RANDOM GRAPHS IN SECURITY AND PRIVACY

Adi Shamir

The Weizmann Institute

Israel

December 5-th 2009

ICITS

The Two Types of Crypto Research:

Information-theoretic:

Assumes that:

- primitives are perfect
- opponent all powerful

Tries to bound:

- Statistical properties
- Information derived

Examples:

- OTP
- secret sharing

Complexity-theoretic:

Assumes that:

- primitives are imperfect
- opponent is bounded

Tries to bound:

- runtime of attack
- memory required

Examples:

- AES
- RSA key exchange

But there is a third type, which combines the two

The Two Types of Crypto Research:

Information-theoretic:

Assumes that:

primitives are perfect

Tries to bound:

Complexity-theoretic:

Assumes that:

opponent is bounded

Tries to bound:

runtime of attack

memory required

Examples:

Finding collisions or inverting edges in random graphs

Cryptography and Randomness:

The notion of random functions (oracles) over the finite domain {0,1,2,…,N-1}:

- truly random when applied to fresh inputs
- consistent when applied to previously used inputs

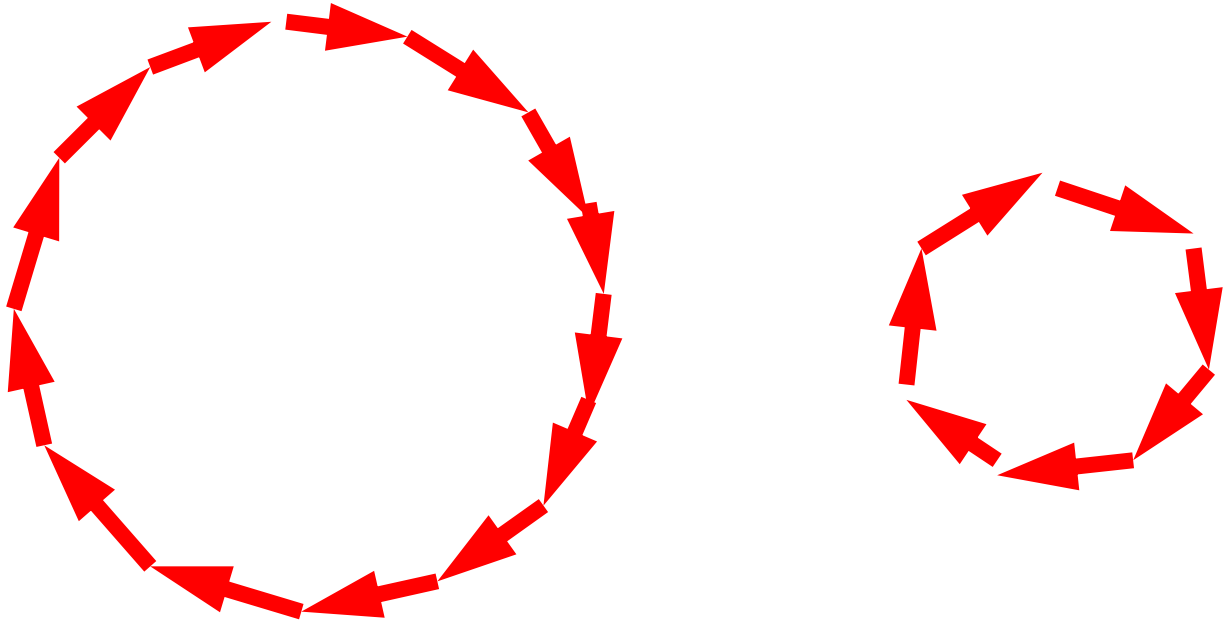
f(0)=37
f(1)=92
f(2)=78

…

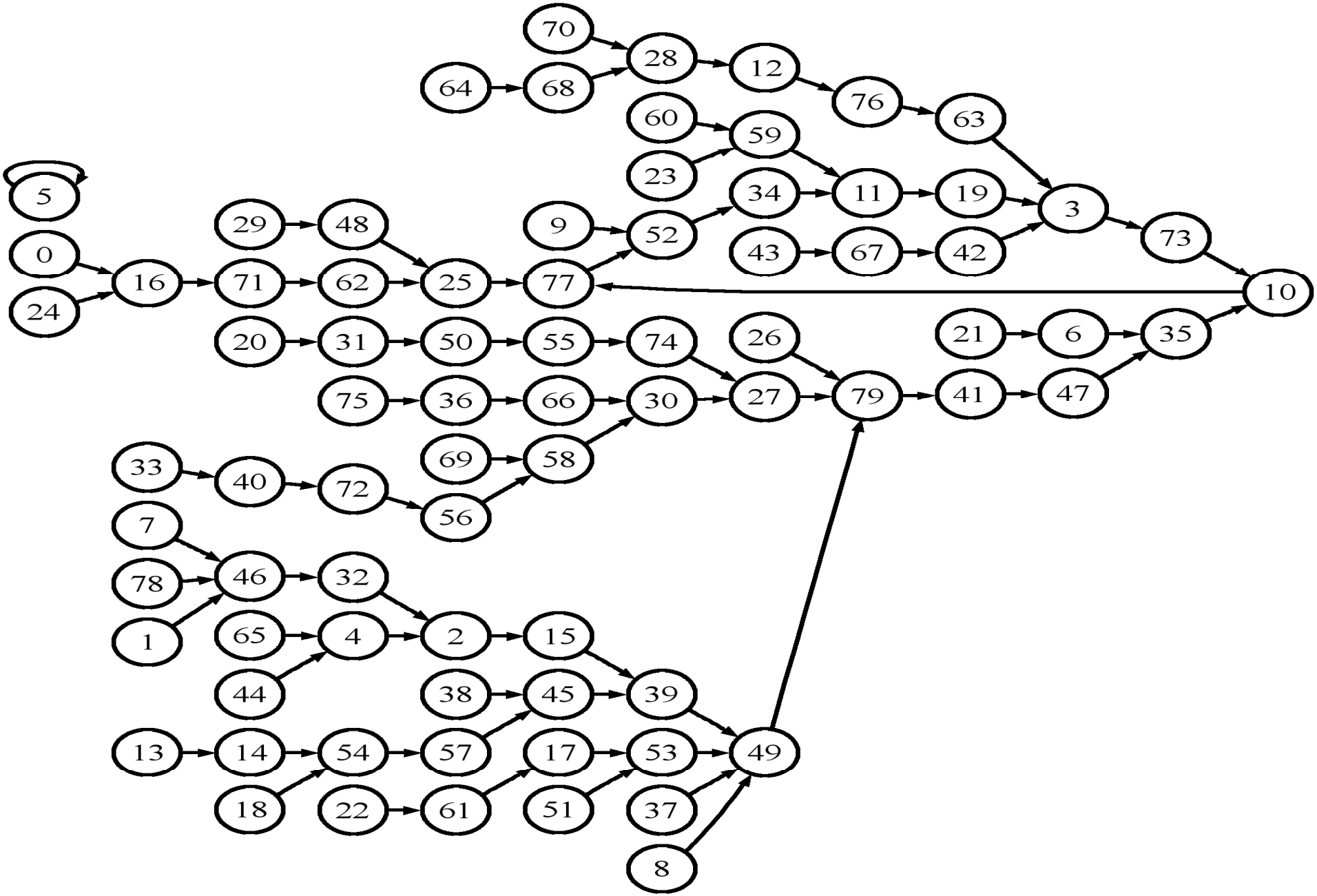
The random graph associated with f: $x \rightarrow f(x)$

Cryptography and Randomness:

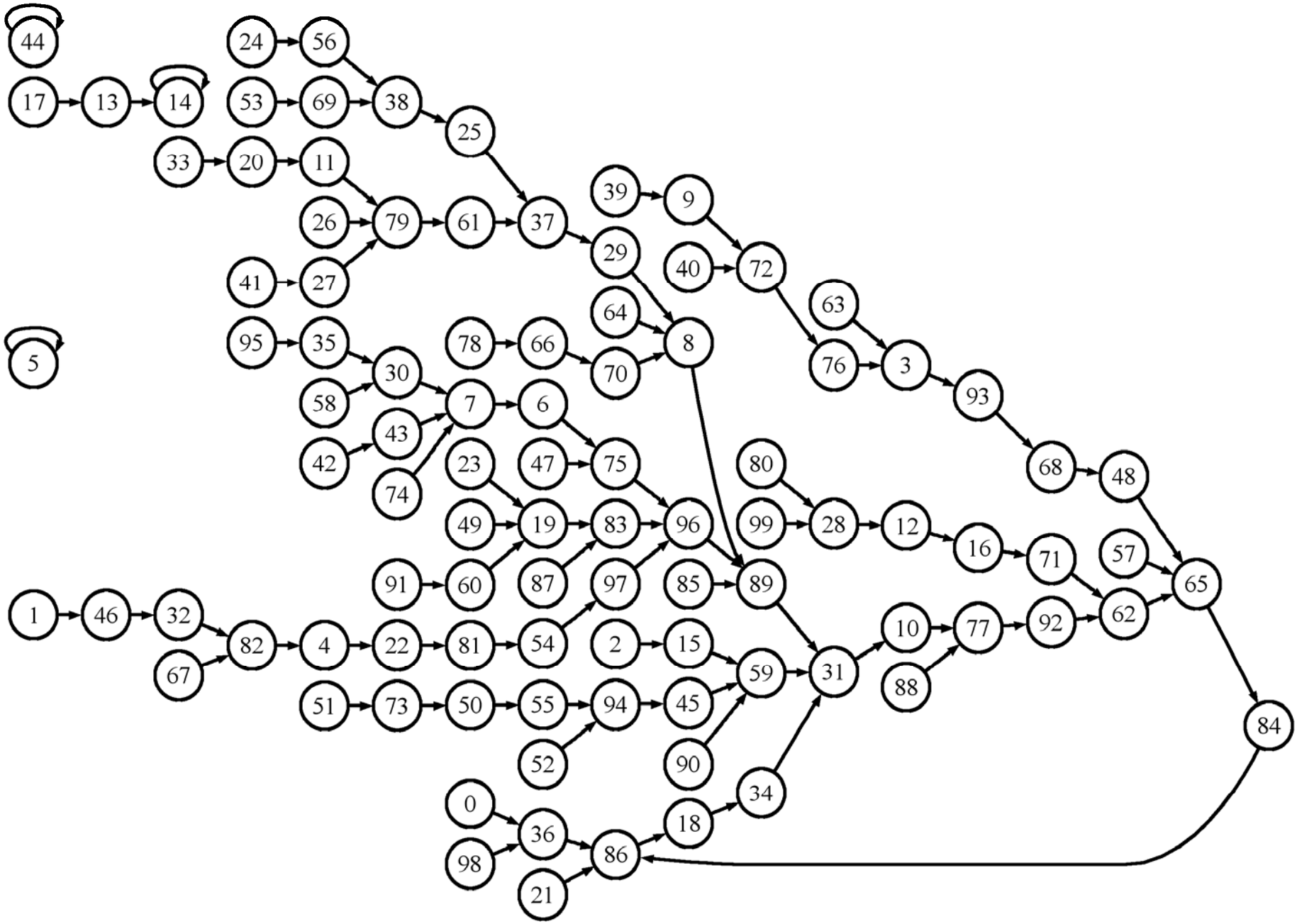
When the function f is a permutation, its associated graph G is quite boring:



Random Graphs Have Much More Interesting Structure:



Another Example of a Random Graph:



Cryptography and Randomness:

There is a huge literature on:

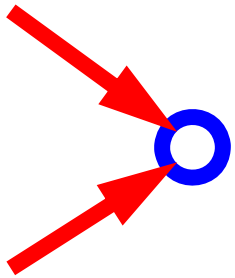
The distribution of component sizes, tree sizes, cycle sizes, vertex in-degrees, number of predecessors, etc.

