On Weird Machines and Lazy Functional Programming

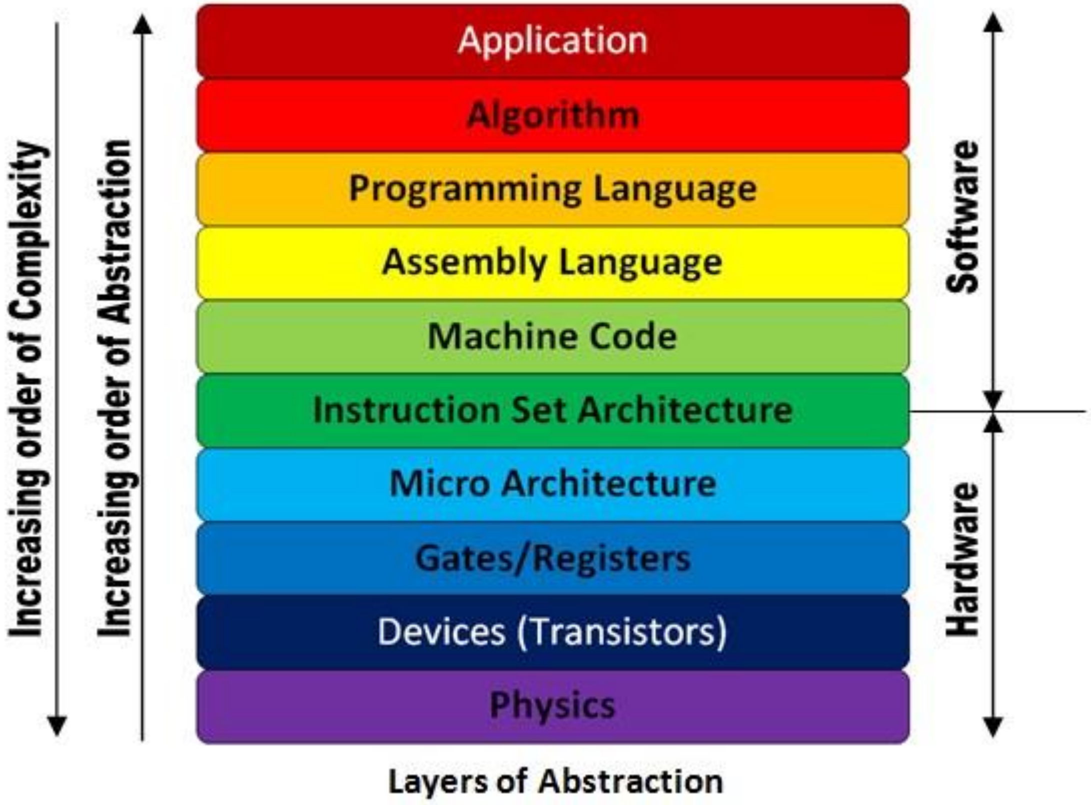
Cecil Accetti - Prof. Peilin Liu

Doctoral College Conference – University of Surrey

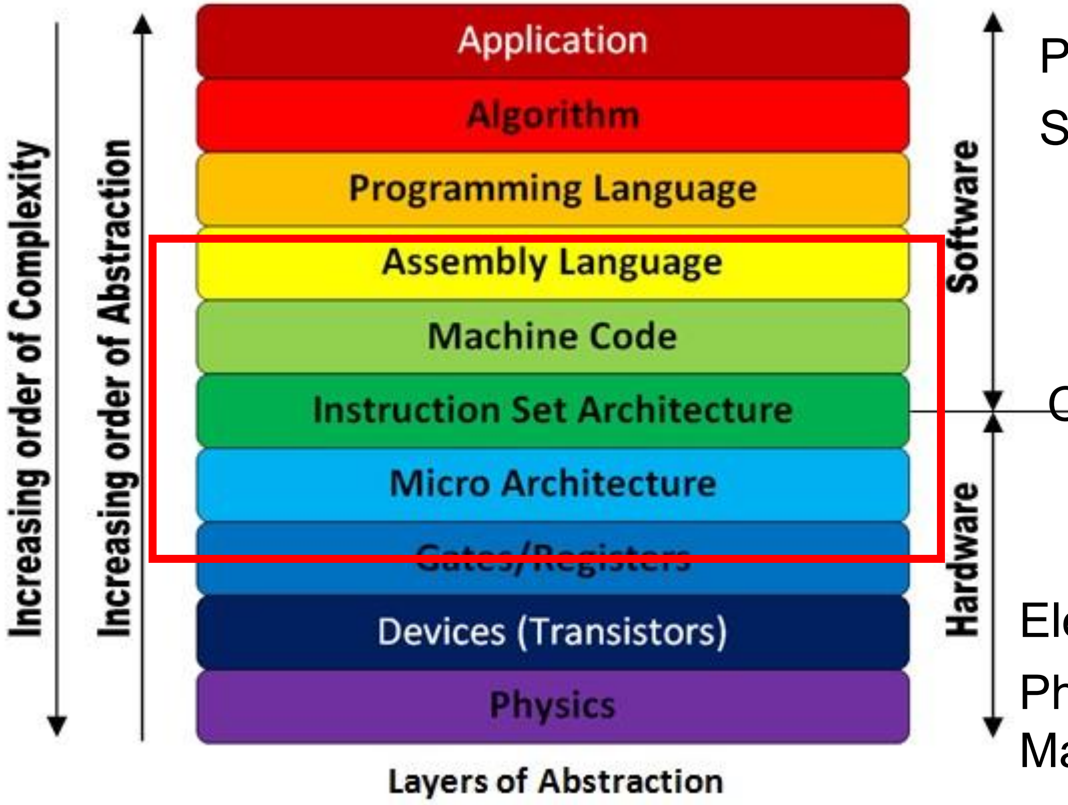
July 10, 2019

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

What is computer architecture?



What is computer architecture?



Users

Programmers

Software engineers

Computer Architects

Electronic Engineers

Physicists, Chemists

Materials Scientists

Life of a Computer Architecture PhD student (before 2018)

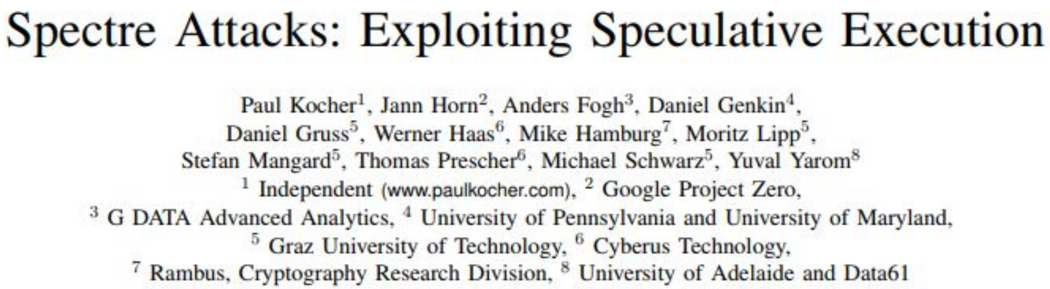
- Intel, AMD, ARM keep shipping a faster, newer, better, multicore processor every 6 months
- It seems they’ve got it all covered…
  - Superscalar, speculative, hyperthreading, out-of-order, SIMD, SIMT, hypervisors, virtualization, neural processing units
- What could *I* do?
  - No budget
  - No contracts, NDAs

2018: Meltdown and Spectre

- Most CPUs are flawed!
  - (Yay!)

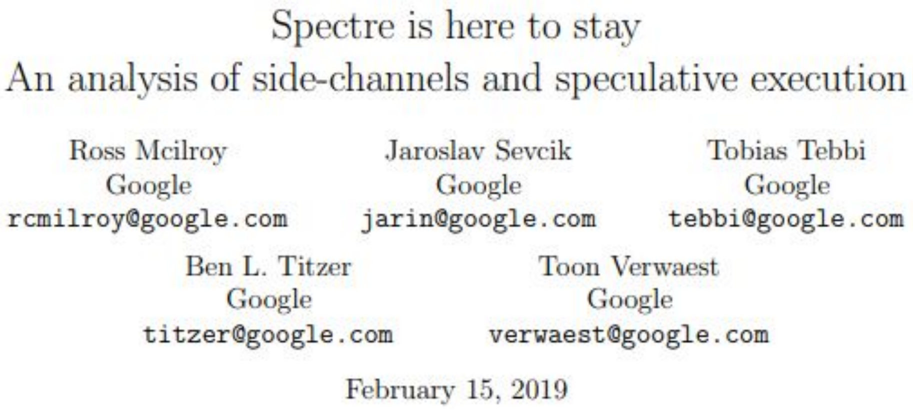
- Vendors can’t* fix it!
  - (Yay!)^2

- It’s a conceptual problem, **not a mistake**!
  - (Yay!)^3



Spectre Attacks: Exploiting Speculative Execution

Paul Kocher[1], Jann Horn[2], Anders Fogh[3], Daniel Genkin[4],
Daniel Gruss[5], Werner Haas[6], Mike Hamburg[7], Moritz Lipp[5],
Stefan Mangard[5], Thomas Prescher[6], Michael Schwarz[5], Yuval Yarom[8]
[1] Independent (www.paulkocher.com), [2] Google Project Zero,
[3] G DATA Advanced Analytics, [4] University of Pennsylvania and University of Maryland,
[5] Graz University of Technology, [6] Cyberus Technology,
[7] Rambus, Cryptography Research Division, [8] University of Adelaide and Data61



Spectre is here to stay
An analysis of side-channels and speculative execution

Ross Mcilroy
Google
rcmilroy@google.com

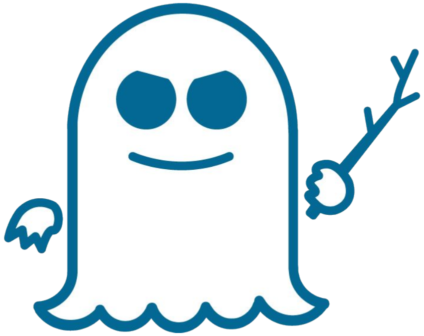
Jaroslav Sevcik
Google
jarin@google.com

Tobias Tebbi
Google
tebbi@google.com

Ben L. Titzer
Google
titzer@google.com

Toon Verwaest
Google
verwaest@google.com

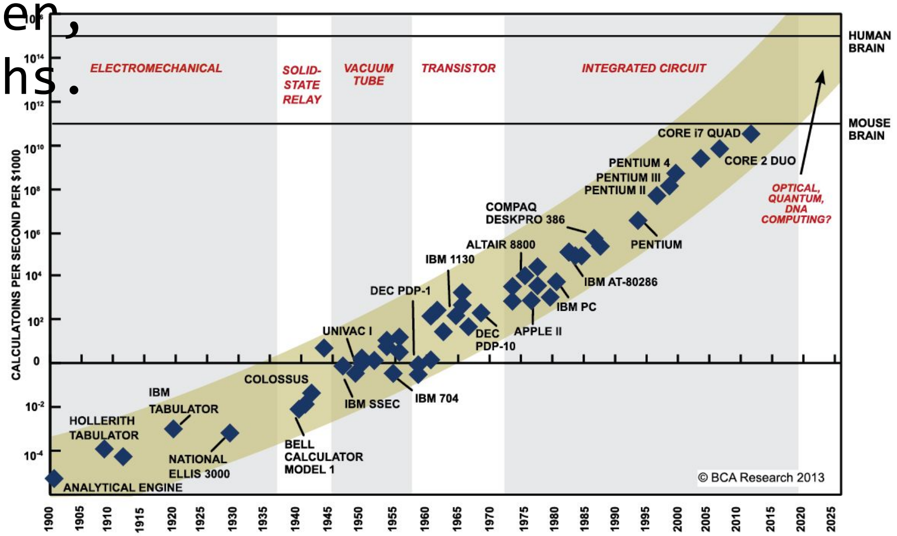
February 15, 2019



* Don’t want or, don’t know how

Spectre – Fast or secure, not both

- Moore’s law(1965) gave us faster, smaller machines every 18 months.
- More transistors per area
- Complex microarchitectures
  - Multilevel caches
  - Superscalar, out-of-order execution
  - Branch Prediction
  - Speculative execution



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

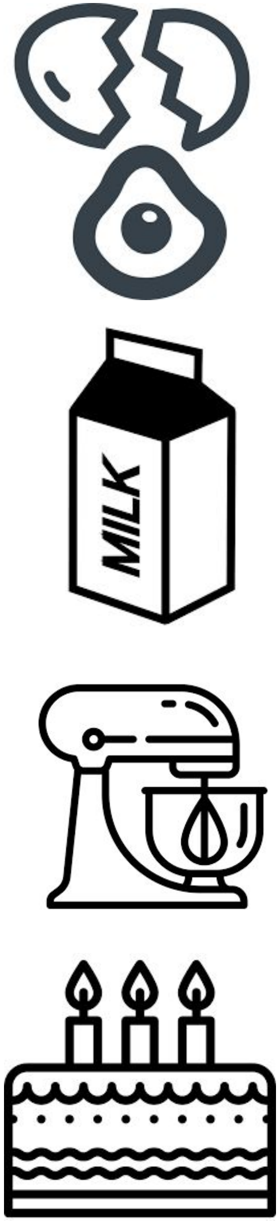
Spectre – Fast or secure, not both

- Speculation: gambling with your program

- Speculation allows transient access to secret information

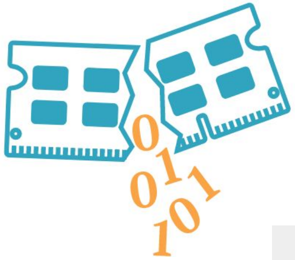
- Secret information becomes a hidden machine state

- Disable speculation?
  ...and we’re back to ~1998 performance levels!



