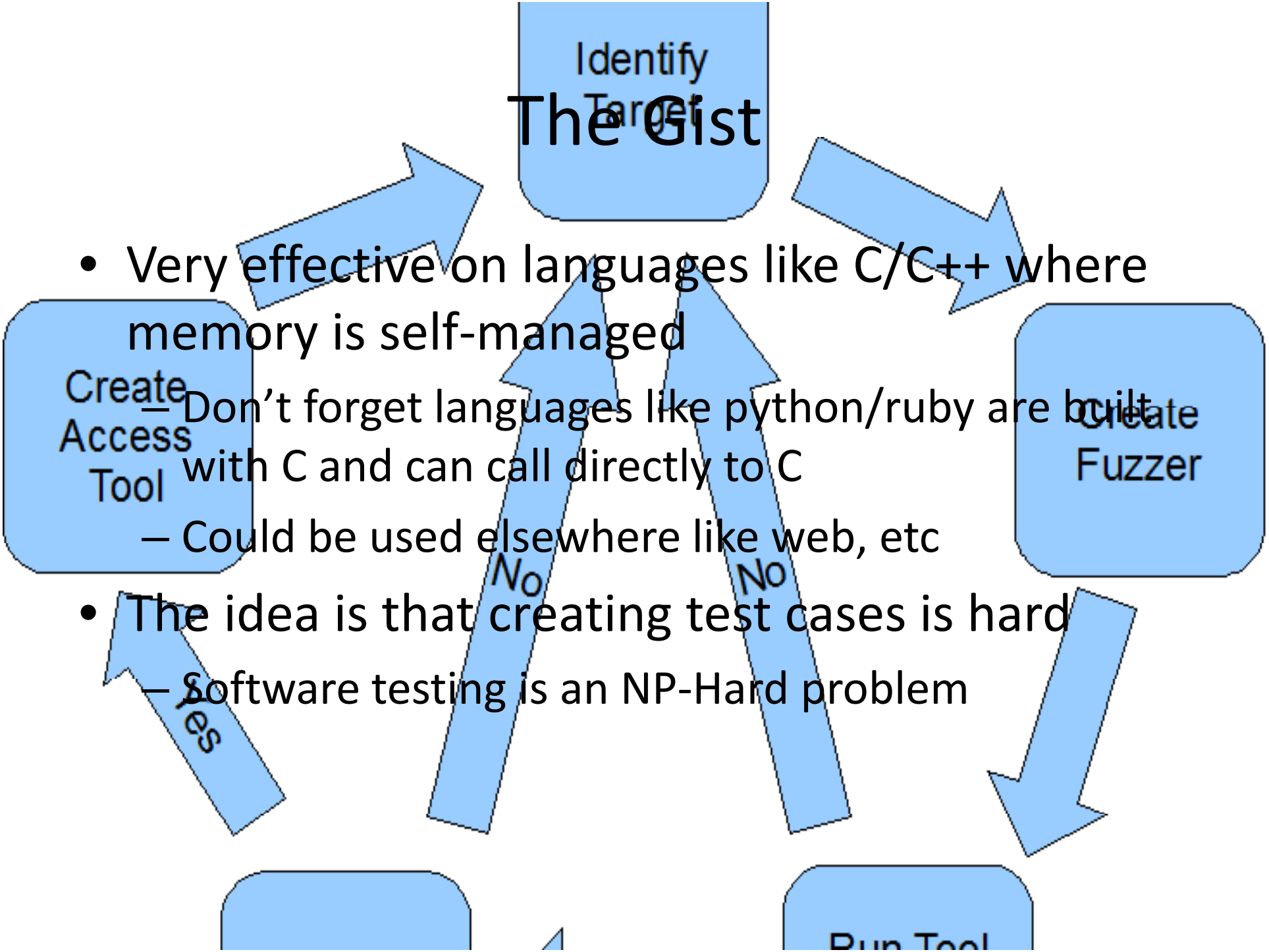


Fuzzing in 2008

- Doesn't replace traditional testing, software audits, or reverse engineering
- But, it is one of the primary methods of 0-day discovery
- All the big companies such as Microsoft, Cisco, IBM, etc. are outsourcing it or doing it in-house – to improve software quality



The Gist



- Very effective on languages like C/C++ where memory is self-managed
  - Don't forget languages like python/ruby are built with C and can call directly to C
  - Could be used elsewhere like web, etc
- The idea is that creating test cases is hard
  - Software testing is an NP-Hard problem