In this talk I'll concentrate on some algorithmic results from the last 5 years related to collision finding and inversion algorithms

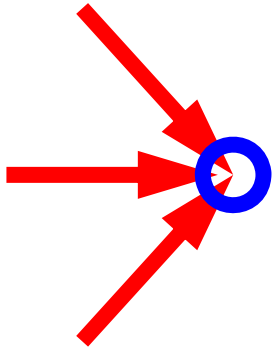
Note that in cryptanalysis, constants are important!

Interesting algorithmic problems in breaking the security of hash functions:

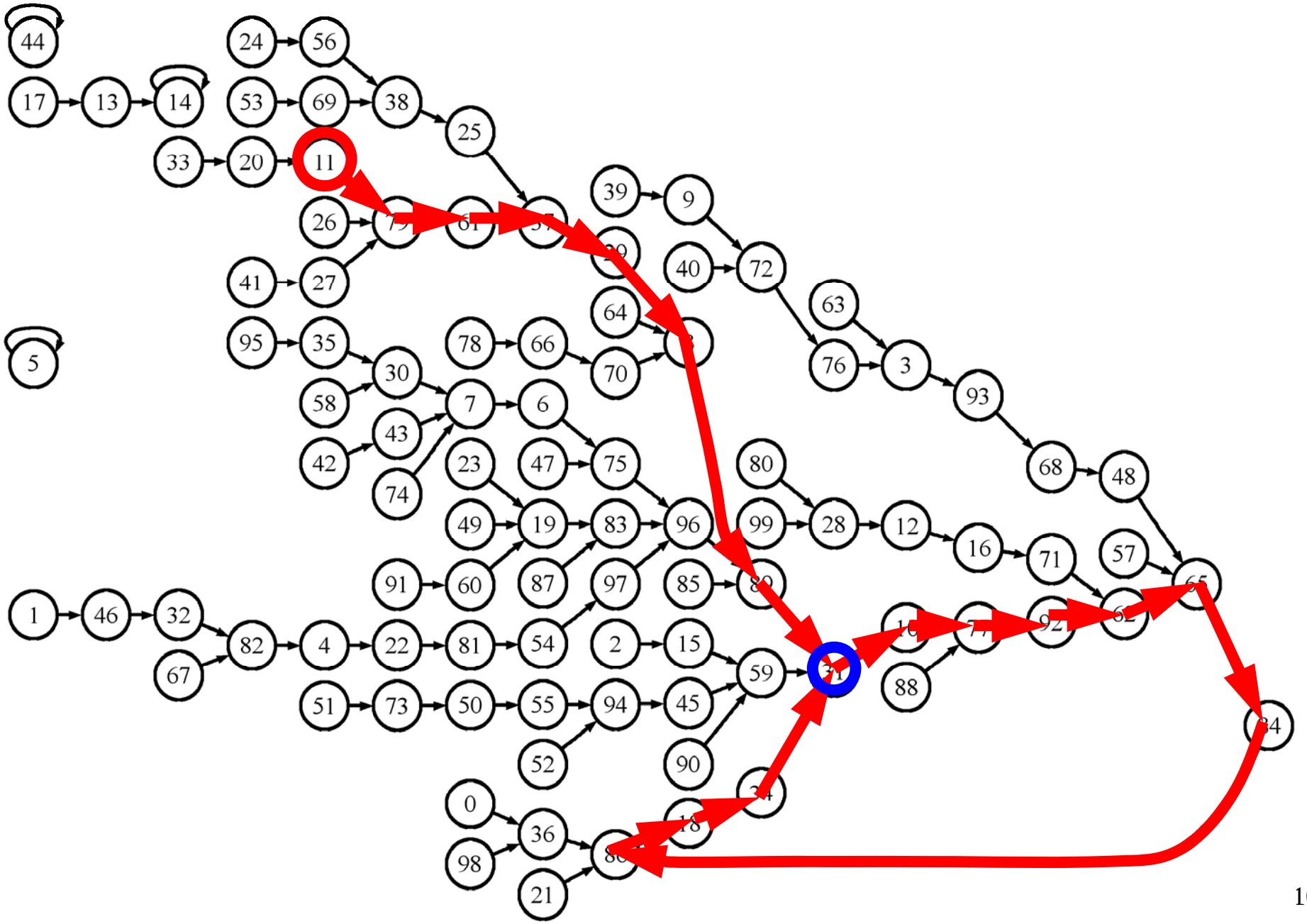
Find some simple collision (assuming that we can only choose random points and move forward along edges):



- Find some multicollision (useful eg in breaking concatenated hash fn's):



A random path in a random graph defines a collision:



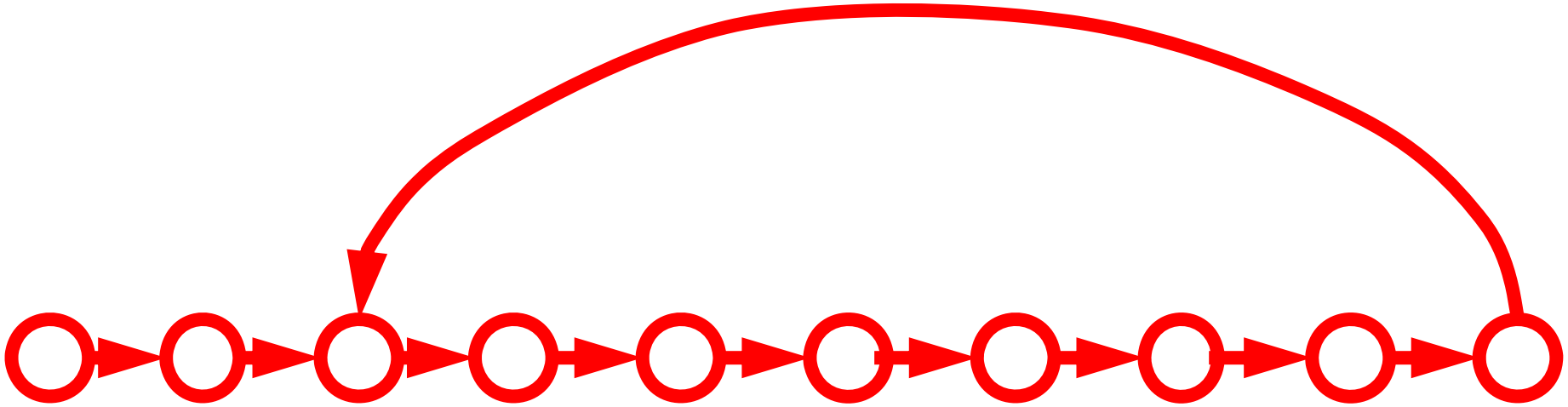
Finding such collisions is a very well studied problem:

- Floyd
- Pollard
- Brent
- Yao
- …

And yet there are new surprising ideas!

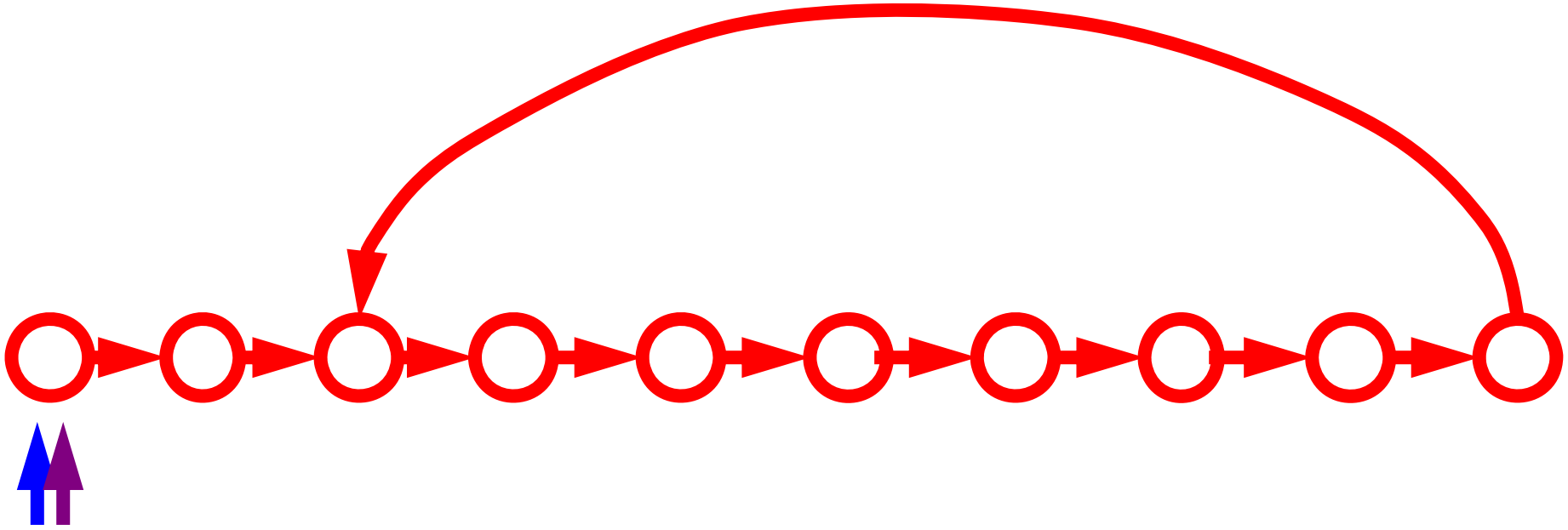
The best known technique: Floyd's two finger algorithm