Other microarchitecture exploits

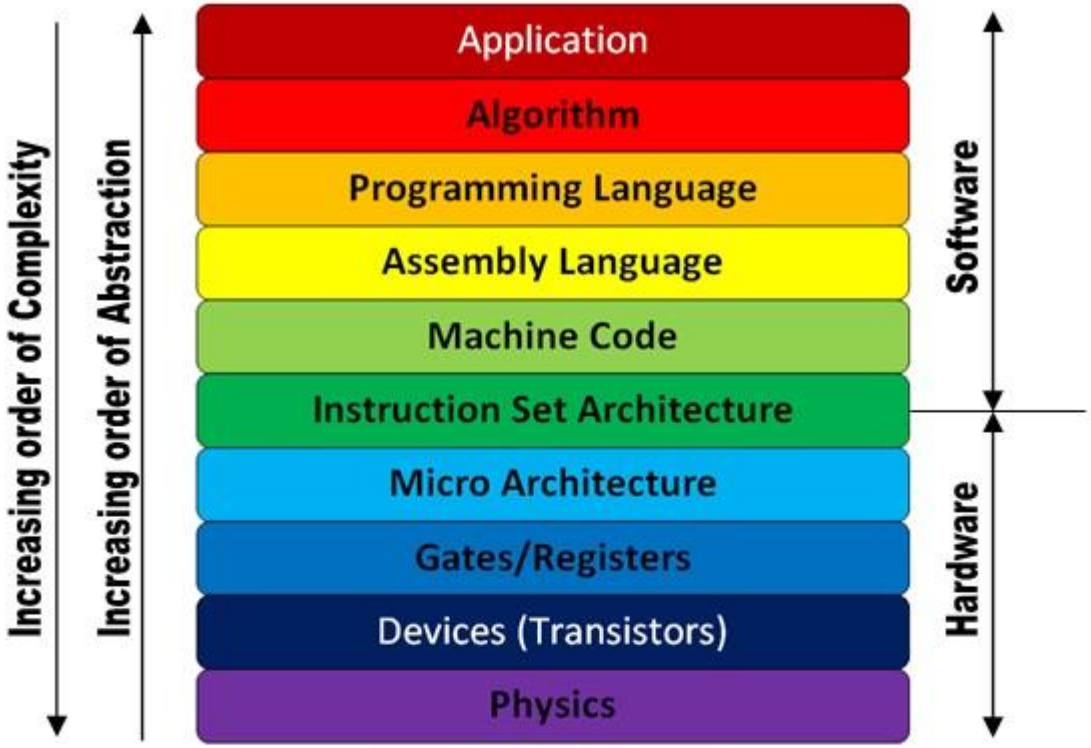
- Rowhammer (2014)
  - Indirect modification of memory contents.
    - Flips bits (0 to 1, 1 to 0)



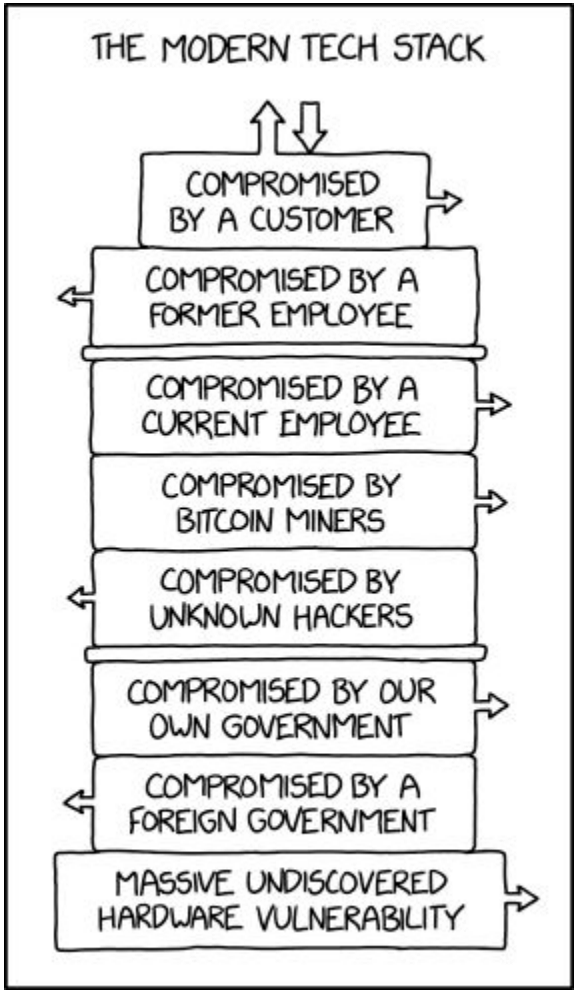
- Rambleed (2019)
  - Leaks data, similar to rowhammer
- Foreshadow (2018)
  - Bypass to Intel security instructions (SGX)

- Spoiler (2019)
  - Speculation execution + Rowhammer
- And the list goes on…

This is not new



Layers of Abstraction


THE MODERN TECH STACK
COMPROMISED BY A CUSTOMER
COMPROMISED BY A FORMER EMPLOYEE
COMPROMISED BY A CURRENT EMPLOYEE
COMPROMISED BY BITCOIN MINERS
COMPROMISED BY UNKNOWN HACKERS
COMPROMISED BY OUR OWN GOVERNMENT
COMPROMISED BY A FOREIGN GOVERNMENT
MASSIVE UNDISCOVERED HARDWARE VULNERABILITY

https://xkcd.com/2166/

This is not new

- A “cambrian explosion” in the 80s and 90s
  - Worms
  - Viruses
  - Trojans
  - Rootkits
  - -> malware

- Coordinated attacks, by corporations and nation states

- Exploit market
  - Brokers
  - Buyers, sellers
  - No questions asked

  - Attacks on:
    - Power grids, automotive systems, governmental communications...

ZERODIUM Payouts for Mobiles*

RJB: Remote Jailbreak with Persistence
RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass

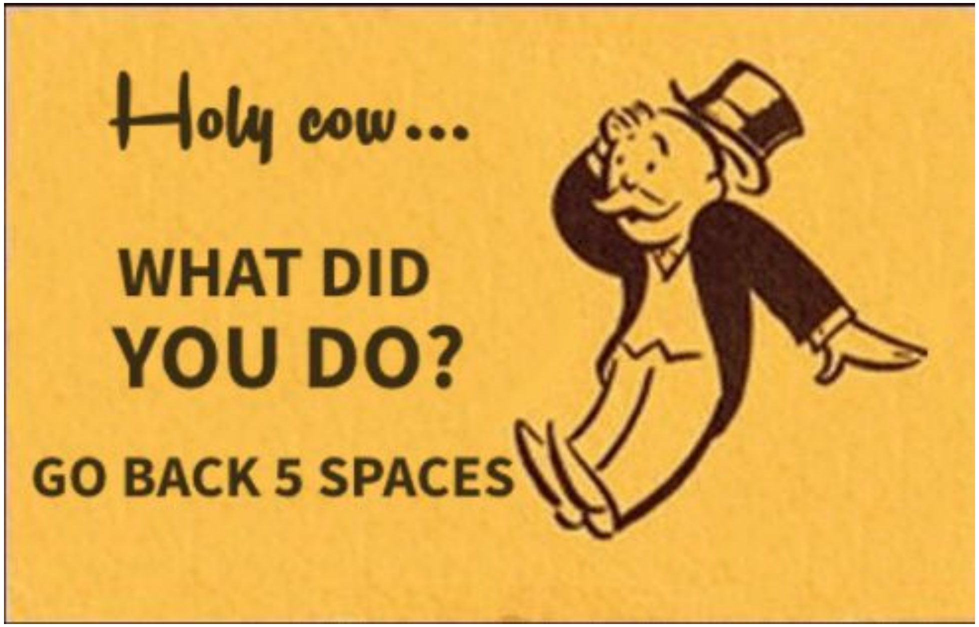
🟥 iOS
🟫 Android
🟦 Any OS

Up to $2,000,000									🟥 1.001 iPhone RJB Zero Click IOS
Up to $1,500,000									🟥 1.002 iPhone RJB IOS
Up to $1,000,000							🟦 2.001 WhatsApp RCE+LPE IOS/Android	🟦 2.002 SMS/MMS RCE+LPE IOS/Android	🟥 2.003 iMessage RCE+LPE IOS
Up to $500,000	🟦 2.004 WeChat RCE+LPE IOS/Android			🟦 2.005 FB Messenger RCE+LPE IOS/Android	🟦 2.006 Signal RCE+LPE IOS/Android	🟦 2.007 Telegram RCE+LPE IOS/Android	🟦 2.008 Email App RCE+LPE IOS/Android	🟫 3.001 Chrome RCE+LPE Android	🟥 3.002 Safari RCE+LPE IOS
Up to $200,000	🟦 4.001 Baseband RCE+LPE IOS/Android		🟦 5.001 LPE to Kernel/Root IOS/Android	🟦 2.009 Media Files RCE+LPE IOS/Android	🟦 2.010 Documents RCE+LPE IOS/Android	🟫 3.003 SBX for Chrome Android	🟫 3.004 Chrome RCE w/o SBX Android	🟥 3.005 SBX for Safari IOS	🟥 3.006 Safari RCE w/o SBX IOS
Up to $100,000	🟦 6.001 Code Signing Bypass IOS/Android	🟦 4.002 WiFi RCE IOS/Android	🟦 4.003 RCE via MitM IOS/Android	🟫 5.002 LPE to System Android	🟦 7.001 Information Disclosure IOS/Android	🟦 7.002 [k]ASLR Bypass IOS/Android	🟫 8.001 PIN Bypass Android	🟥 8.002 Passcode Bypass IOS	🟥 8.003 Touch ID Bypass IOS

*All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/01 © zerodium.com

What should we do?



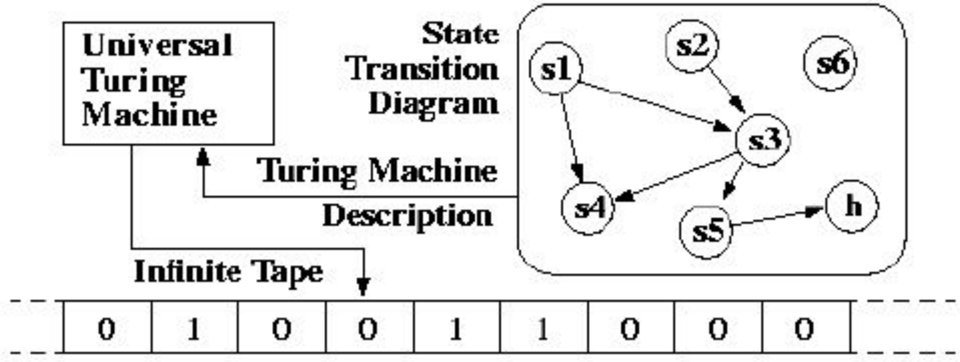
Security x Computing Theory

- Is there a fundamental cause of vulnerabilities?
- How to derive security threats from computing theory concepts?
- How to design a more secure computer?
- Open problem: A handful of publications in this decade tried to formalize attacks and exploiting methods. (langsec.org, T.Dullien)

Foundations

- *Computational Models*
  - Finite State Machines (time-dependent, stateful)
  - Turing Machines (time-dependent, stateful)
  - Combinatory Logic ≡ λ-Calculus (time-independent, stateless)

- Von Neumann Architecture :
  - Stored program -> Universal Turing Machine

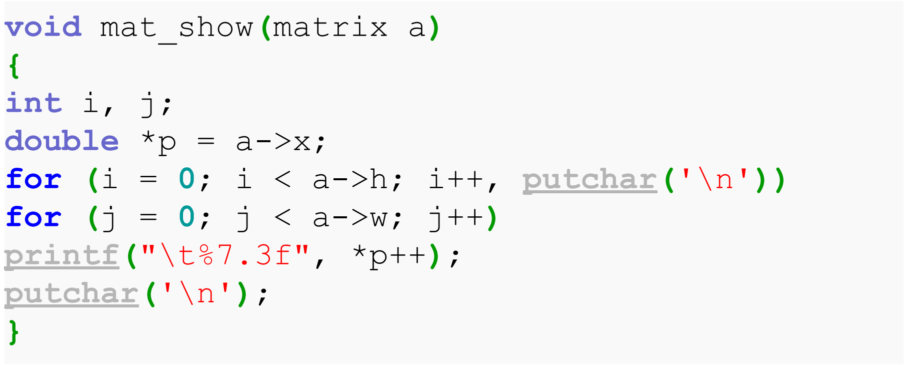


Imperative programming

- *C/C++, Pascal, Java, Python, x86, ARM, MIPS, RISC-V...*
- *Programmer describes an algorithm : a Turing machine that solves a problem, to be executed by the physical machine*
- *Program controls the behavior (state) of the machine:*
    - *Read (write) data from (to) memory*
    - *Set the order of operations*
        - *Variable assignments, loops, procedures*

Imperative programming

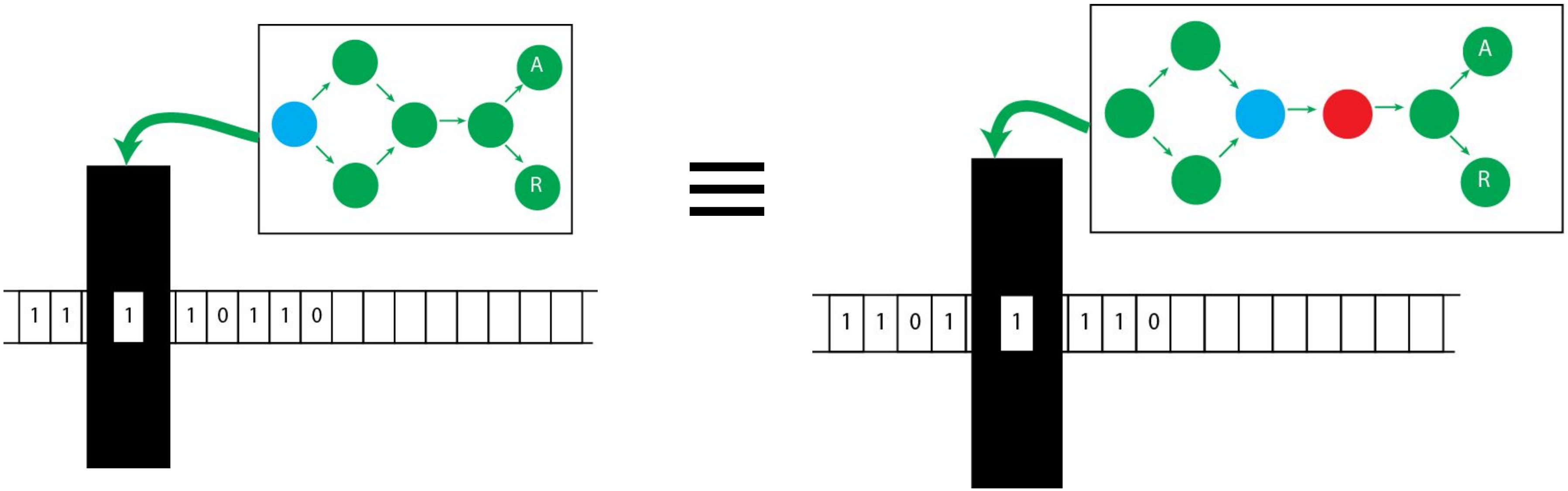
- *C/C++, Pascal, Java, Python, x86, ARM, MIPS, RISC-V...*
- *Programmer describes an algorithm : a Turing machine that solves a problem, to be executed by the physical machine*
- *Program controls the behavior (state) of the machine:*
  - *Read (write) data from (to) memory*
  - *Set the order of operations*
    - *Variable assignments, loops, procedures*



```
void mat_show(matrix a)
{
int i, j;
double *p = a->x;
for (i = 0; i < a->h; i++, putchar('\n'))
for (j = 0; j < a->w; j++)
printf("\t%7.3f", *p++);
putchar('\n');
}
```