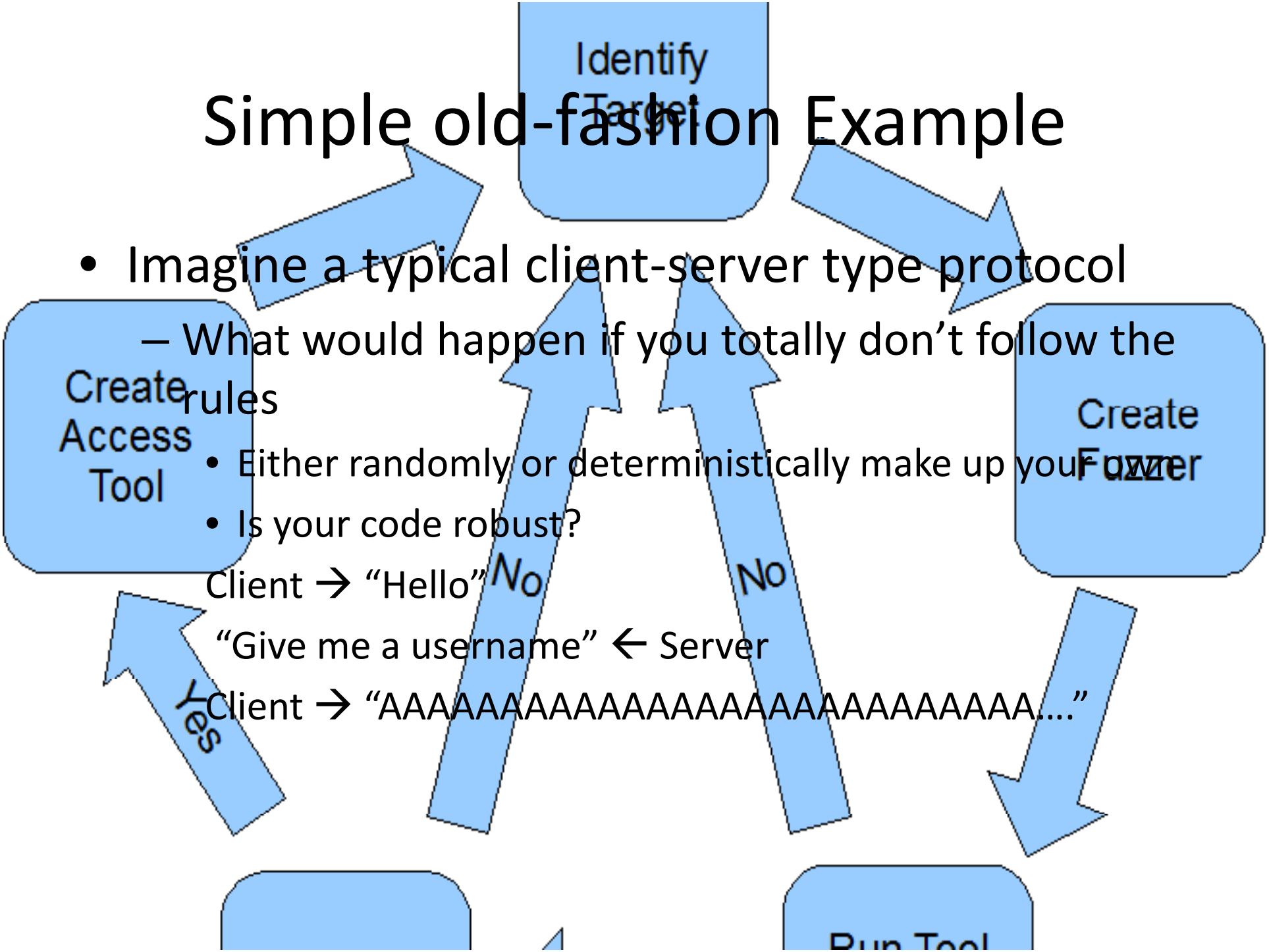
Simple old-fashion Example

- Imagine a typical client-server type protocol
  - What would happen if you totally don't follow the rules
    - Either randomly or deterministically make up your own
    - Is your code robust?