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide



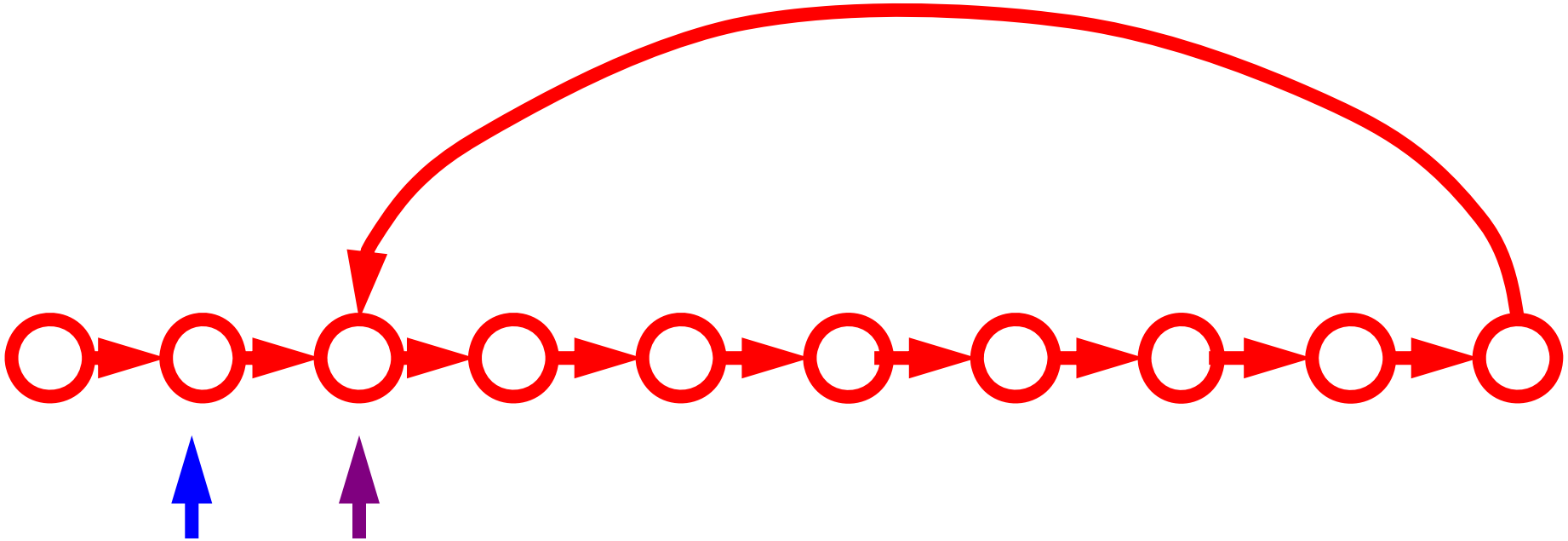
Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide



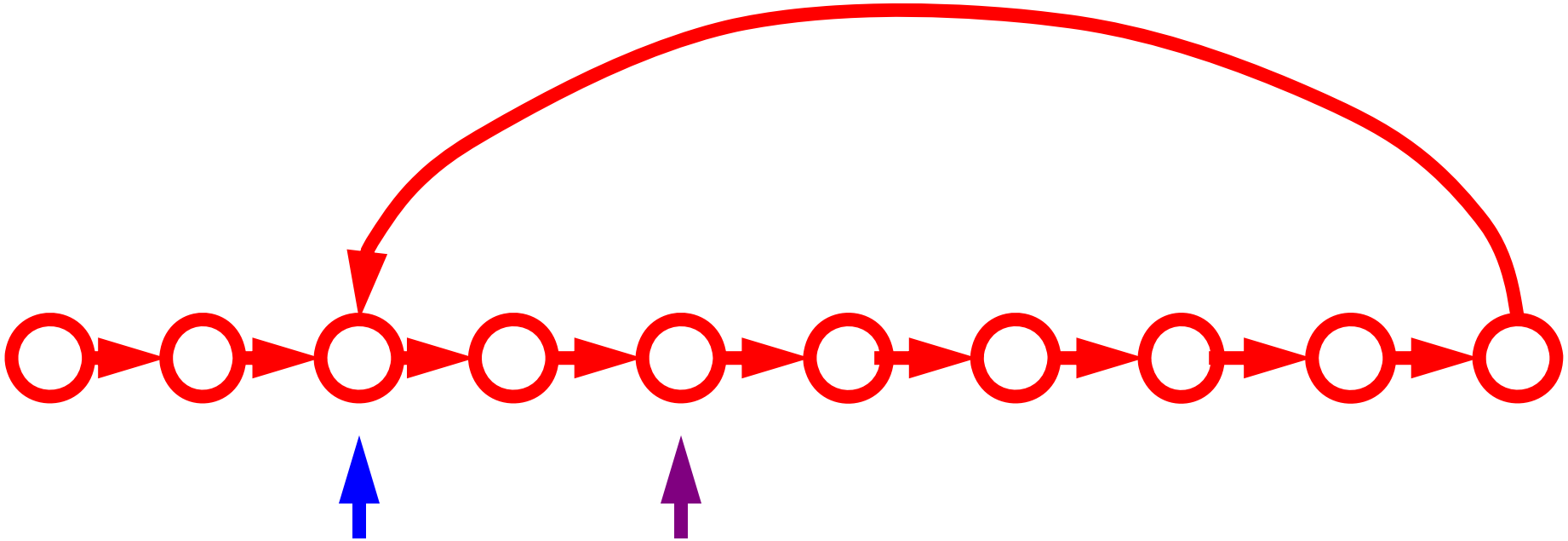
Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide



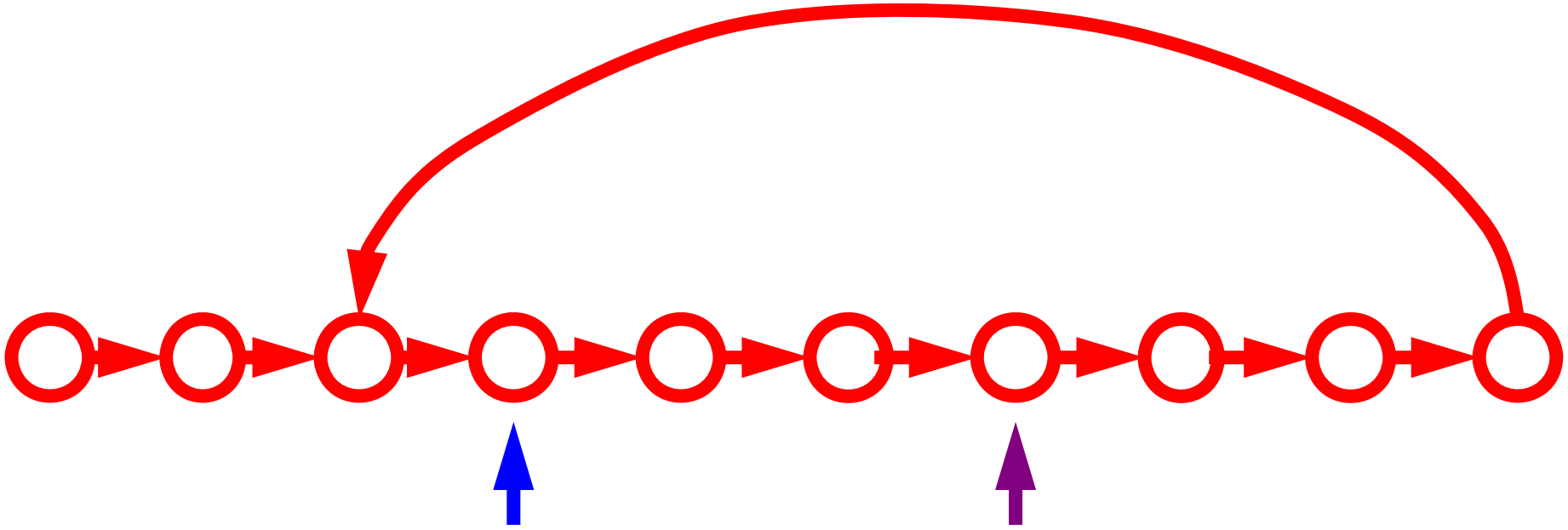
Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide



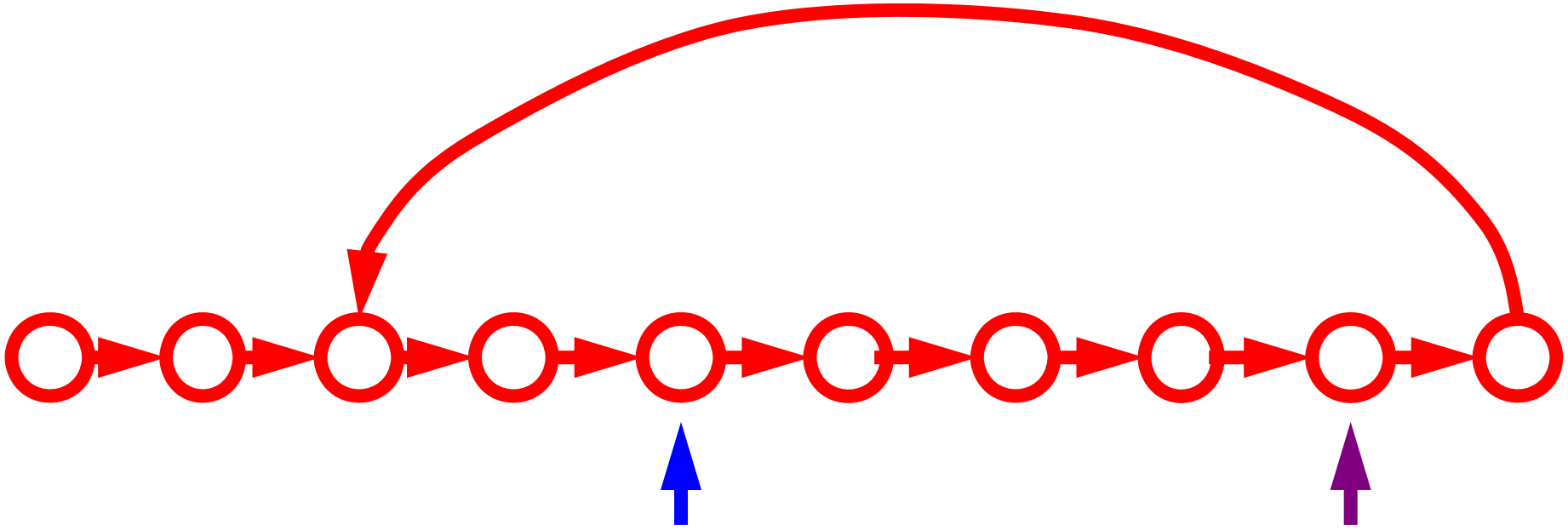
Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide



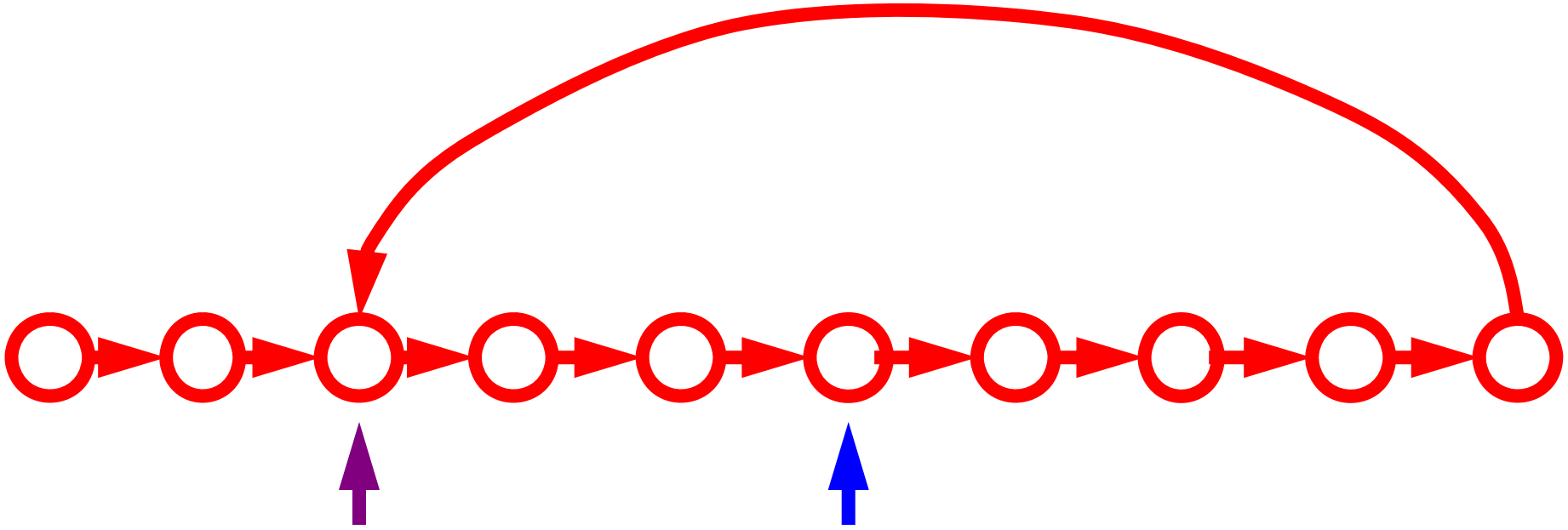
Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide



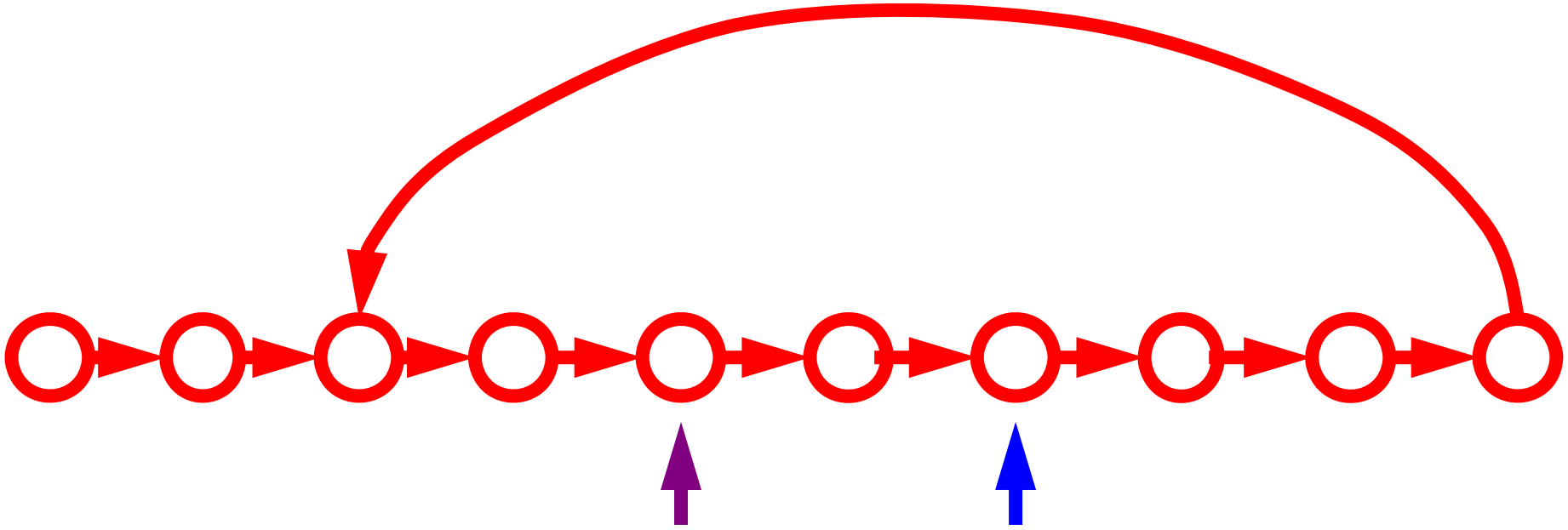
Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide



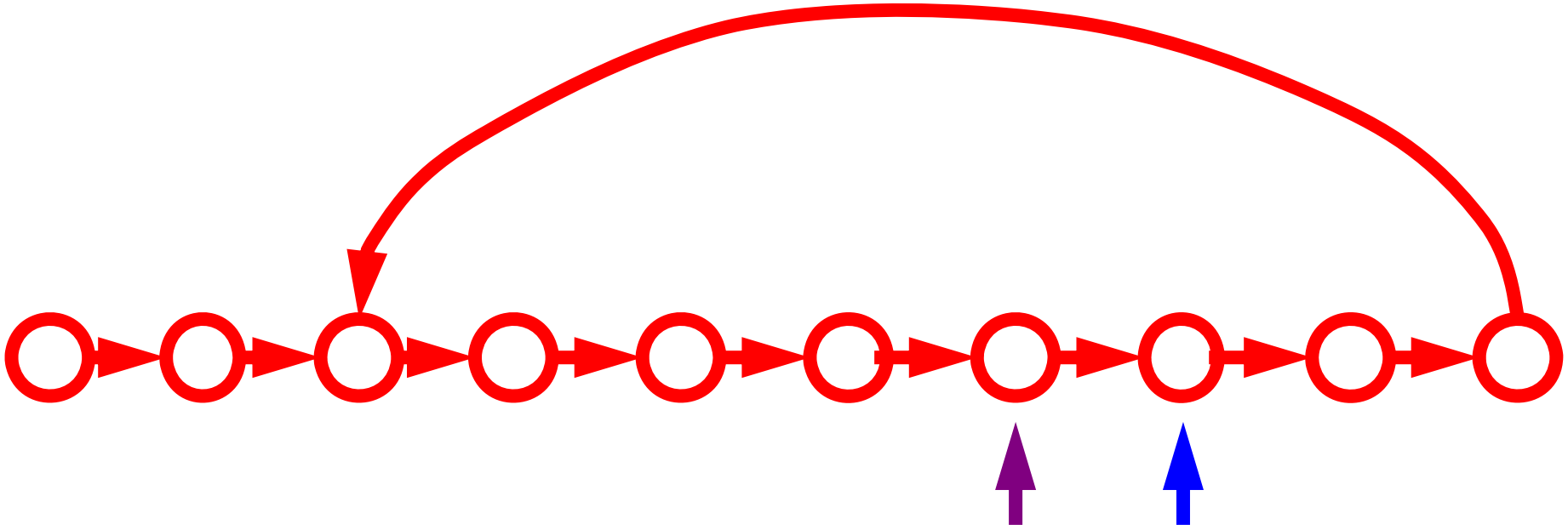
Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide



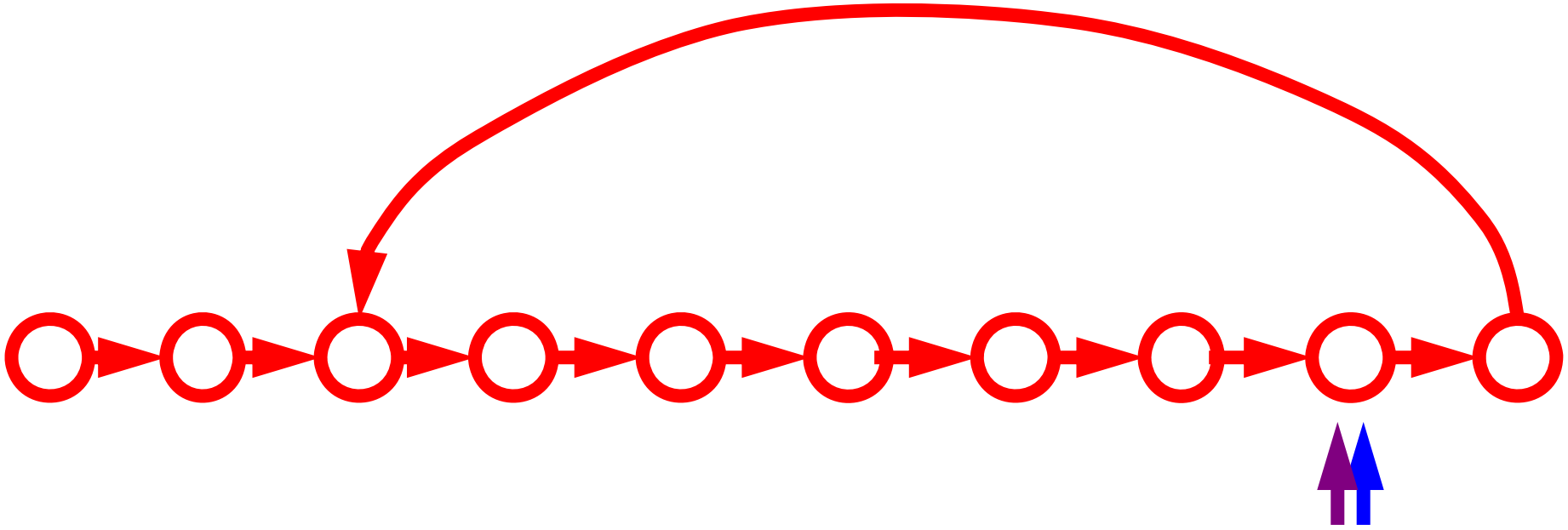
Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide

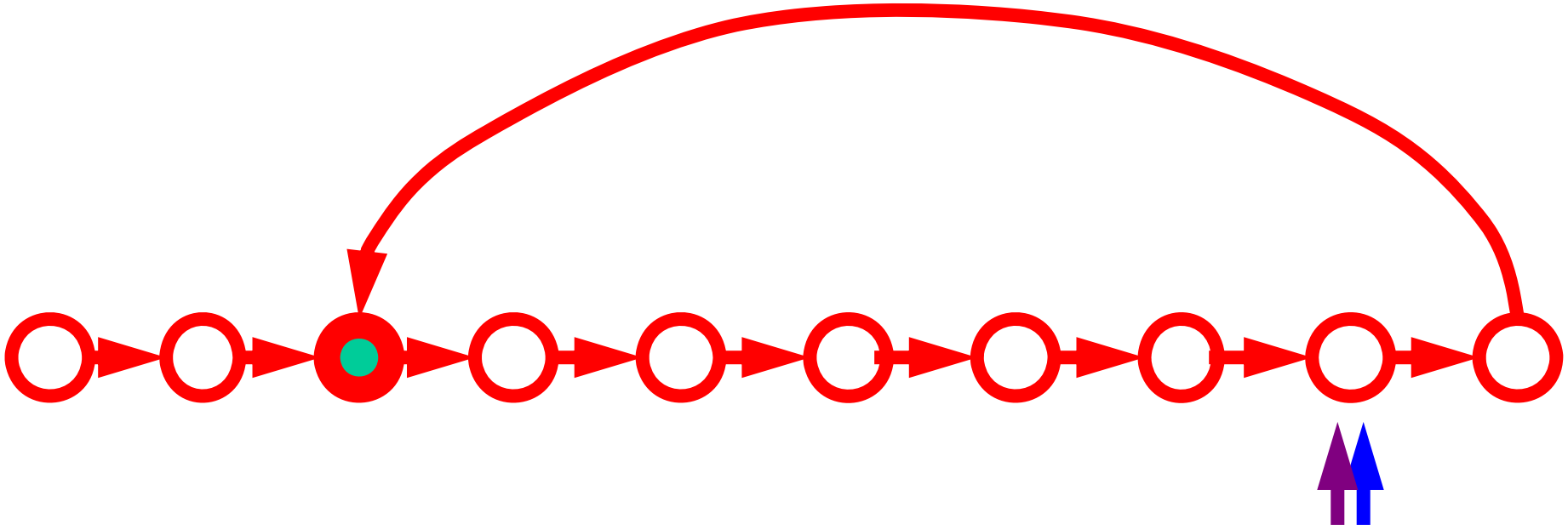


Floyd’s two finger algorithm:

- Keep two pointers
- Run one of them at normal speed, and the other at double speed, until they collide

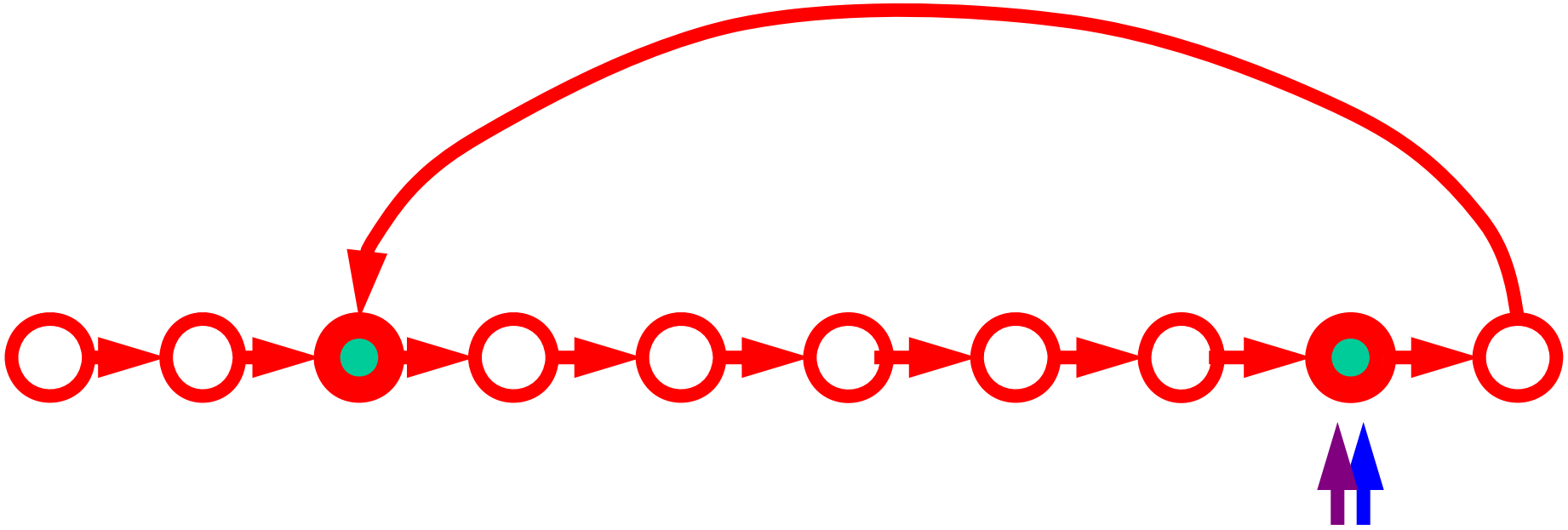


Can we use Floyd's algorithm to find the entry point into the cycle?



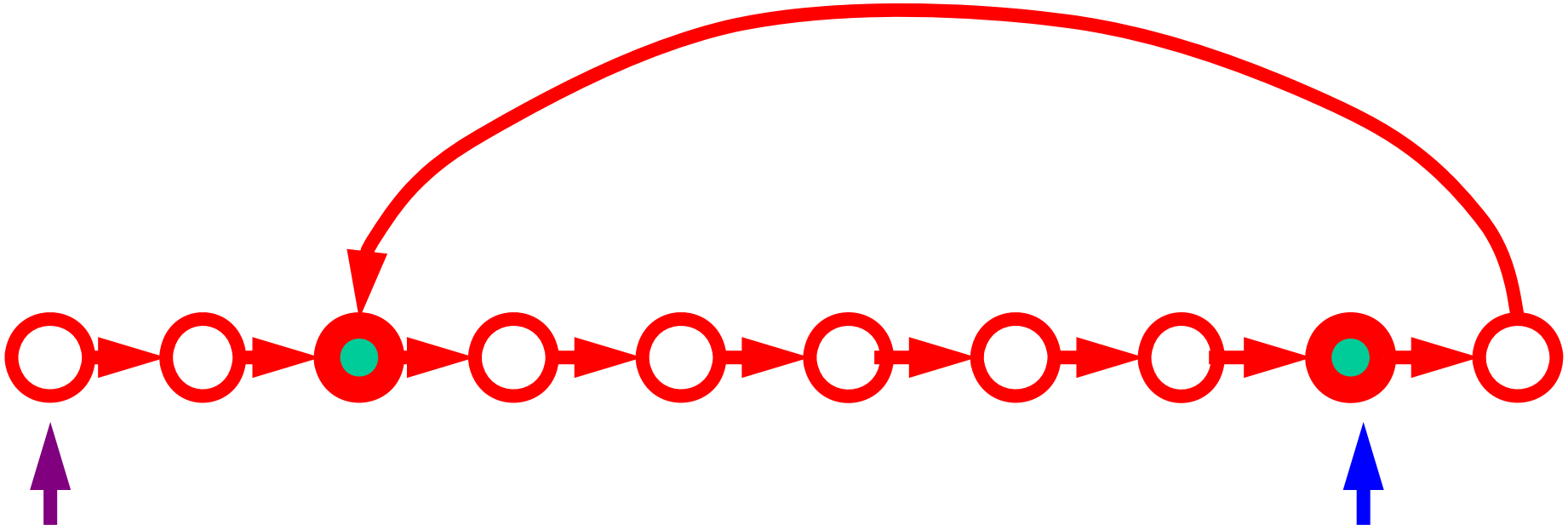
Can we use Floyd's algorithm to find the entry point into the cycle?

-First find the meeting point



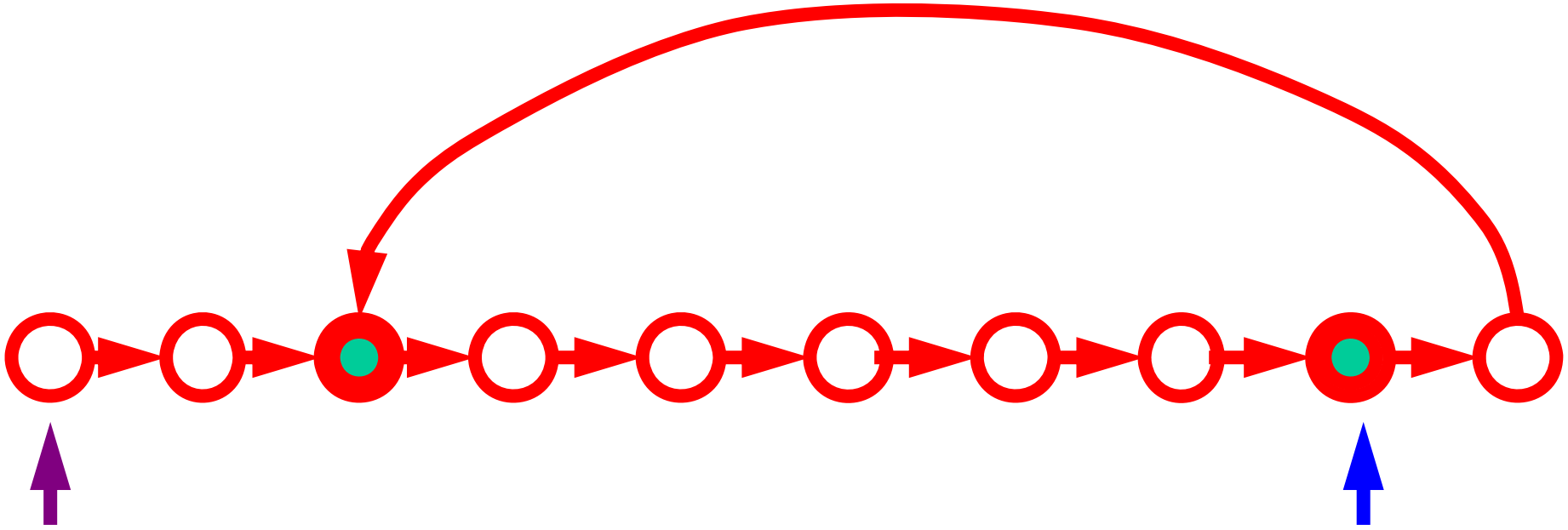
Can we use Floyd's algorithm to find the entry point into the cycle?

- first find the meeting point
- move one of the fingers back to the beginning



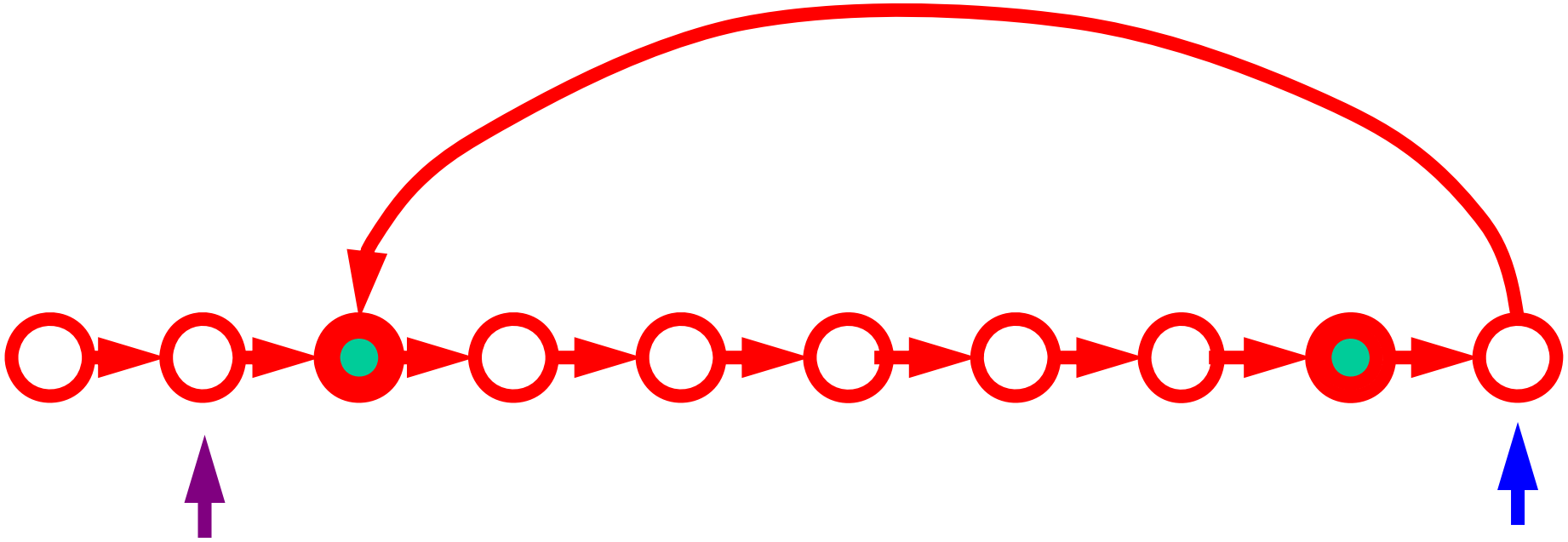
Can we use Floyd's algorithm to find the entry point into the cycle?