Read
Write
Control operations:
    branches
    jumps

The Security Problem of Turing Machines

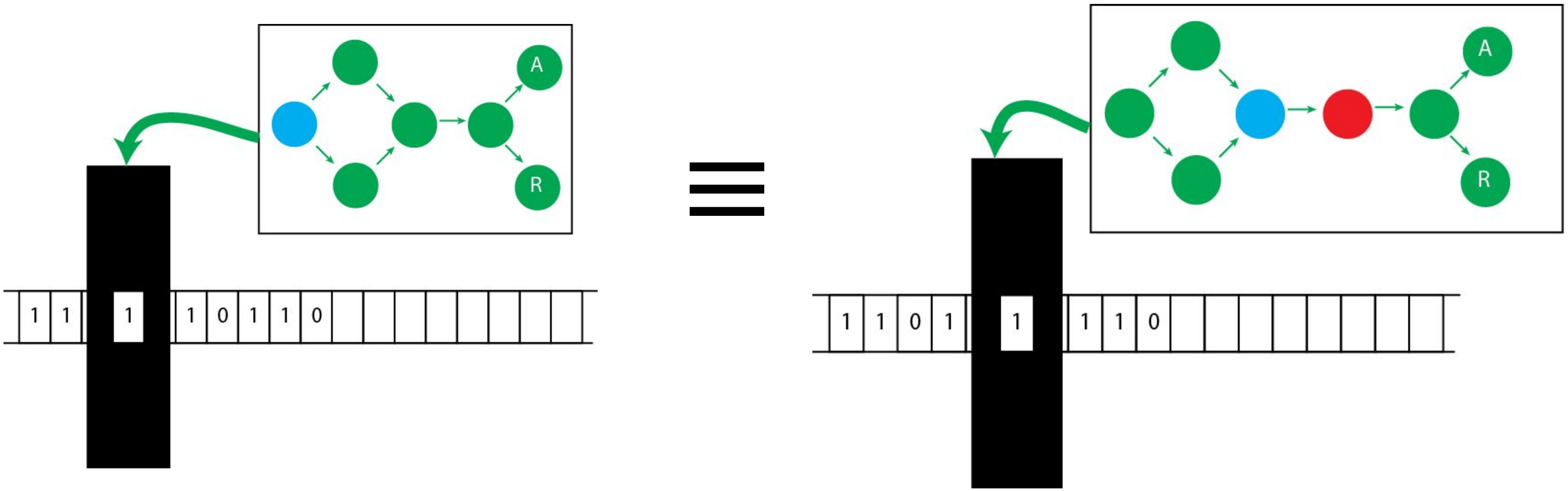


Turing Machines with different set of states can be equivalent!

The theory gives no mechanism to verify the behavior of a TM, other than observing its output.(Halting problem –Entscheidungsproblem)

→ You have an infinite set of programs with the same output

The Security Problem of Turing Machines



Turing Machines with different set of states can be equivalent!

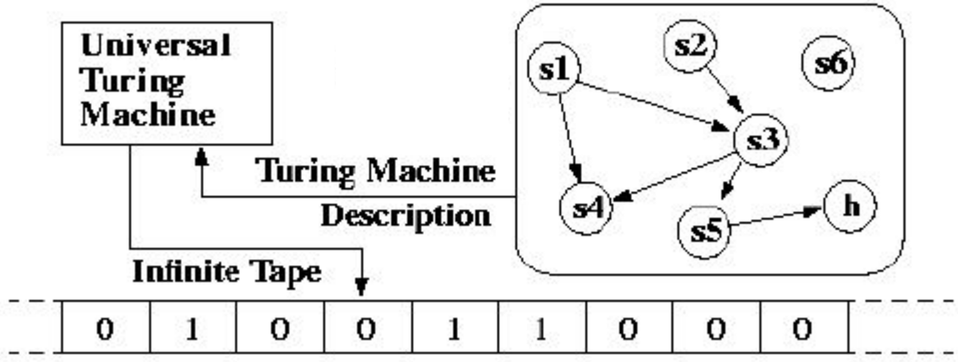
The theory gives no mechanism to verify the behavior of a TM, other than observing its output.(Halting problem -Entscheidungsproblem)

→ You have an infinite set of programs with the same output 😈

Foundations

- *Computational Models*
  - Finite State Machines (time-dependent, stateful)
  - Turing Machines (time-dependent, stateful)
  - Combinatory Logic ≡ λ-Calculus (time-independent, stateless)

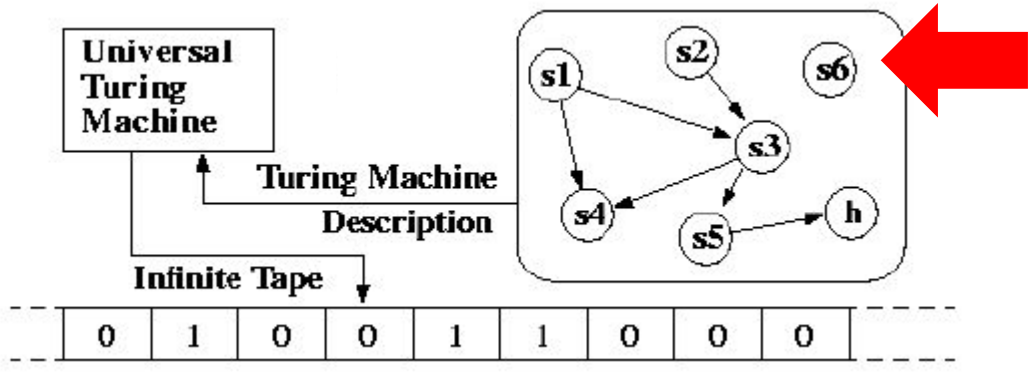
- Von Neumann Architecture :
  - Stored program -> Universal Turing Machine



Foundations

- *Computational Models*
  - Finite State Machines (time-dependent, stateful)
  - Turing Machines (time-dependent, stateful)
  - Combinatory Logic ≡ λ-Calculus (time-independent, stateless)

- Von Neumann Architecture :
  - Stored program -> Universal Turing Machine

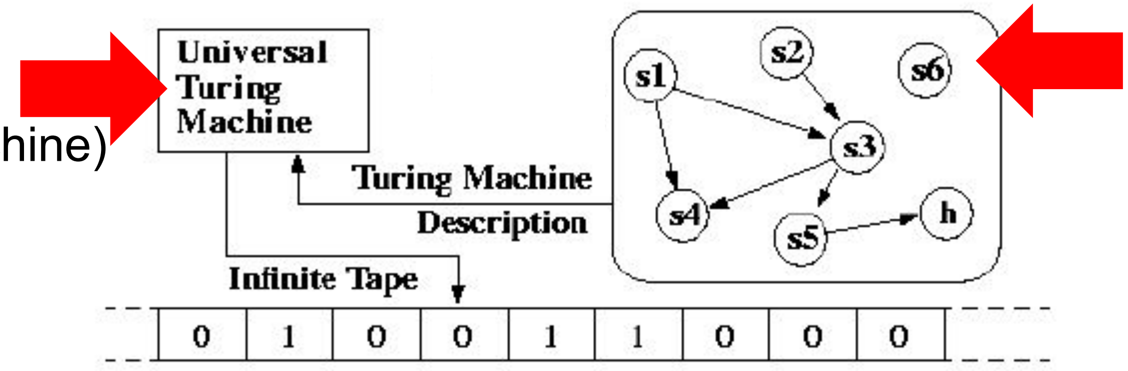


Source code
(Turing machine)

Foundations

- *Computational Models*
  - Finite State Machines (time-dependent, stateful)
  - Turing Machines (time-dependent, stateful)
  - Combinatory Logic ≡ λ-Calculus (time-independent, stateless)

- Von Neumann Architecture :
  - Stored program -> Universal Turing Machine

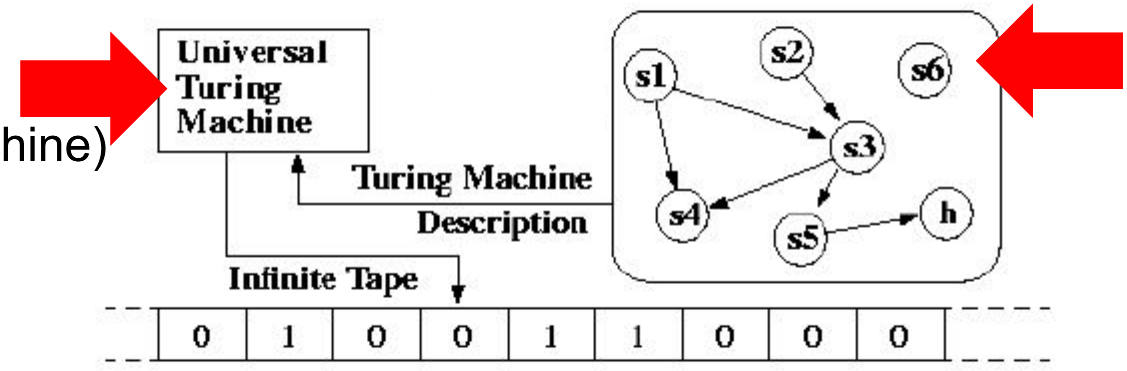


Processor (Turing machine)

Source code (Turing machine)

Foundations

- *Computational Models*
  - Finite State Machines (time-dependent, stateful)
  - Turing Machines (time-dependent, stateful)
  - Combinatory Logic ≡ λ-Calculus (time-independent, stateless)
- Von Neumann Architecture :
  - Stored program -> Universal Turing Machine



Processor
(Turing machine)

Source code
(Turing machine)

Hidden states compose: (source TM) • (processor TM) 😈

Security x Computing Theory

- Security fundamentals:
  - Confidentiality
    - Keep secrets secret
  - Availability
    - Keep system running
  - Integrity
    - Keep data unchanged
- Too abstract
  - Exploit practitioners have their own community
    - Language, jargon, methods
    - Forums, message boards, blogs
    - Academy-averse

Weird Machines

Weird machines

- A set of states outside the original program, but enabled by it.

Weird machines

- ~~A set of states outside the original program, but enabled by it.~~
- What you call a computer when it is not doing what you told it to do

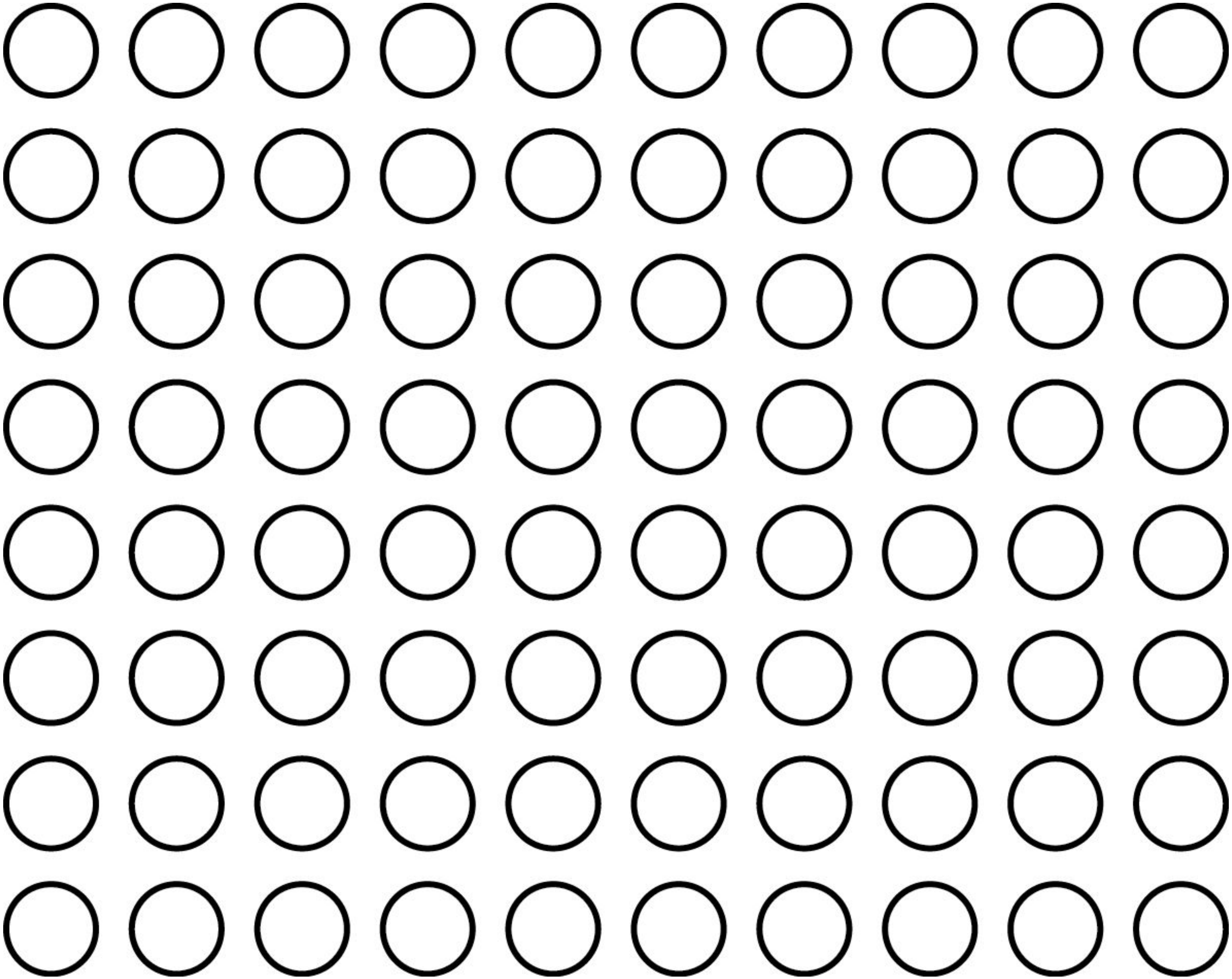
- Vulnerabilities
  - Bugs (programming errors)
  - Bad programming practices
    - Lack of boundary checks on data
  - Design flaws (Spectre)
  - Technology flaws (bit flipping)