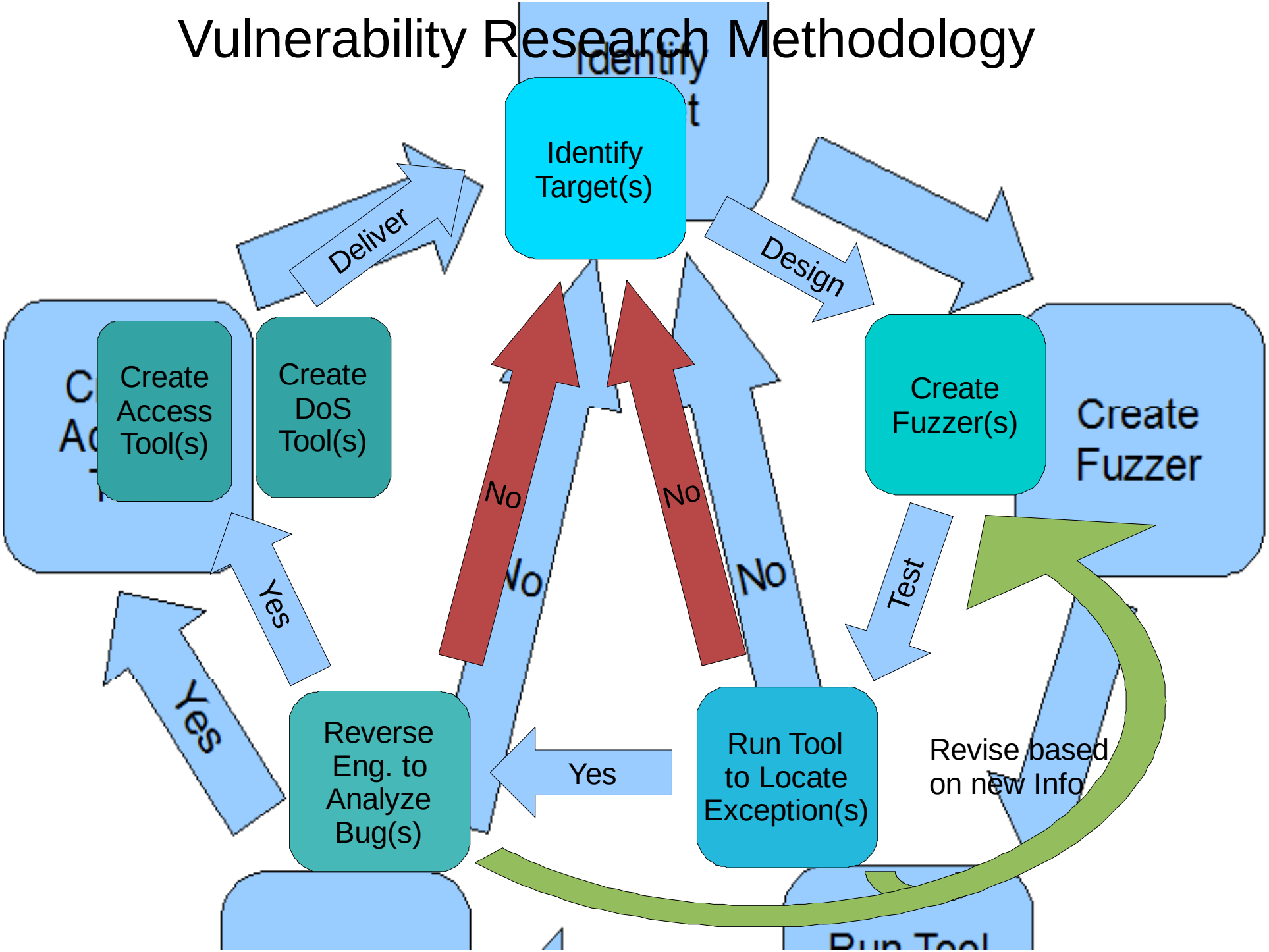
Client → "Hello"

"Give me a username" ← Server

Client → "AAAAAAAAAAAAAAAAAAAAAAAAAAA…."