- first find the meeting point
- move one of the fingers back to the beginning
- move the two fingers at equal speed



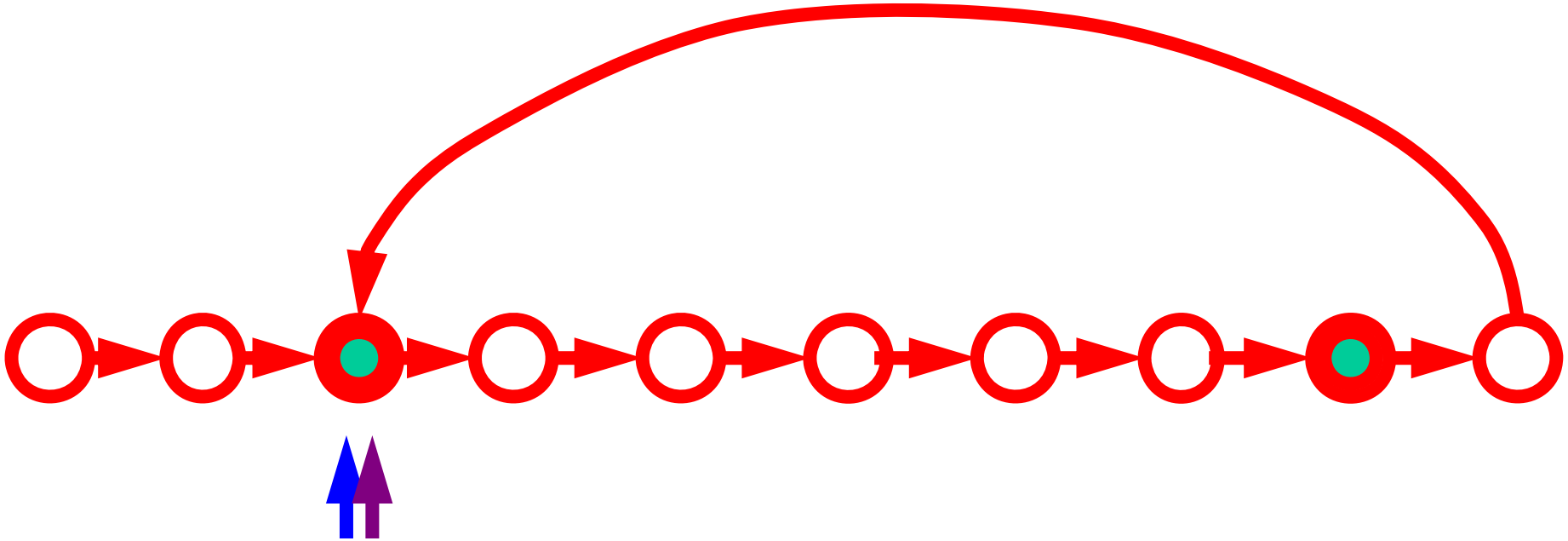
Can we use Floyd's algorithm to find the entry point into the cycle?

- first find the meeting point
- move one of the fingers back to the beginning
- move the two fingers at equal speed



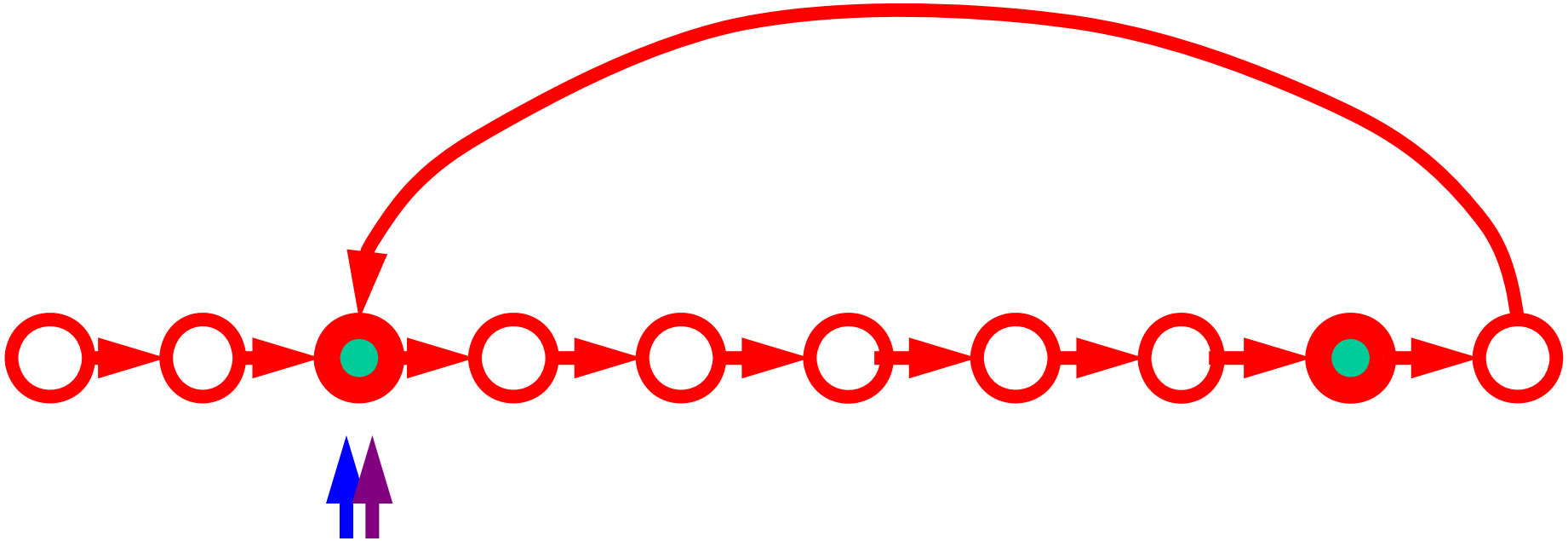
Can we use Floyd's algorithm to find the entry point into the cycle?

- first find the meeting point
- move one of the fingers back to the beginning
- move the two fingers at equal speed

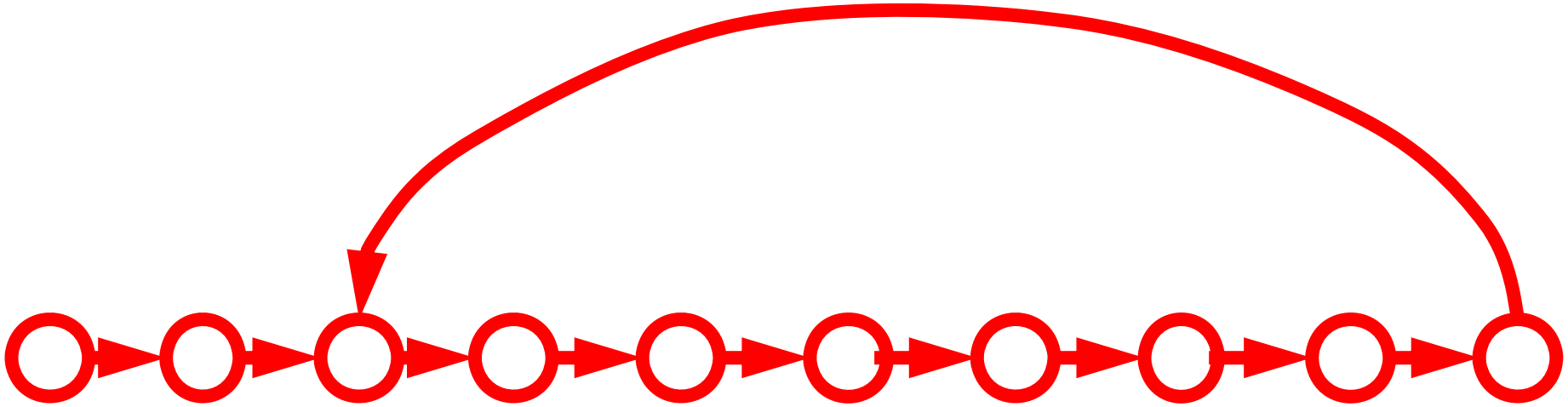


Why does it work?

(a good exercise for students)

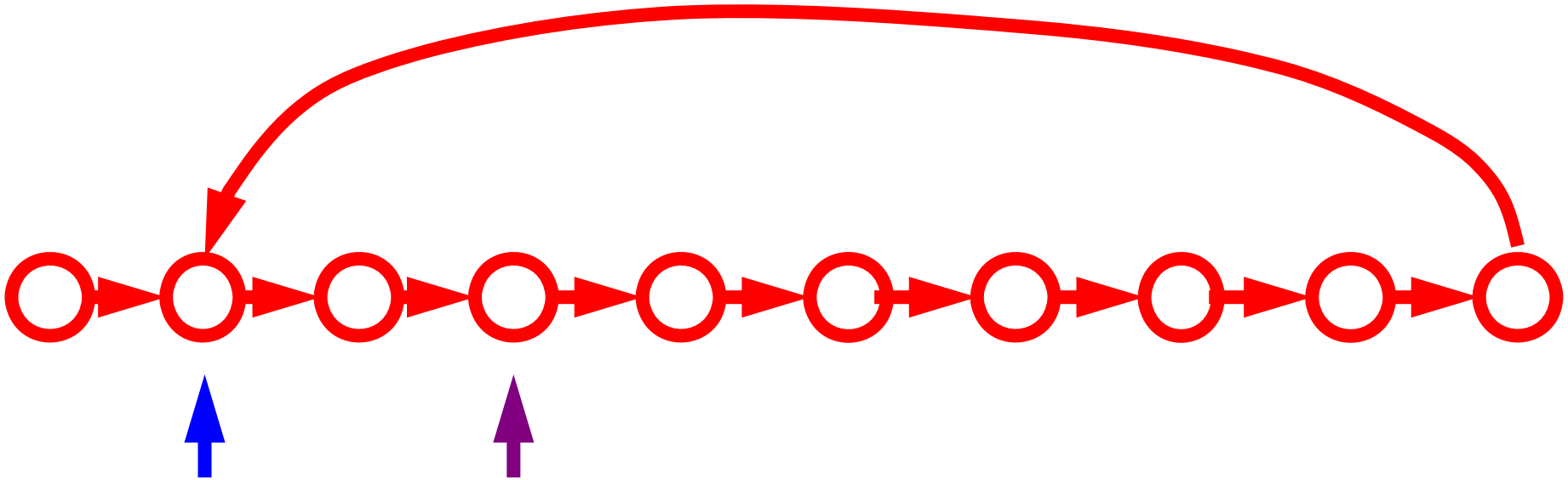


Is this the most efficient cycle detection algorithm?