Weird machines

- A computer with 1GB RAM, 32x32bits registers
  - 2^{30} * 8 bits = 2^{33} bits
  - 32 * 32 bits = 1024 = 2^{10} bits
    - 2^{33} * 2^{10} = 2^{43} storage cells
    - 1 or 0 -> 2 * 2^{43} = 2^{44} possibilities

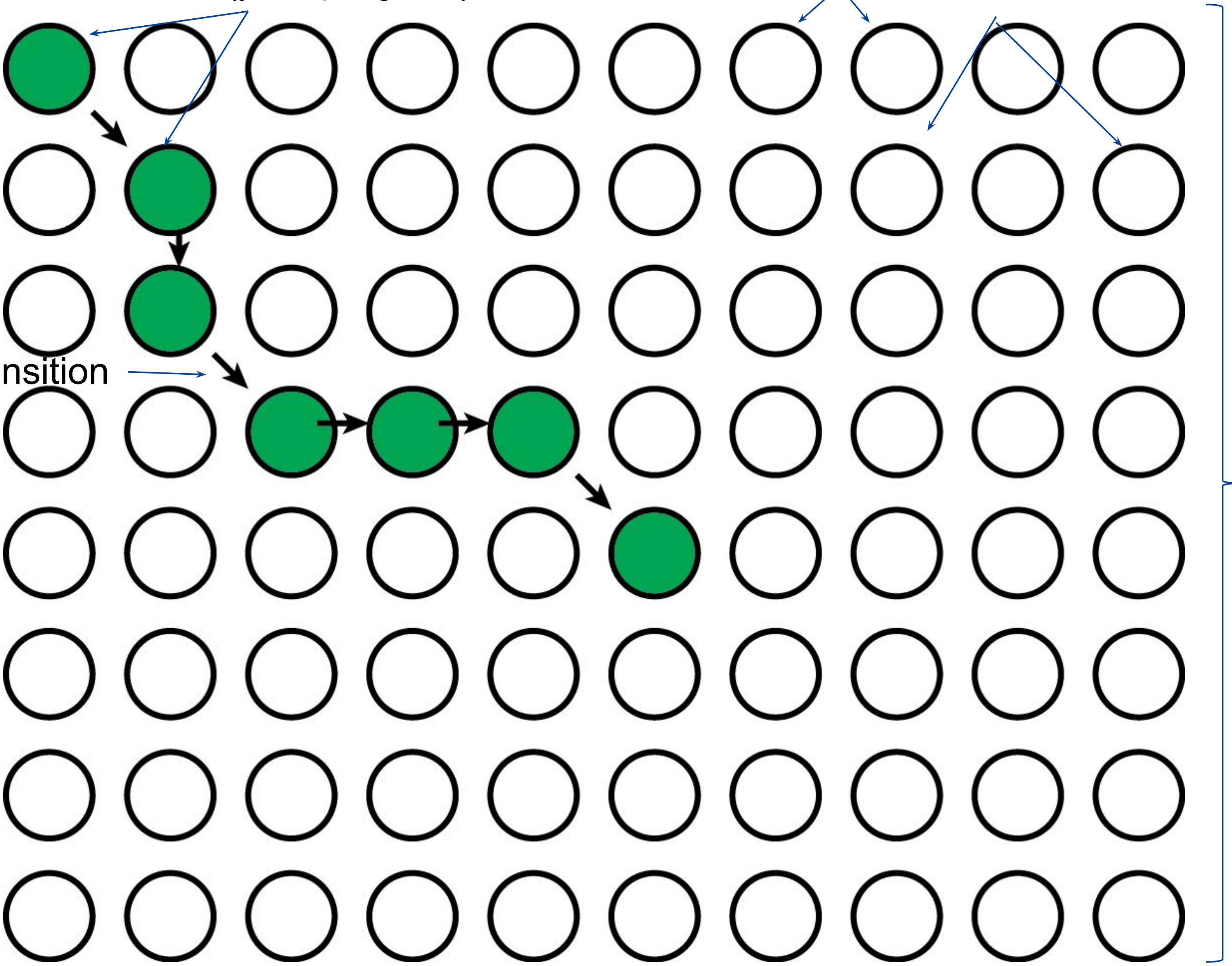
17,592,186,044,416 possible configurations

Sea of states



Sane states (your program)