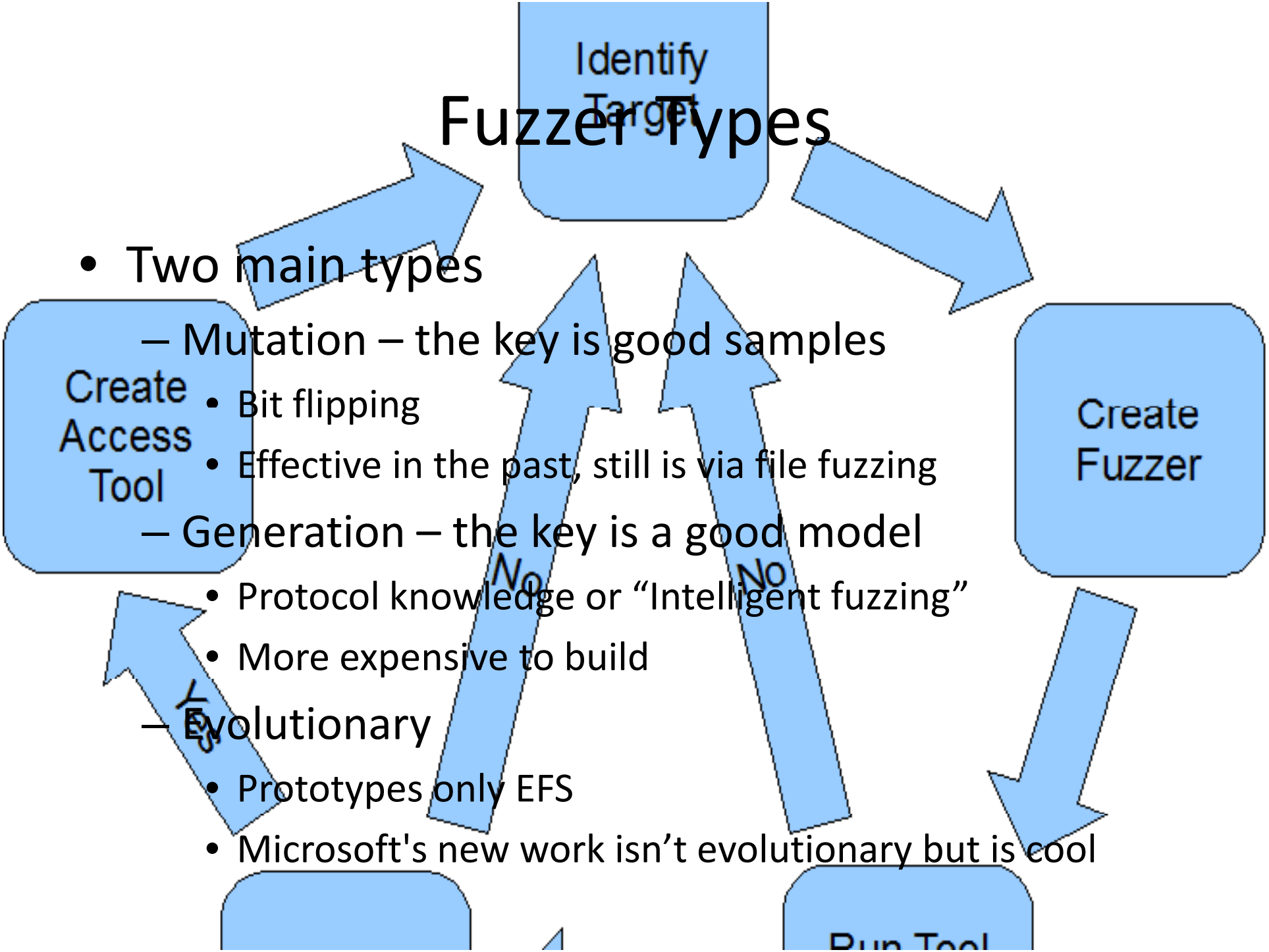


Vulnerability Research Methodology



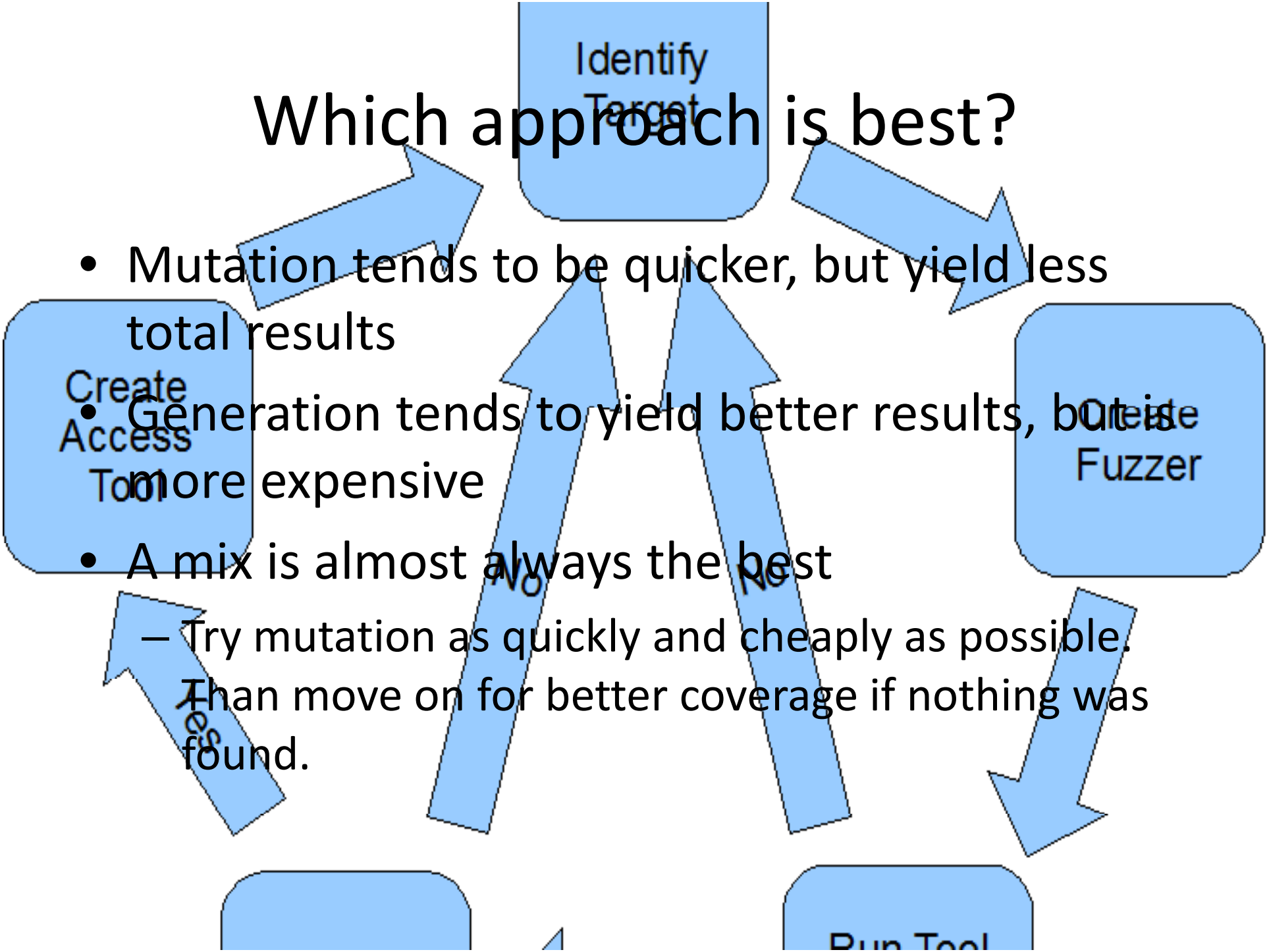
Fuzzer Types

- Two main types
  - Mutation – the key is good samples
    - Bit flipping
    - Effective in the past, still is via file fuzzing