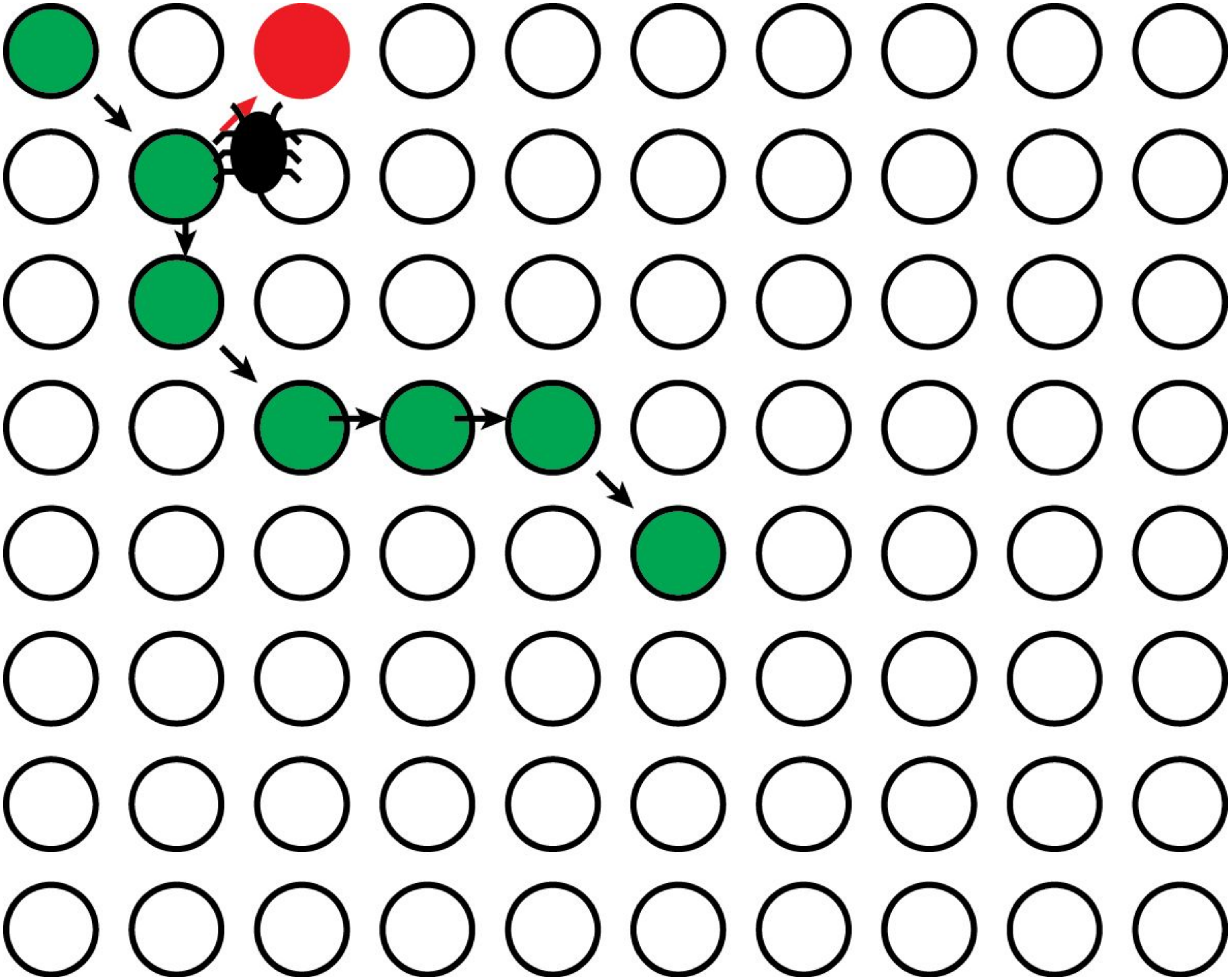
Undefined states

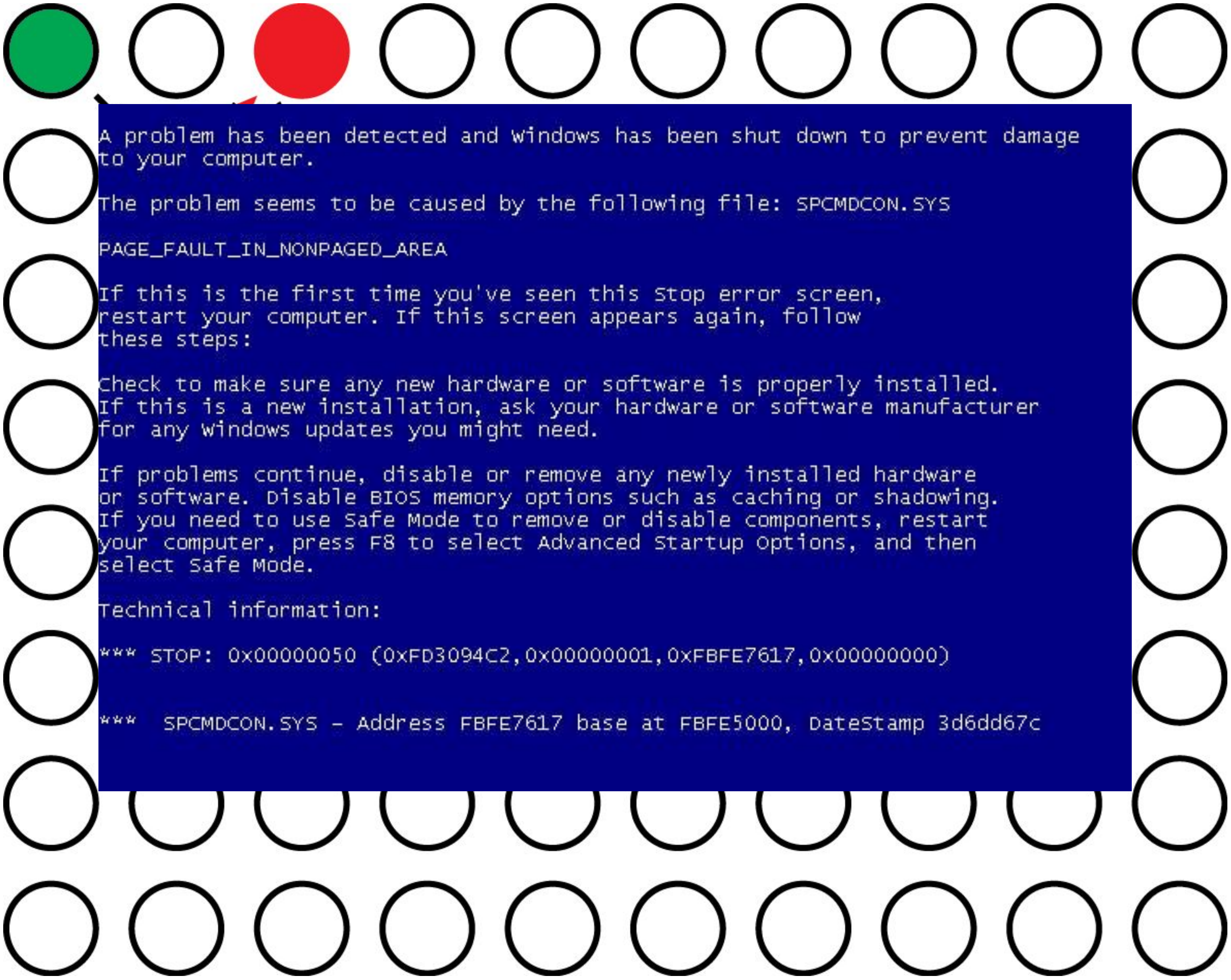


State transition

All machine states

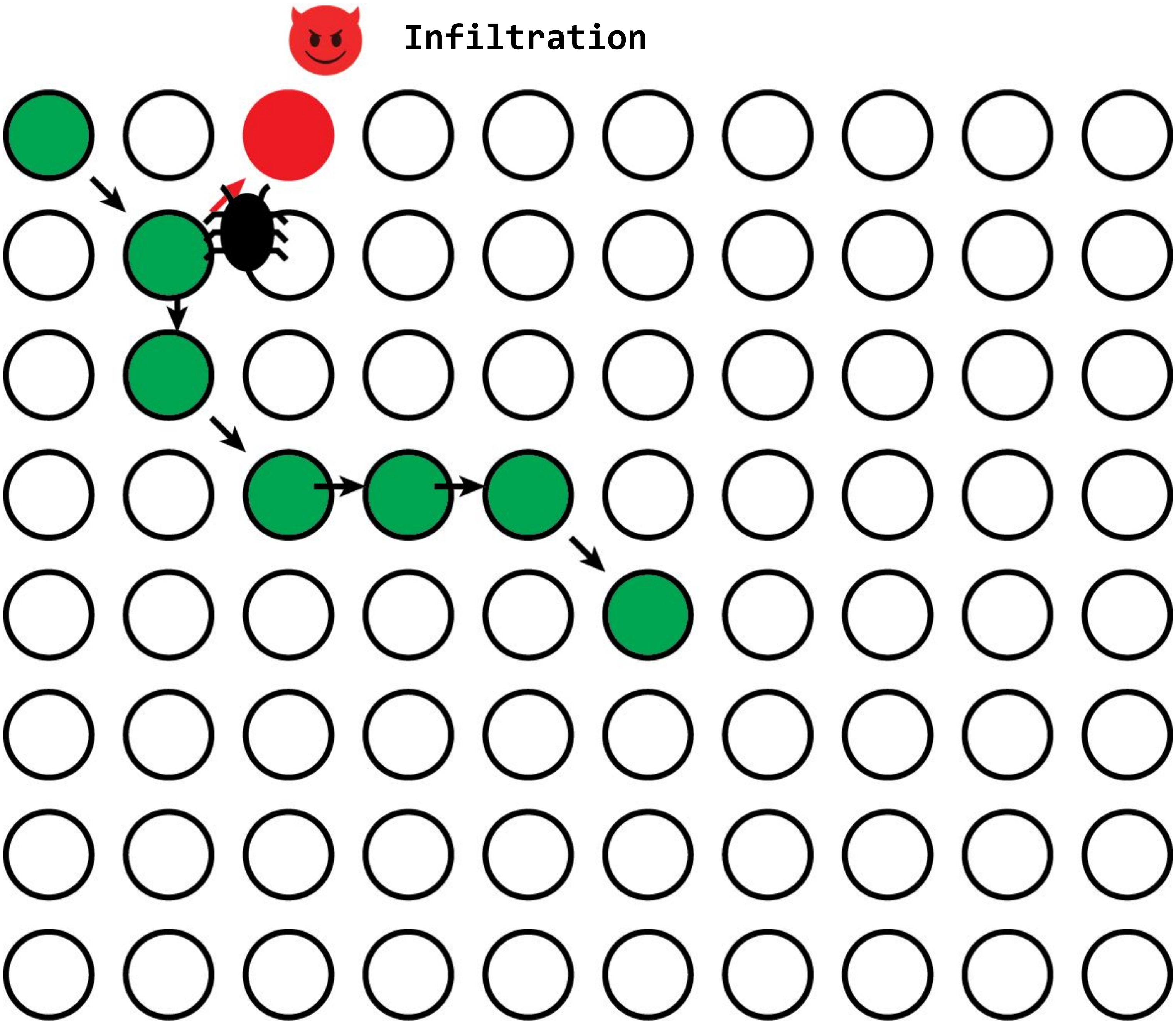
Case 1:
Bugs

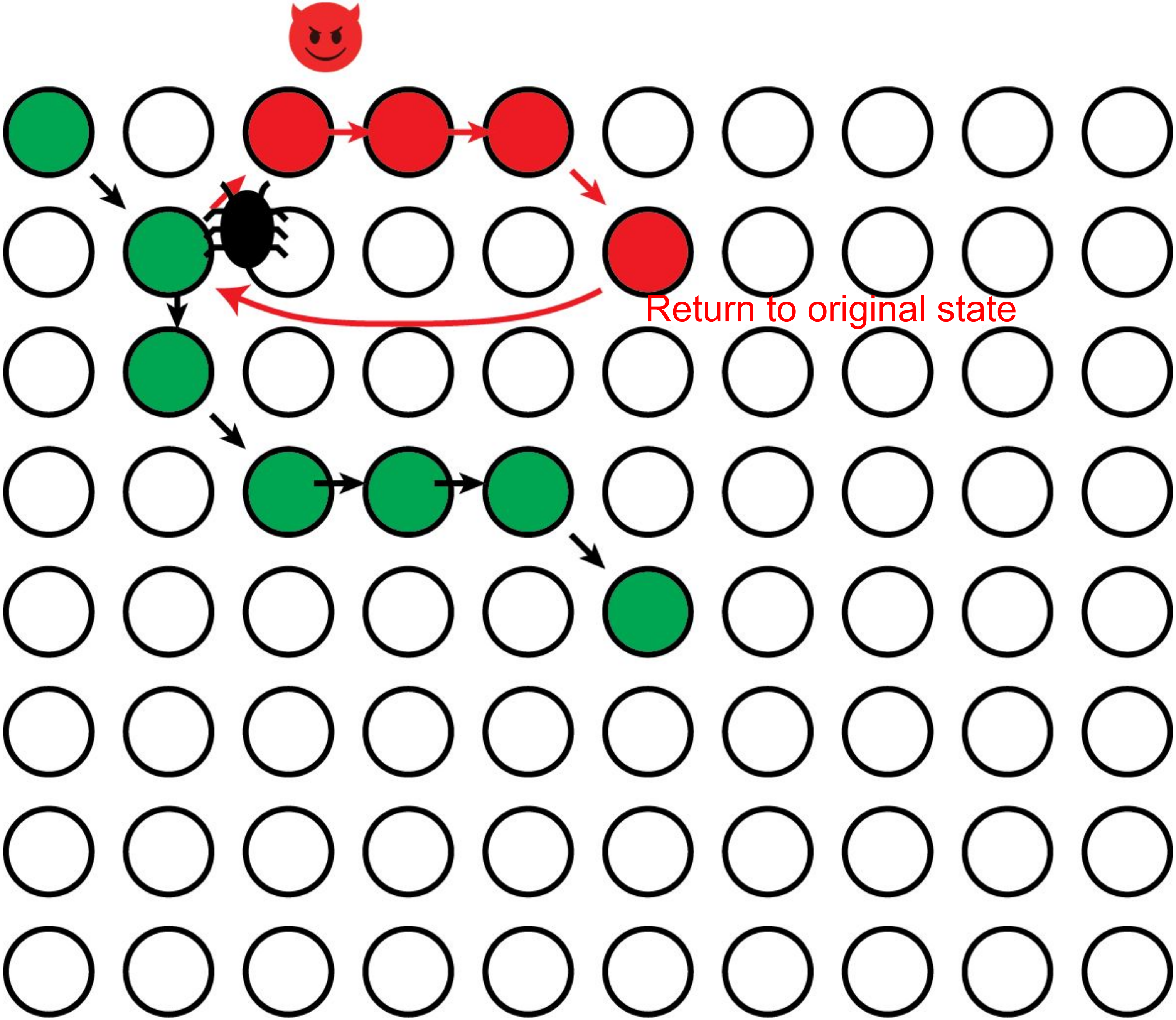




```
A problem has been detected and Windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)


***  SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
```

Infiltration

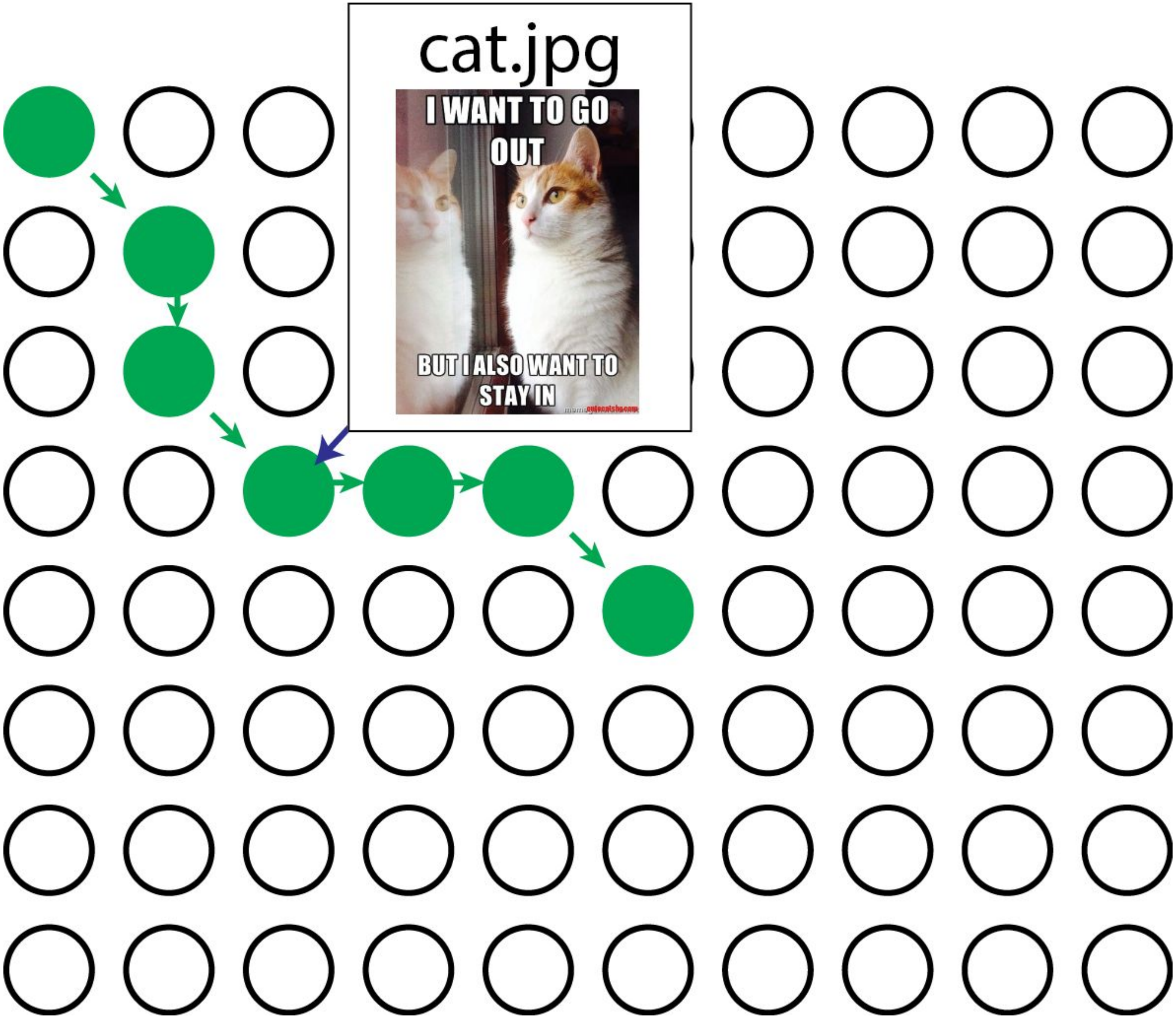


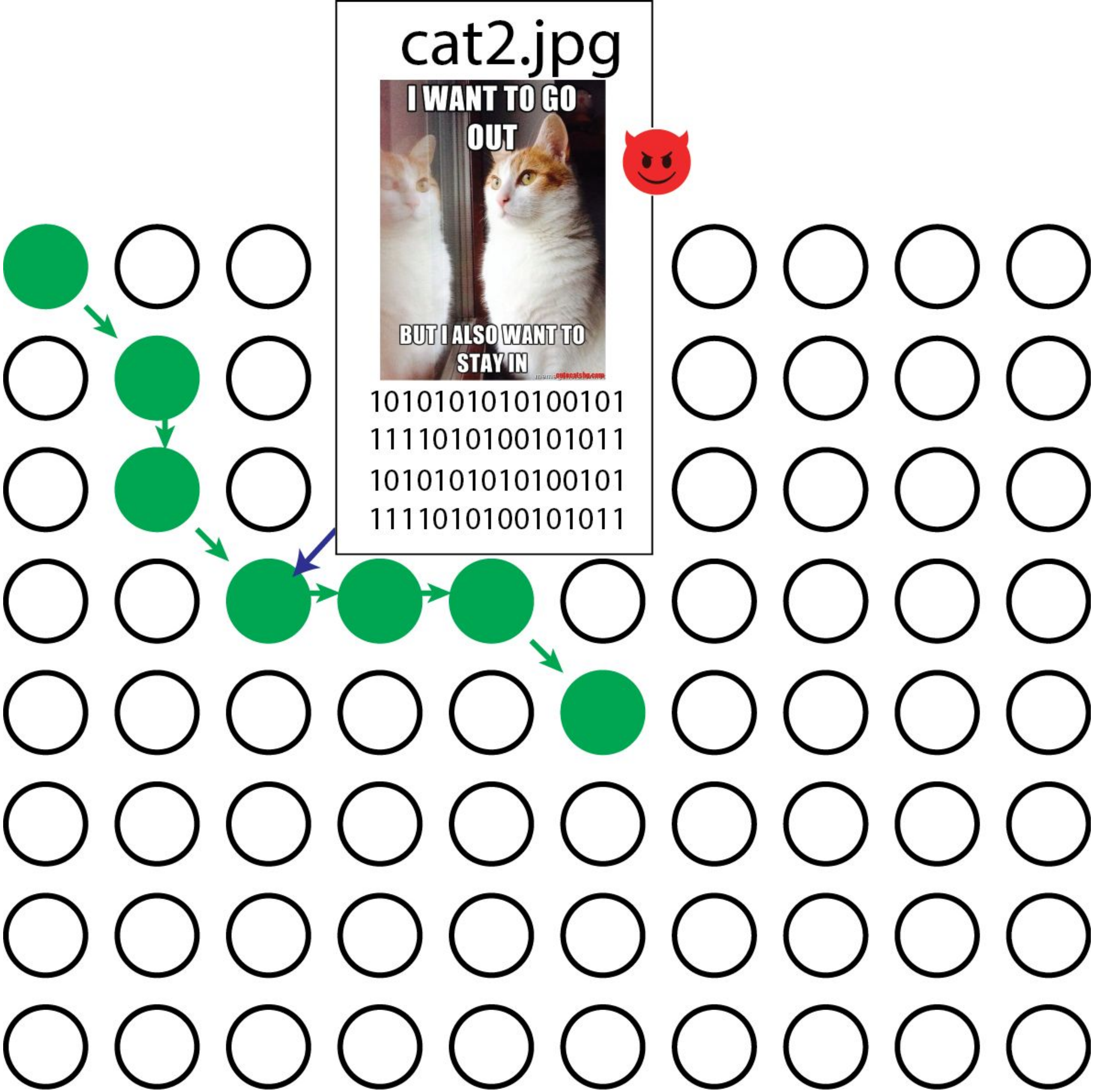


Return to original state

Case 2:
Crafted inputs

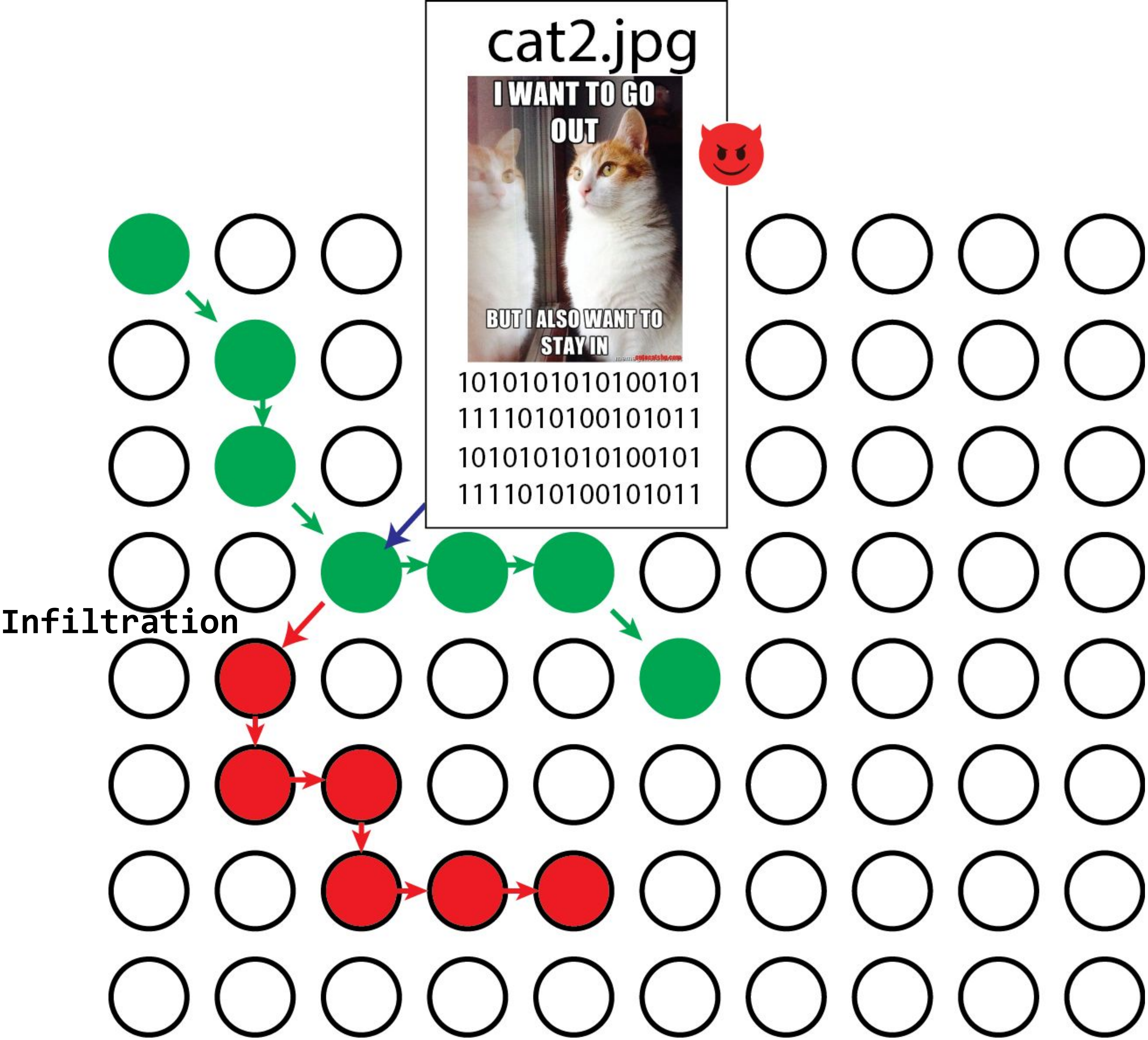
(and bad programming practices)


cat.jpg
I WANT TO GO OUT
BUT I ALSO WANT TO STAY IN



cat2.jpg

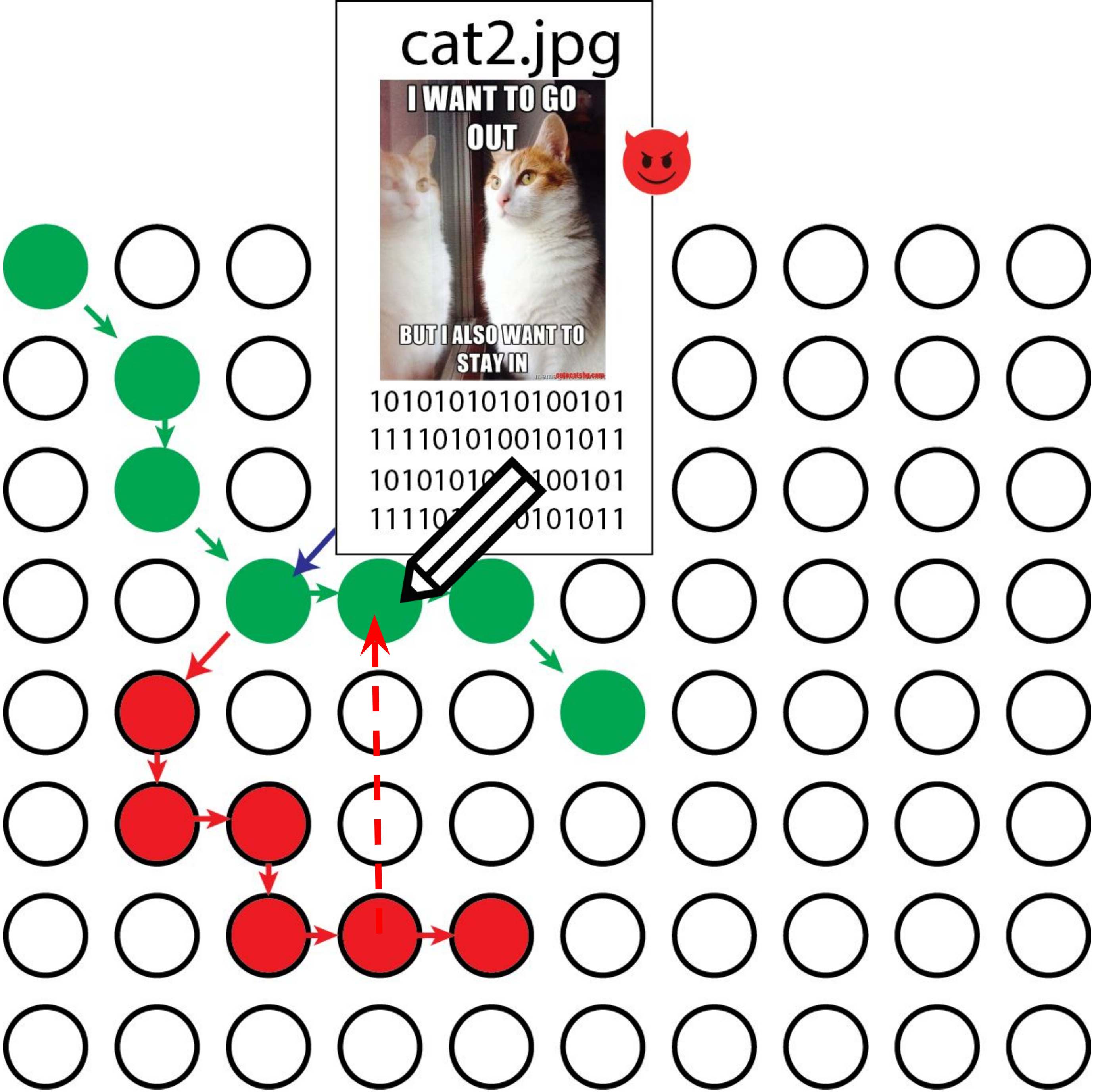
1010101010100101
1111010100101011
1010101010100101
1111010100101011

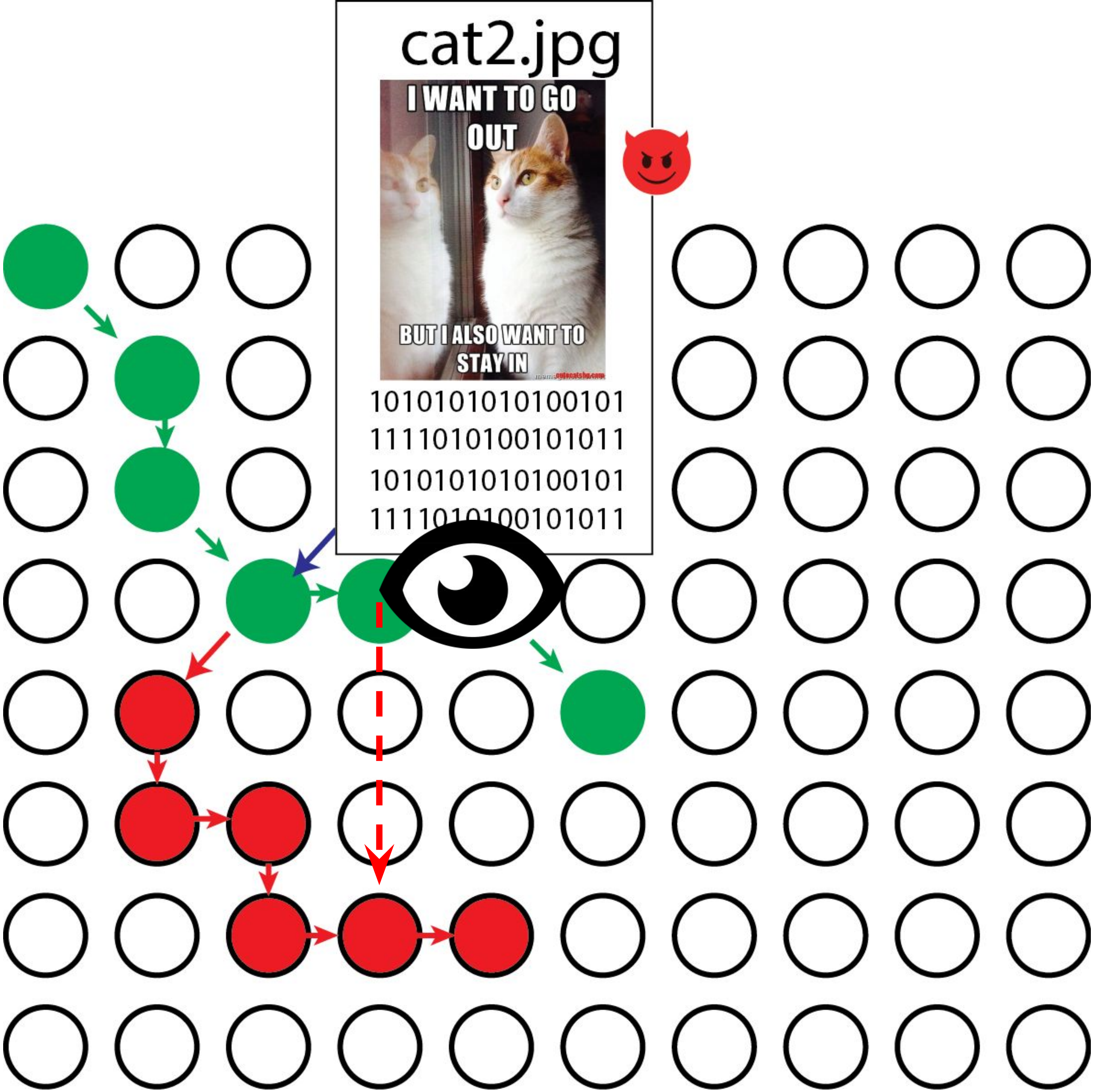


cat2.jpg

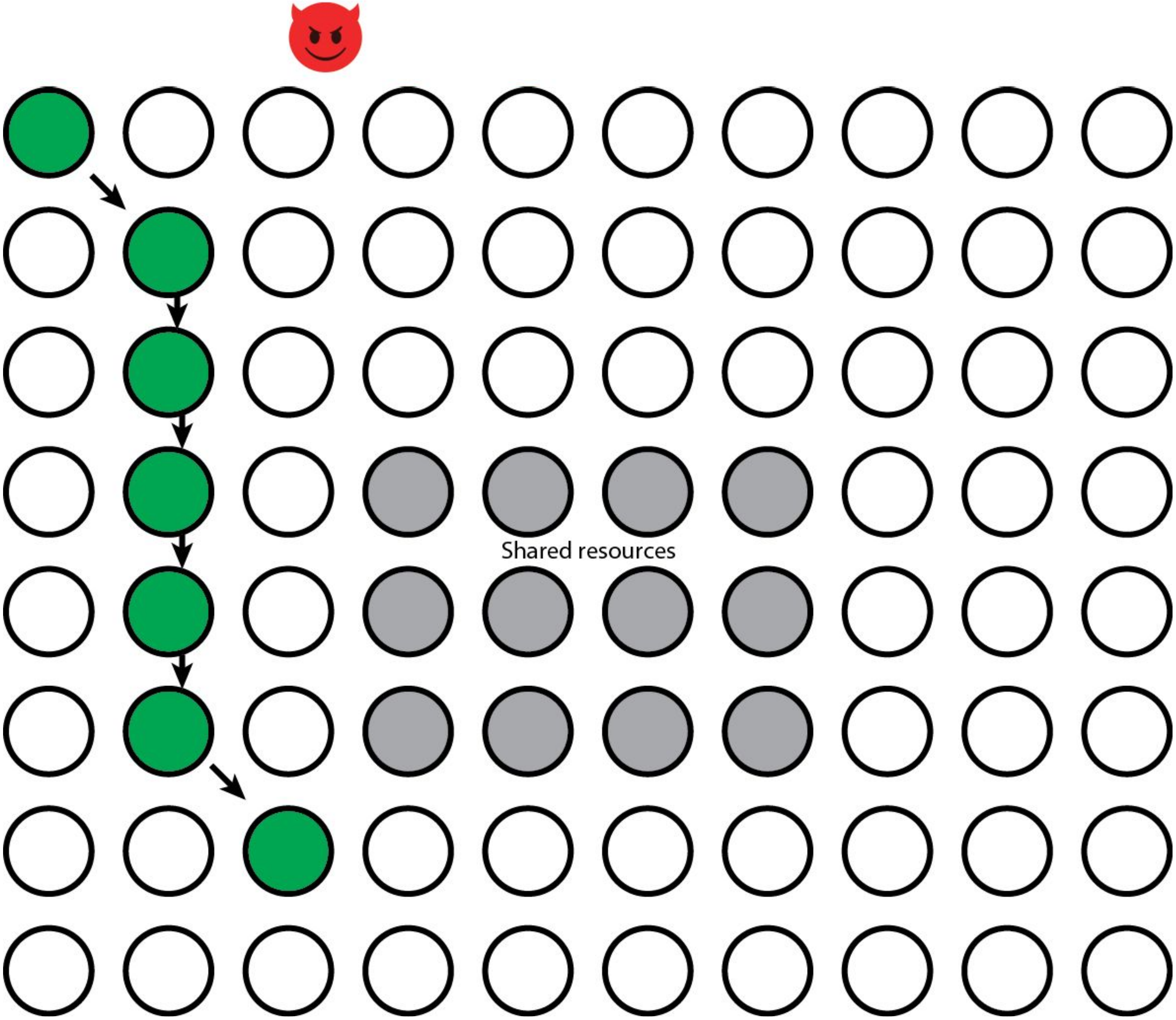
1010101010100101
1111010100101011
1010101010100101
1111010100101011