  - Generation – the key is a good model
    - Protocol knowledge or “Intelligent fuzzing”
    - More expensive to build
  - Evolutionary
    - Prototypes only EFS
    - Microsoft's new work isn’t evolutionary but is cool



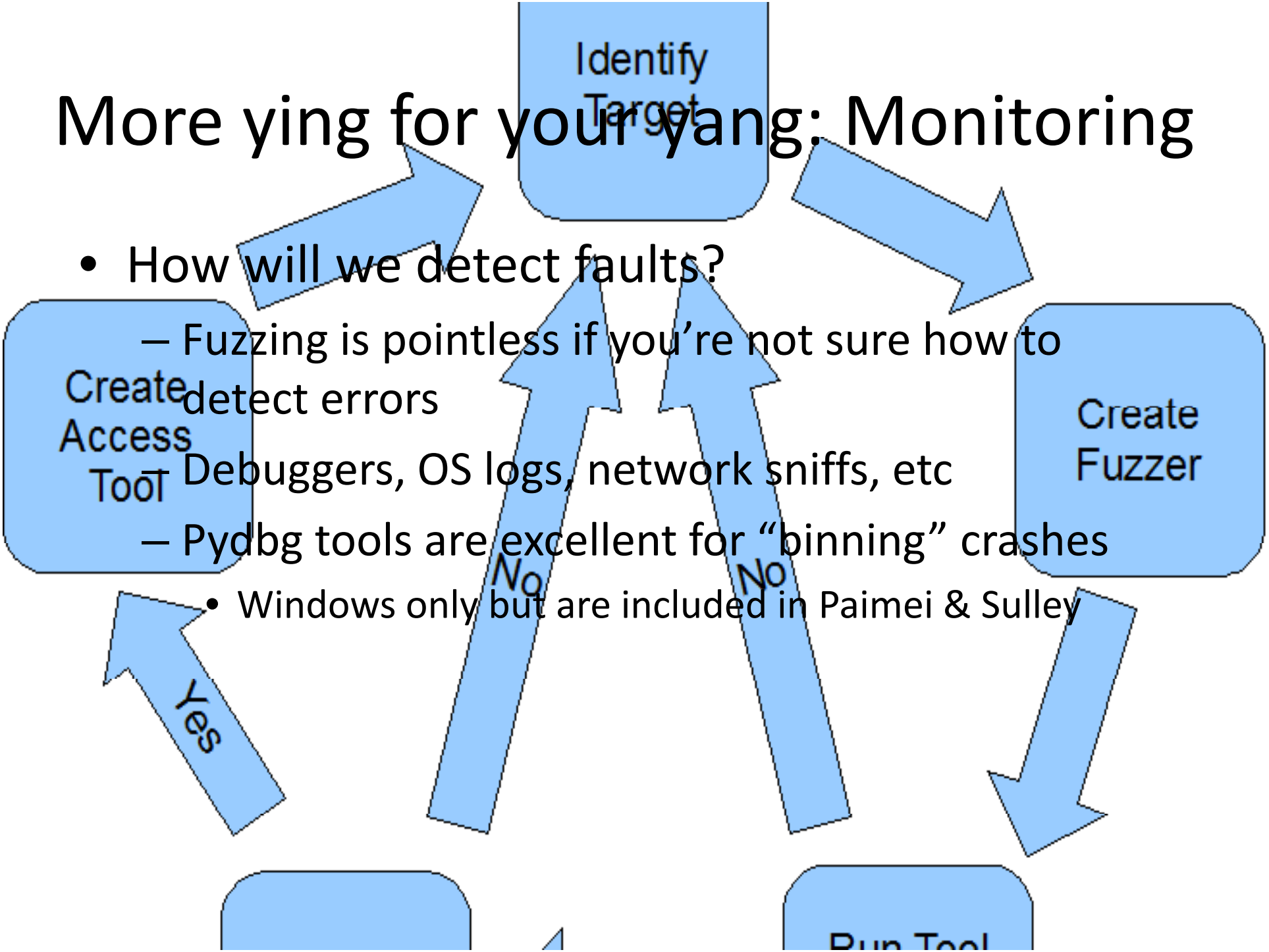
Which approach is best?