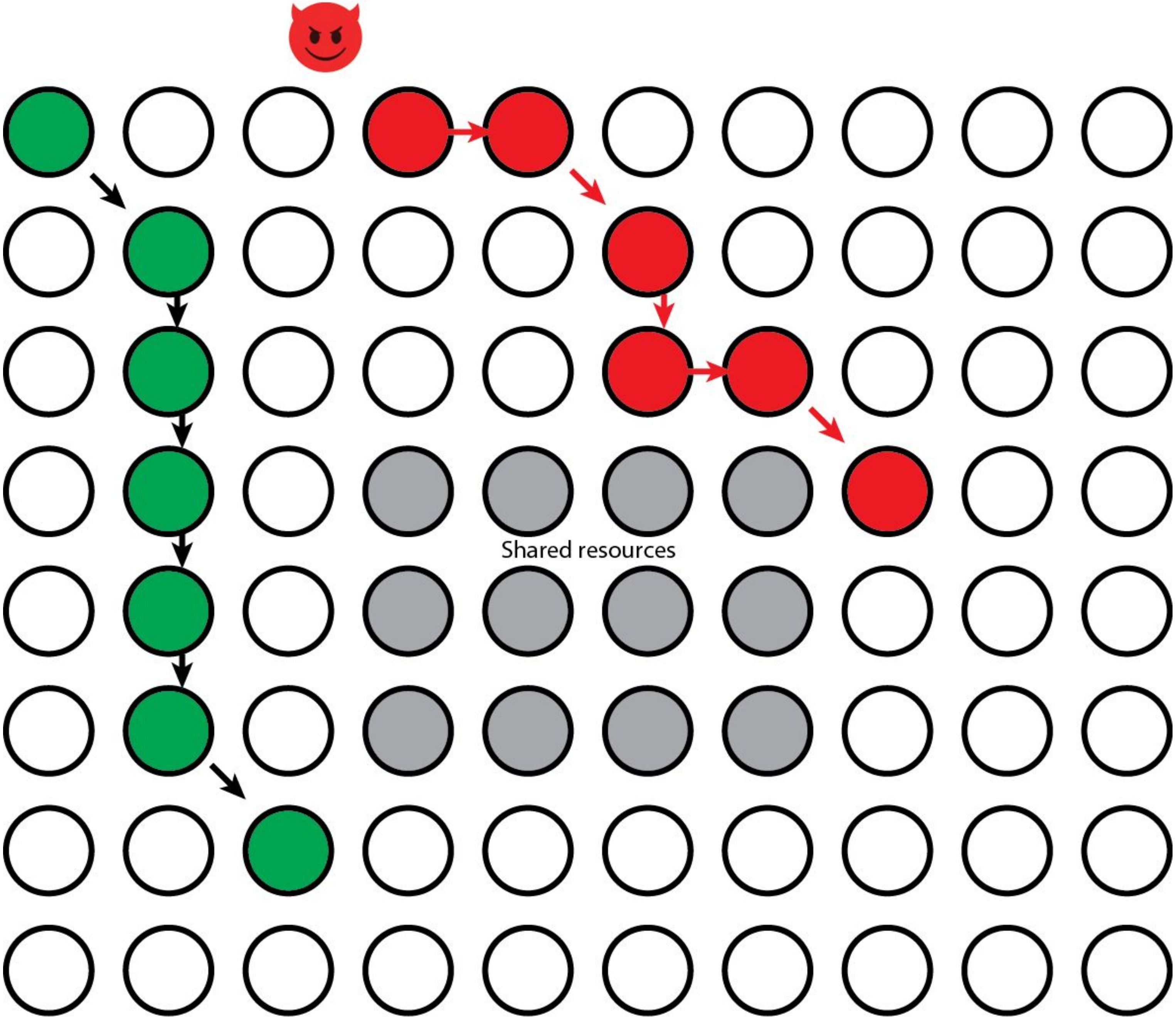
Infiltration

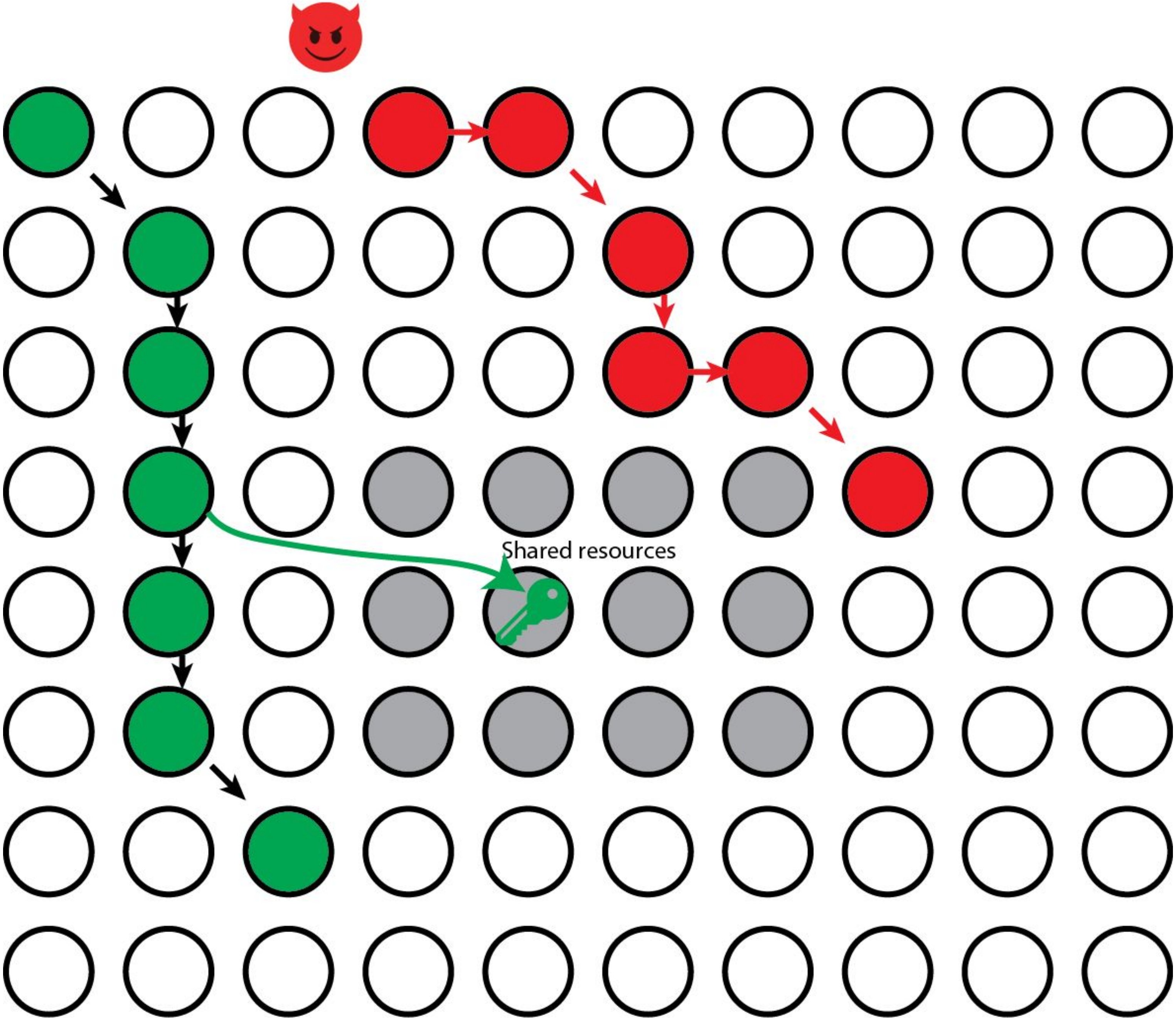

cat2.jpg
I WANT TO GO OUT
BUT I ALSO WANT TO STAY IN
1010101010100101
1111010100101011
1010101010100101
11110 0101011


cat2.jpg
I WANT TO GO OUT
BUT I ALSO WANT TO STAY IN
1010101010100101
1111010100101011
1010101010100101
1111010100101011

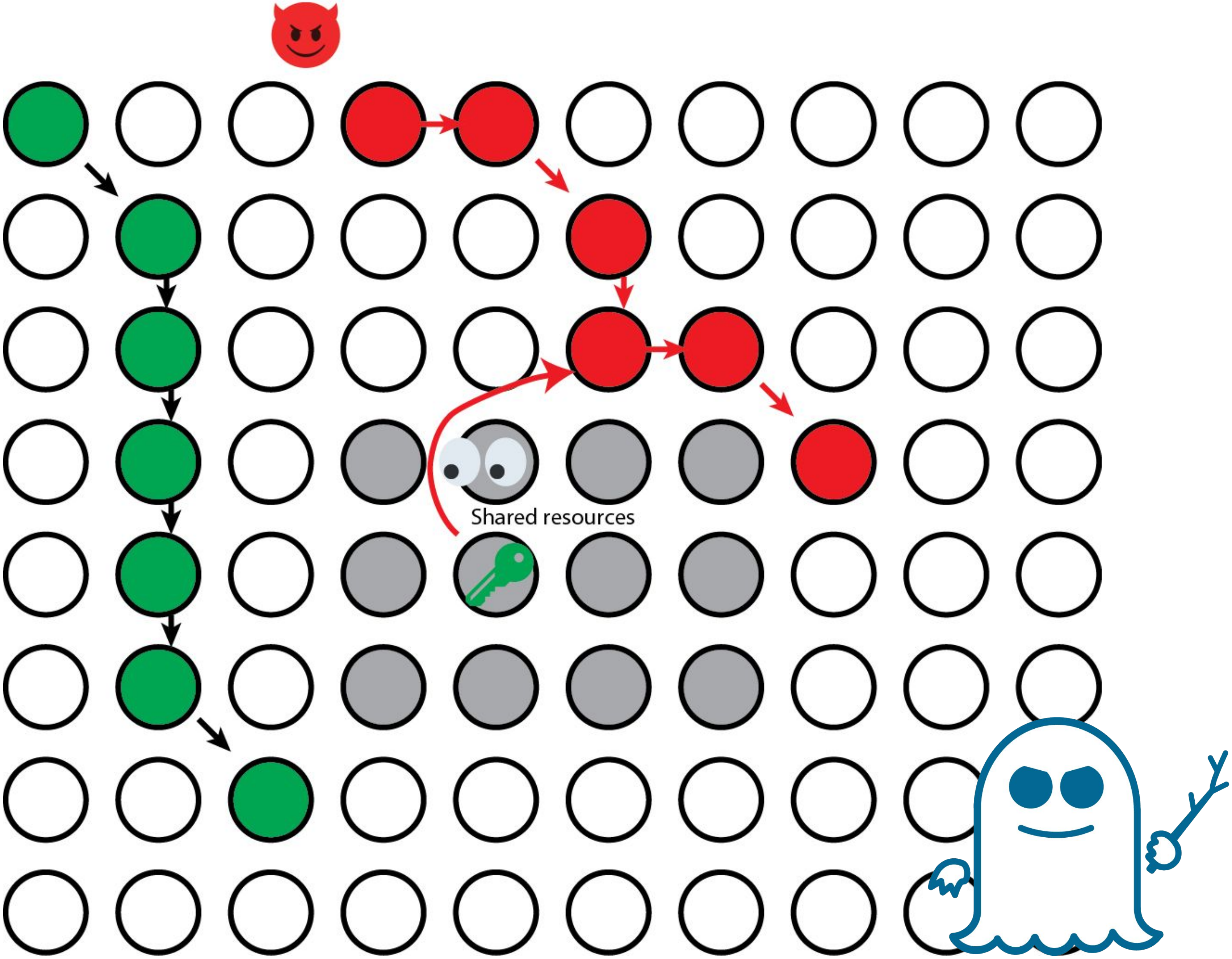
Case 3:
Design flaws


Shared resources


Shared resources



Shared resources


Shared resources

Infiltrate – modify - read

- Attackers should be able to
  - Read/write arbitrary memory locations
    - Directly
    - Indirectly (side-channels)
  - Modify the execution order of a victim program
  - Infiltrate without detection, through vulnerabilities (side-effects)
- Can we compute without
  - Read/write,
  - Control operations,
  - Side-effects?

YES!

Foundations

- *Computational Models*
  - Finite State Machines (time-dependent, stateful)
  - Turing Machines (time-dependent, stateful)
  - Combinatory Logic ≡ λ-Calculus (time-independent, stateless)

CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

- **Functional programming**
  - Immutable data
  - Pure (as in maths) functions
  - Restricted side-effects
  - Referential transparency
  - Mathematical reasoning
    - Function composition: f(g(x))
  - List operations – no overflow issues

Functional Programming Architectures

- Not a new idea
  - (1979) Turing Award – John Backus: Can programming be liberated from the von Neumann style?
  - (1979) University of Kent – David Turner: Miranda
  - (80s) Symbolics’ LISP machines
  - (1984) Cambridge University – W. Stoye : SKIM I and II
  - (1985) Burroughs’ NORMA
  - (1988) University College London – S.Peyton Jones: GRIP
  - ...