- Mutation tends to be quicker, but yield less total results
- Generation tends to yield better results, but is more expensive
- A mix is almost always the best
  - Try mutation as quickly and cheaply as possible. Than move on for better coverage if nothing was found.



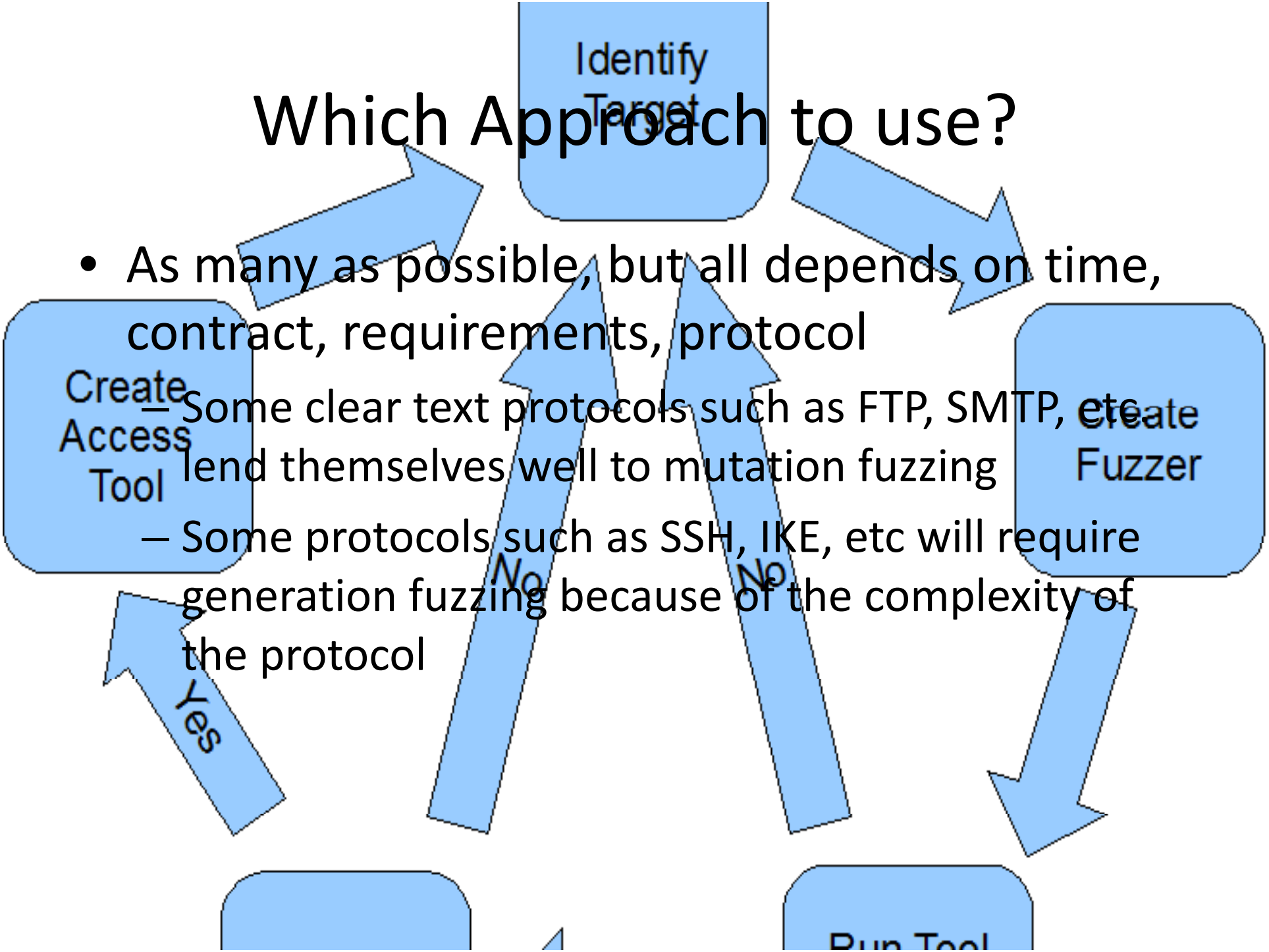
More ying for your yang: Monitoring

- How will we detect faults?
  - Fuzzing is pointless if you're not sure how to detect errors
  - Debuggers, OS logs, network sniffs, etc
  - Pydbg tools are excellent for "binning" crashes
    - Windows only but are included in Paimei & Sulley



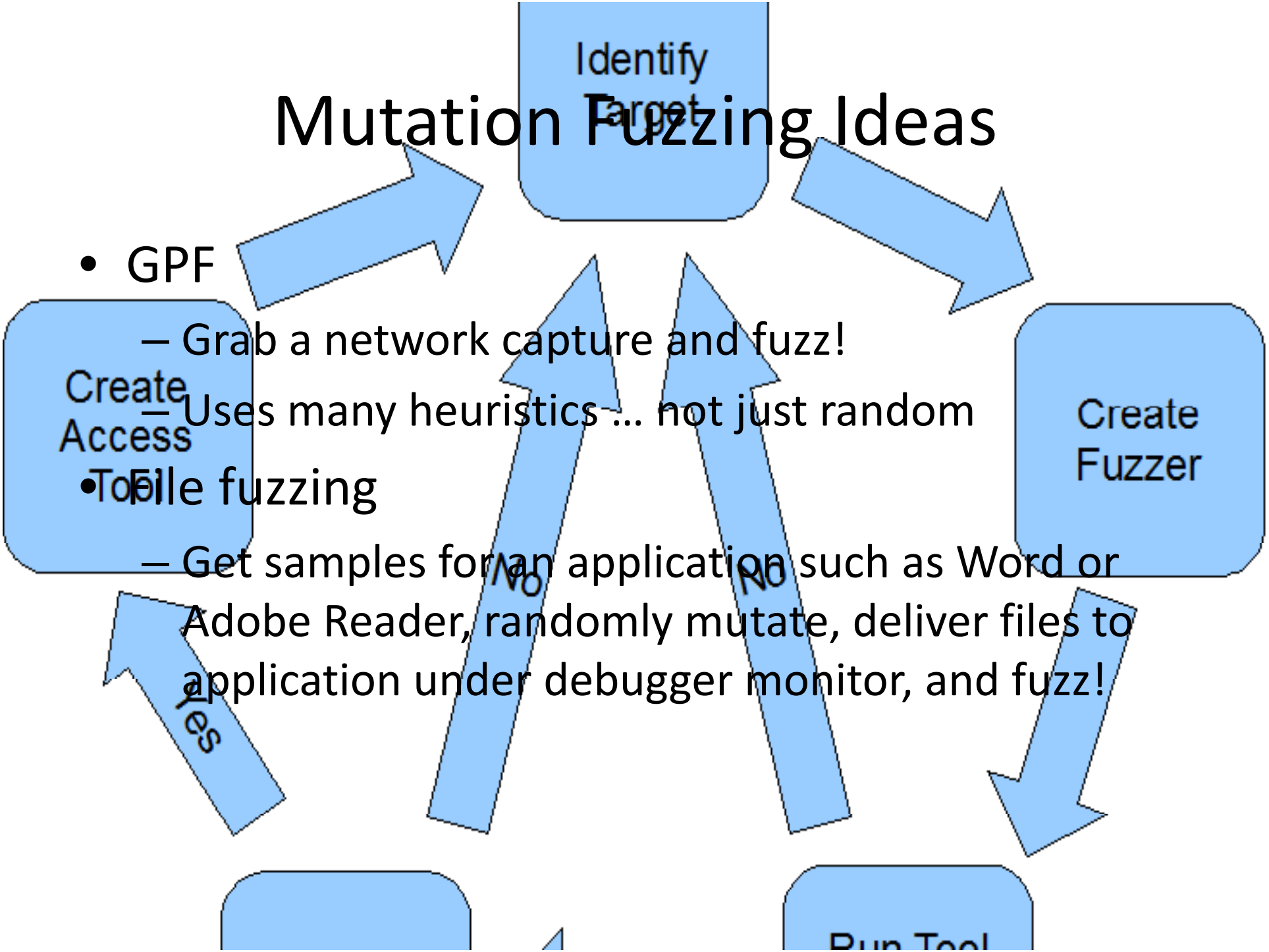
Which Approach to use?