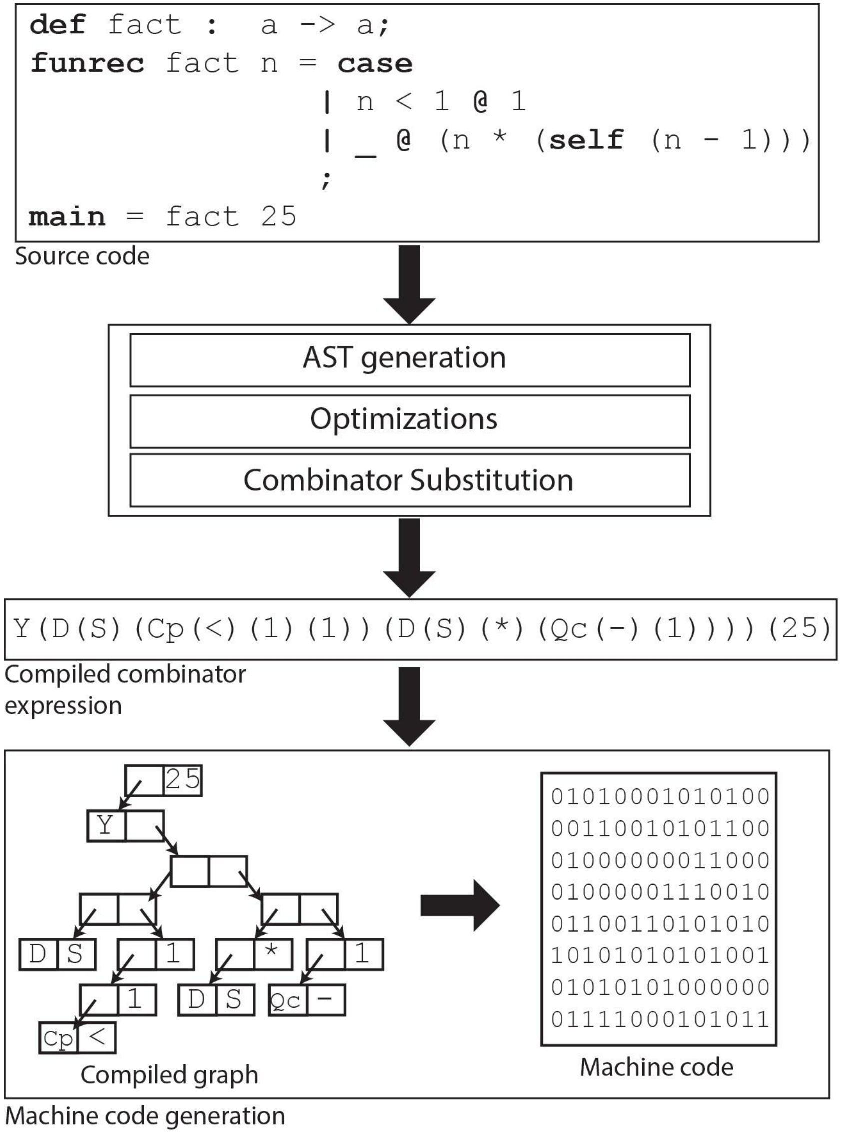
Functional Programming Architectures

- Not a new idea
  - (1979) Turing Award – John Backus: Can programming be liberated from the von Neumann style?
  - (1979) University of Kent – David Turner: Miranda
  - (80s) Symbolics’ LISP machines
  - (1984) Cambridge University – W. Stoye : SKIM I and II
  - (1985) Burroughs’ NORMA
  - (1988) University College London – S.Peyton Jones: GRIP
  - ...

  - The rise of PCs shifts research to compiling functional languages
  - (1988) Haskell
  - (1998) Haskell - (standard), Glasgow Haskell Compiler
  - ...
  - (2009) University of York : Reduceron
  - ...

Our approach - *fun*

- Graph reduction of combinator graphs



```
def fact :  a -> a;
funrec fact n = case
               | n < 1 @ 1
               | _ @ (n * (self (n - 1)))
               ;
main = fact 25
```

Source code

AST generation

Optimizations

Combinator Substitution

```
Y(D(S)(Cp(<)(1)(1))(D(S)(*)(Qc(-)(1)))))(25)
```

Compiled combinator expression

Compiled graph

```
01010001010100
00110010101100
01000000011000
01000001110010
01100110101010
10101010101001
01010101000000
01111000101011
```

Machine code

Machine code generation

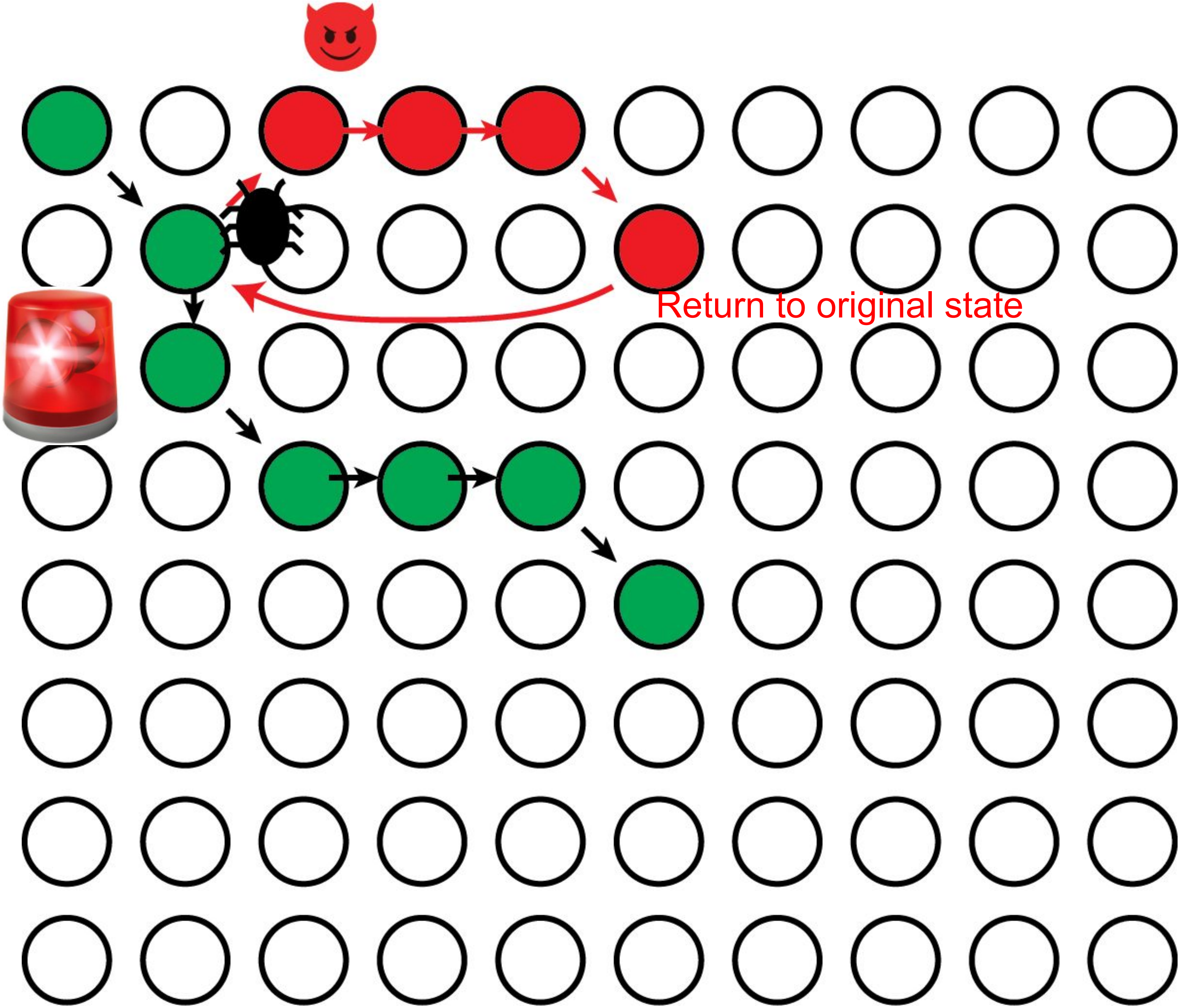
Lazy Functional Programming

- Lazy evaluation
  - Only useful computations are performed – no side-effects
  - State transitions only occur if they contribute to computing the result wanted by the user/programmer.

- Referential transparency
  - G(x) = x + x
  - F(y) = 3y
  - F(G(2)) = F(4) = 12

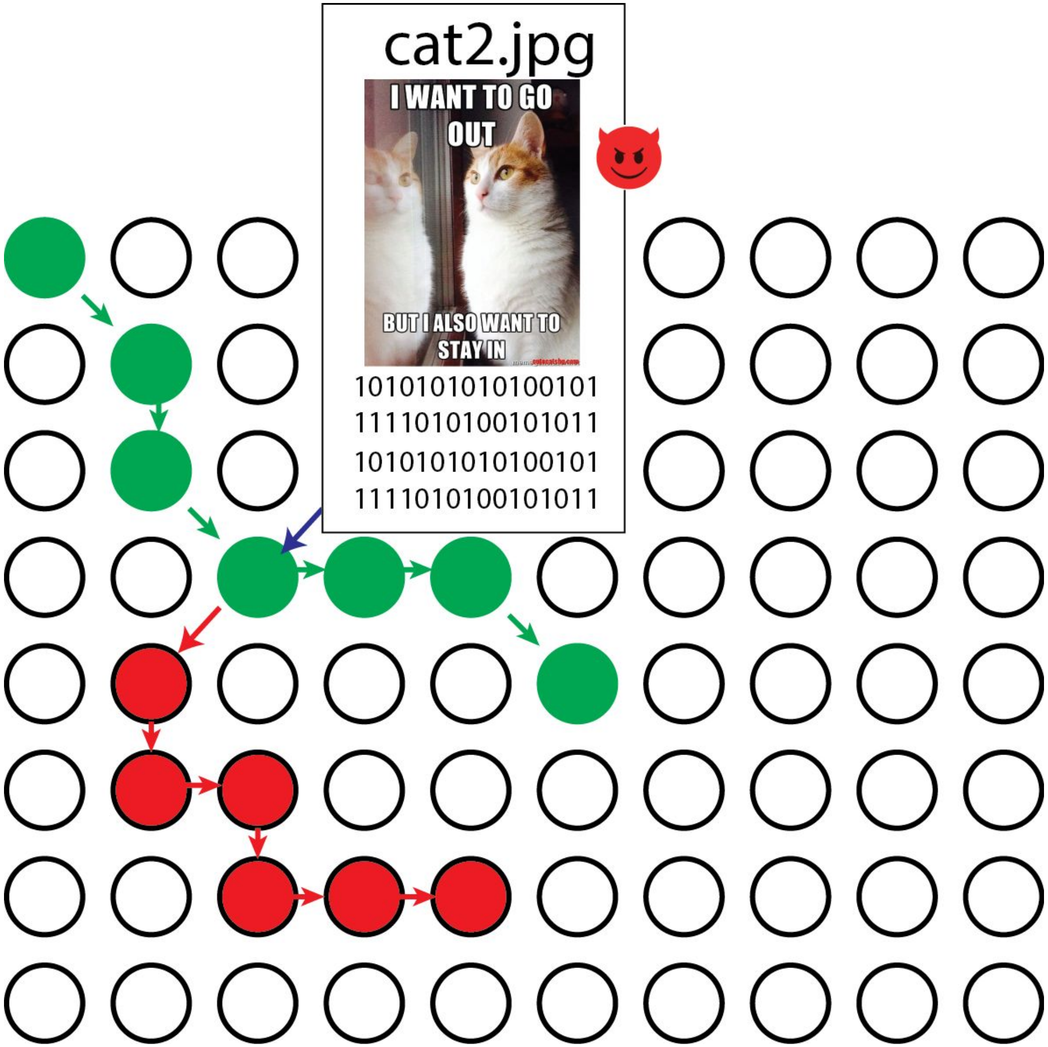
  - M(z) = (Do something evil) and return z

  - F(M(G(2))) = ?
  - M(4) = ? F(M(4)) = ?


Return to original state

Weird Machines in FP

- No variable assignments
  - Can’t read data directly
- Immutable data
  - Can’t modify data
- No control instructions
  - Jumps, branches
  - Can’t modify execution flow
- Pure functions
  - No side-effects
- Lazy evaluation
  - Malware is not useful!



Project Status

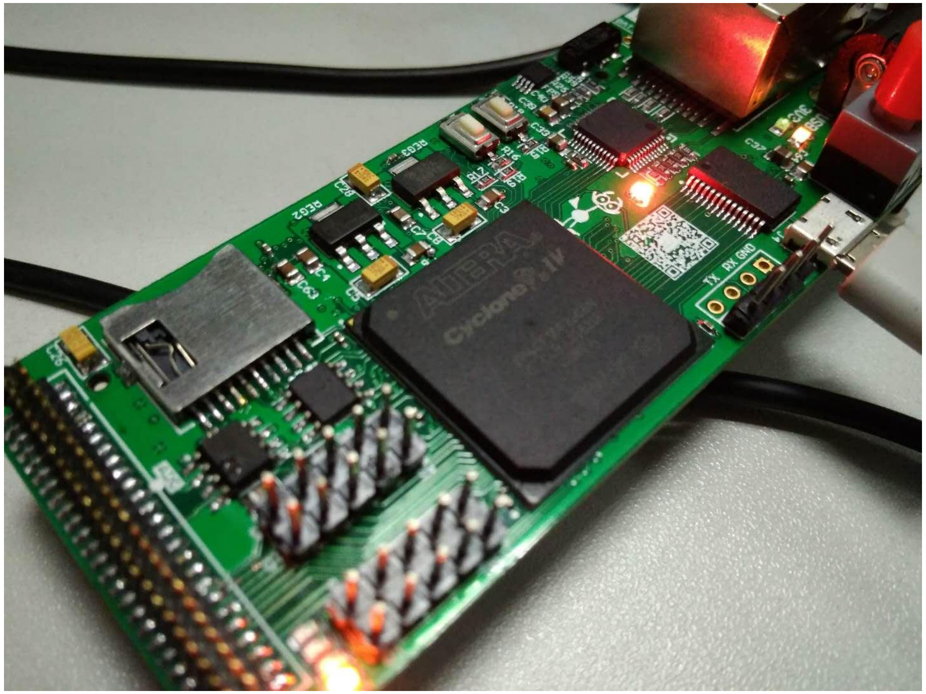
- FPGA prototyping stage
- Stable compiler for Wu

- Operational system kernel - *funk*
  - Under development

- Device drivers
  - Ethernet
  - TCP/IP stack

- Benchmarking
  - Evaluating Performance

- Hacking
  - Trying to break
  - Searching for a weird machine that violates confidentiality, integrity or availability.



交通大學