- As many as possible, but all depends on time, contract, requirements, protocol
  - Some clear text protocols such as FTP, SMTP, etc lend themselves well to mutation fuzzing
  - Some protocols such as SSH, IKE, etc will require generation fuzzing because of the complexity of the protocol



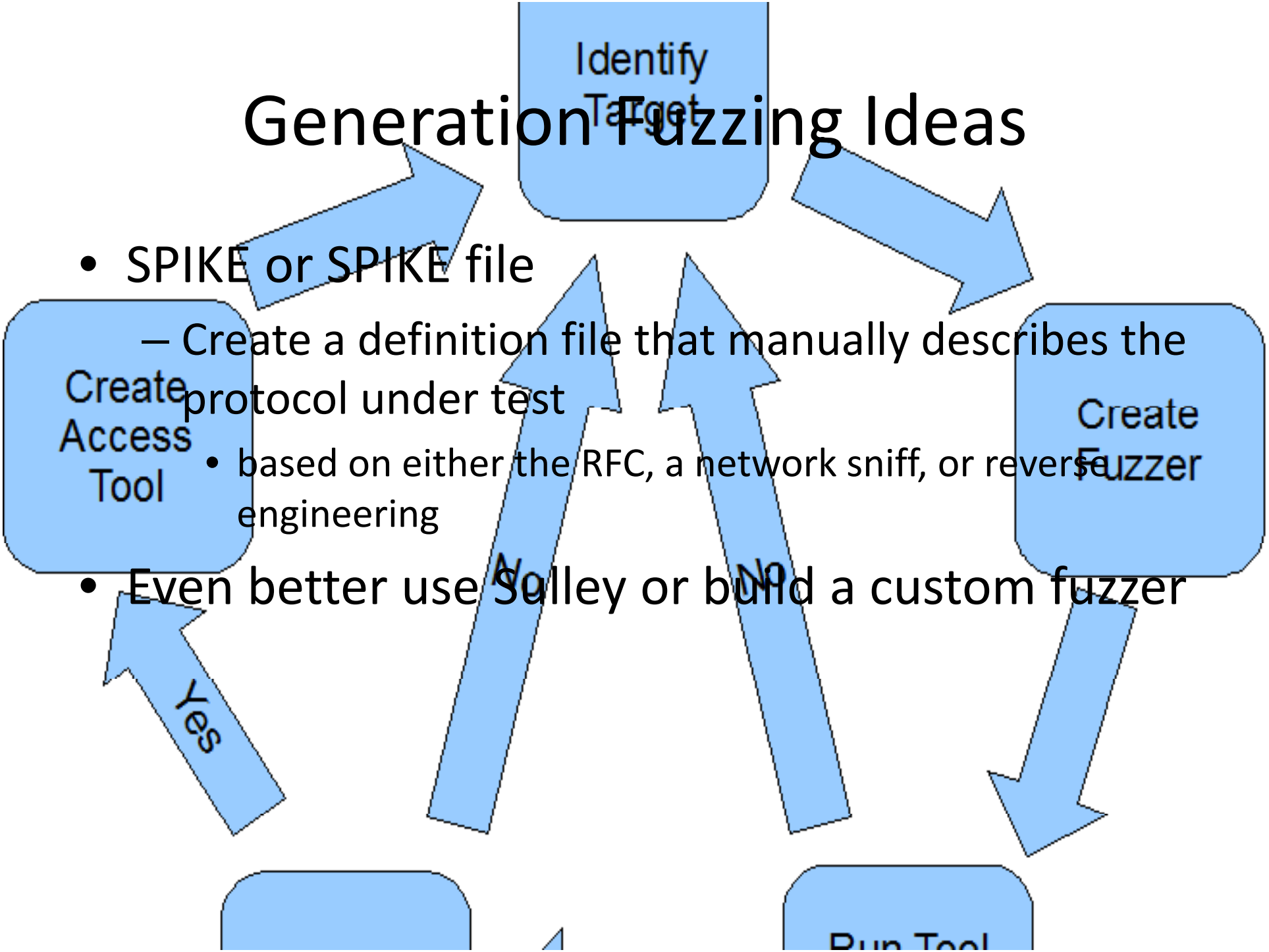
Mutation Fuzzing Ideas

- GPF
  - Grab a network capture and fuzz!
  - Uses many heuristics … not just random
- File fuzzing
  - Get samples for an application such as Word or Adobe Reader, randomly mutate, deliver files to application under debugger monitor, and fuzz!



Generation Fuzzing Ideas

- SPIKE or SPIKE file
  - Create a definition file that manually describes the protocol under test
    - based on either the RFC, a network sniff, or reverse engineering
- Even better use Sulley or build a custom fuzzer



Summary

- For MUCH more information please read our book!