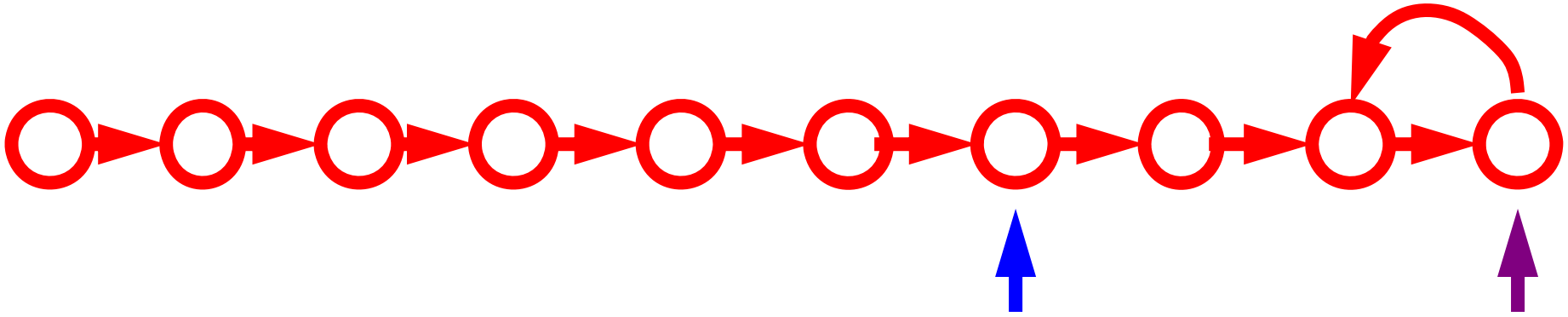
Is this the most efficient cycle detection algorithm?

- When the path has n vertices and the tail is short, Floyd's algorithm requires about 3n steps, and its extension requires up to 5n steps



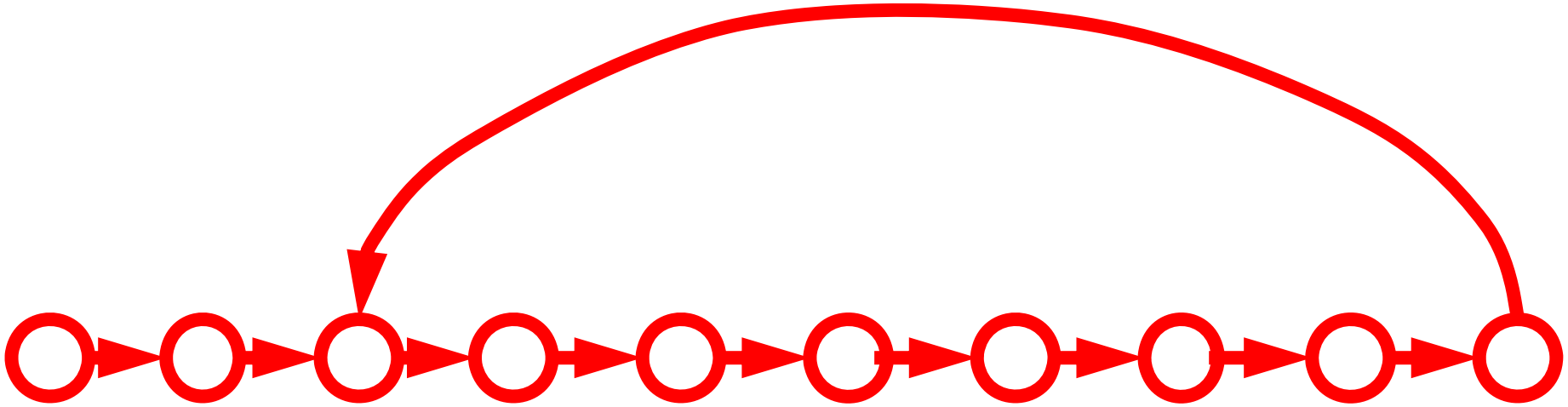
Is this the most efficient cycle detection algorithm?

- When the cycle is short, the fast finger can traverse it many times without noticing



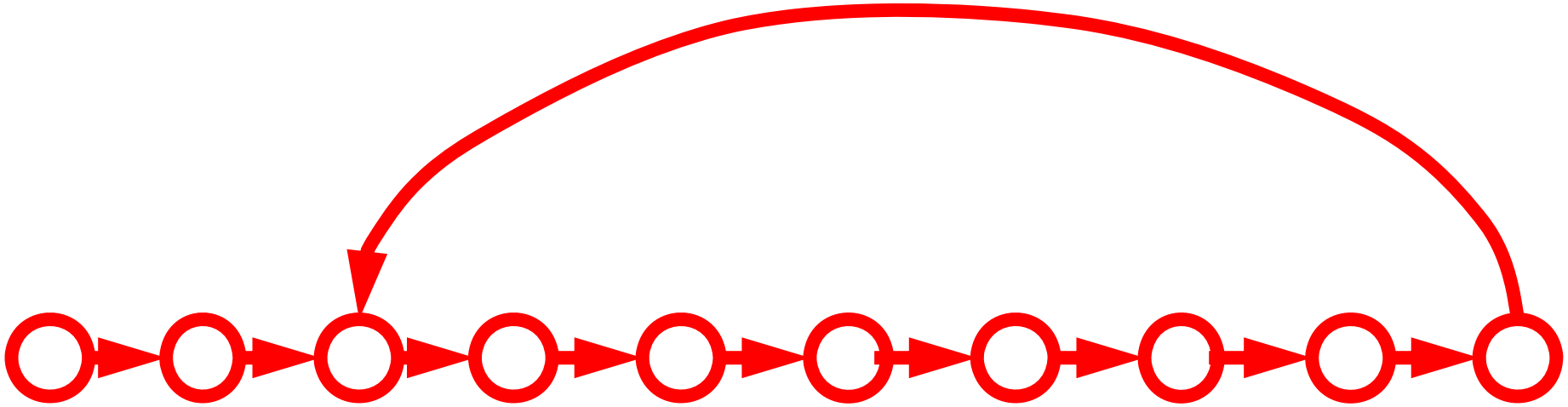
A very elegant solution:

Published by Gabriel Nivasch in 2004



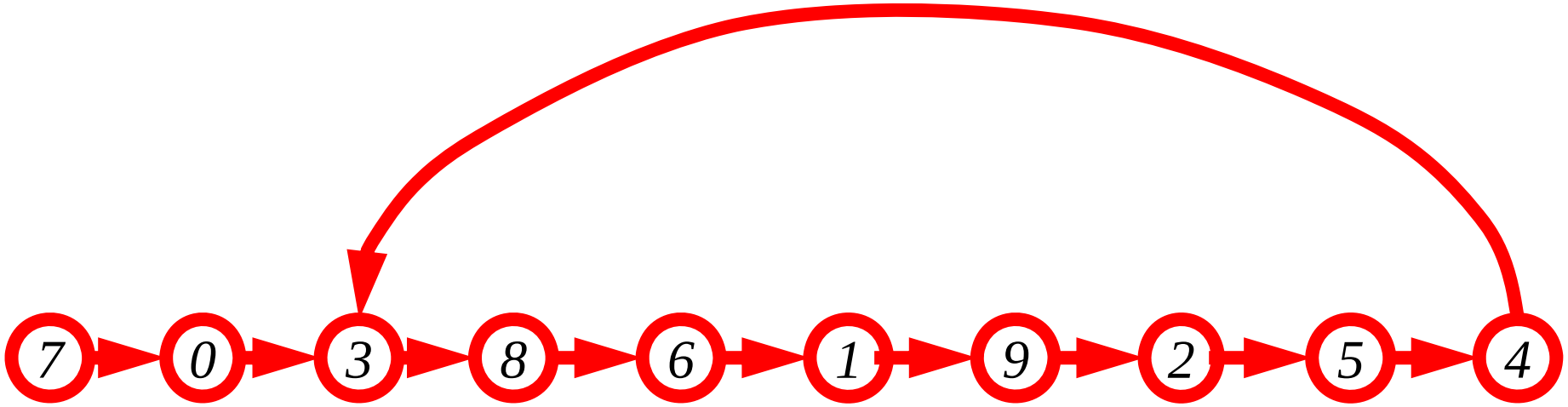
Properties of the Nivasch algorithm:

- Uses a single finger
- Uses negligible amount of memory
- Stops almost immediately after recycling
- Efficient for all possible lengths of cycle and tail
- Ideal for fast hardware implementations

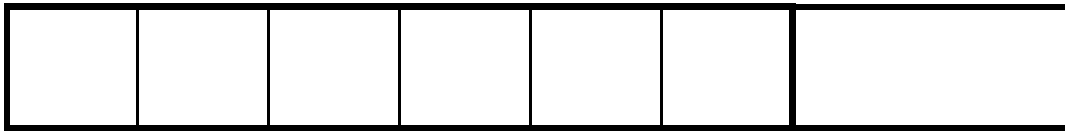


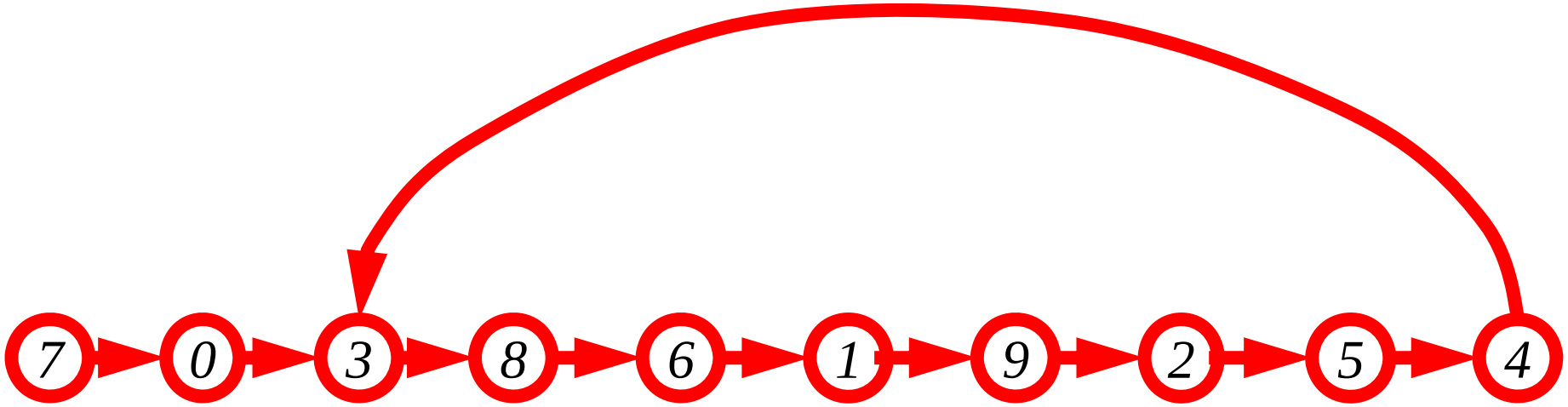
The basic idea of the algorithm:

- Maintain a stack of values, which is initially empty
- Insert each new value into the top of the stack
- Force the values in the stack to be monotonically increasing

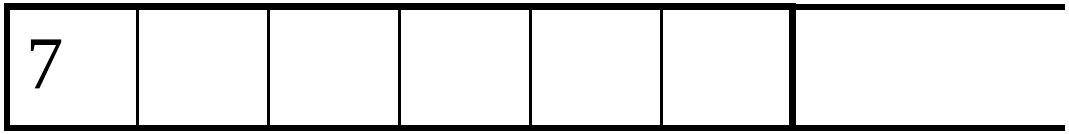


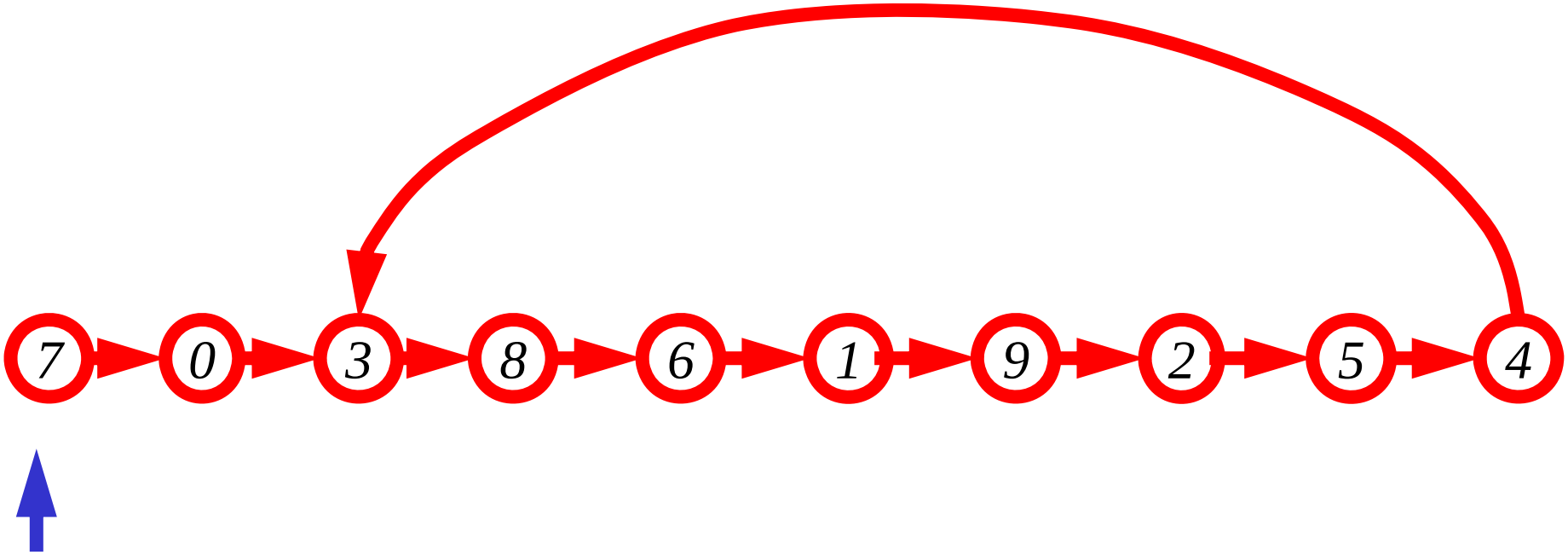
The Stack Algorithm:



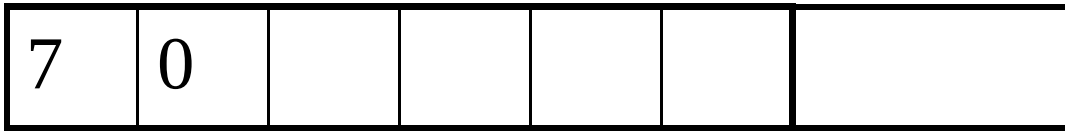


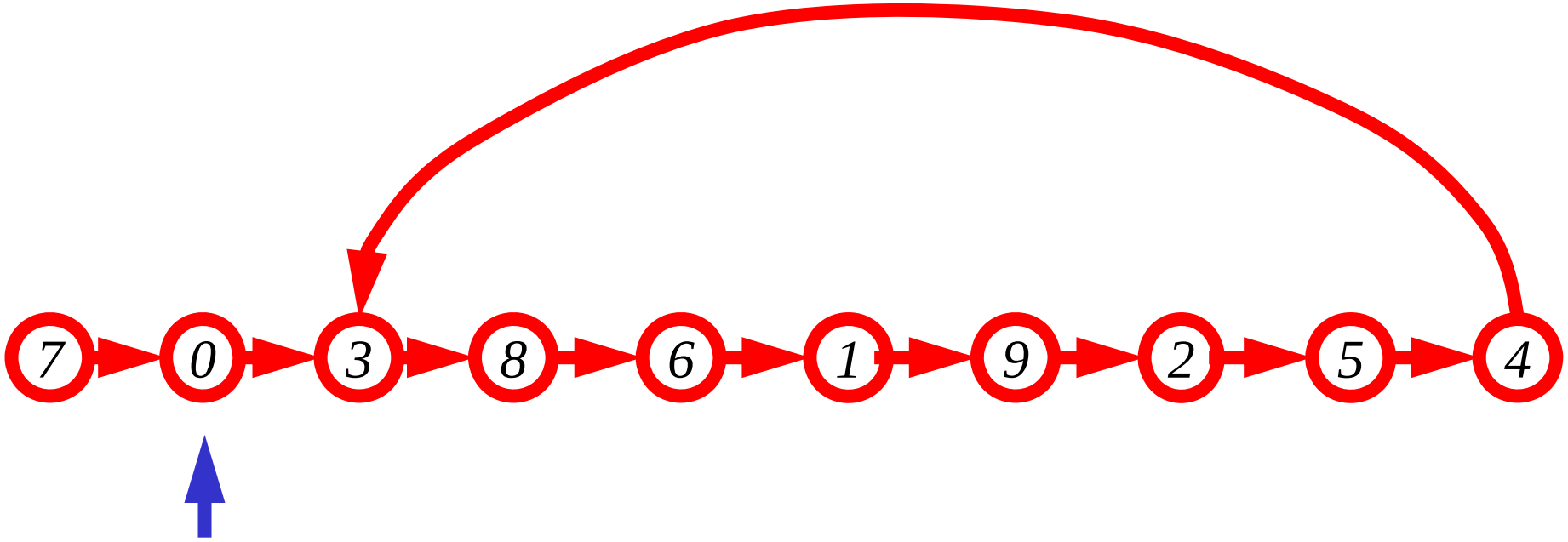
The Stack Algorithm:



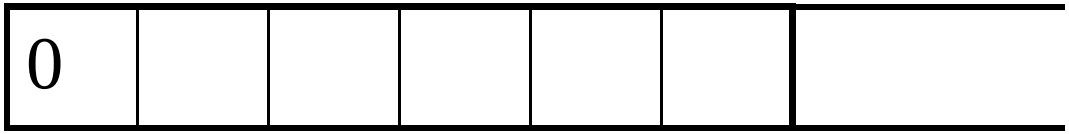


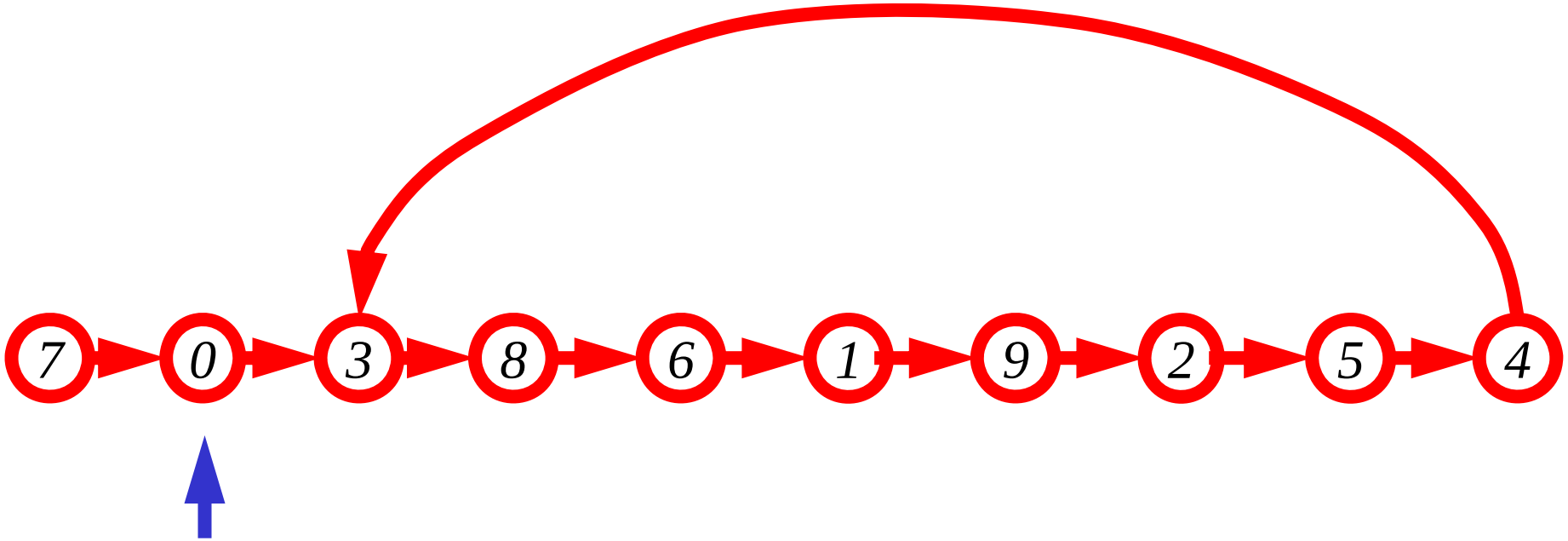
The Stack Algorithm:



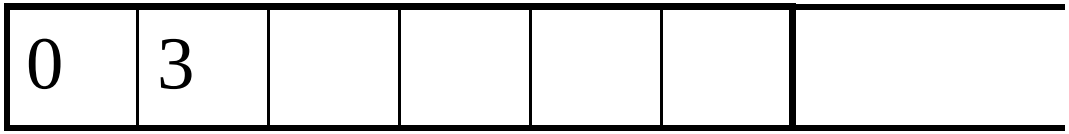


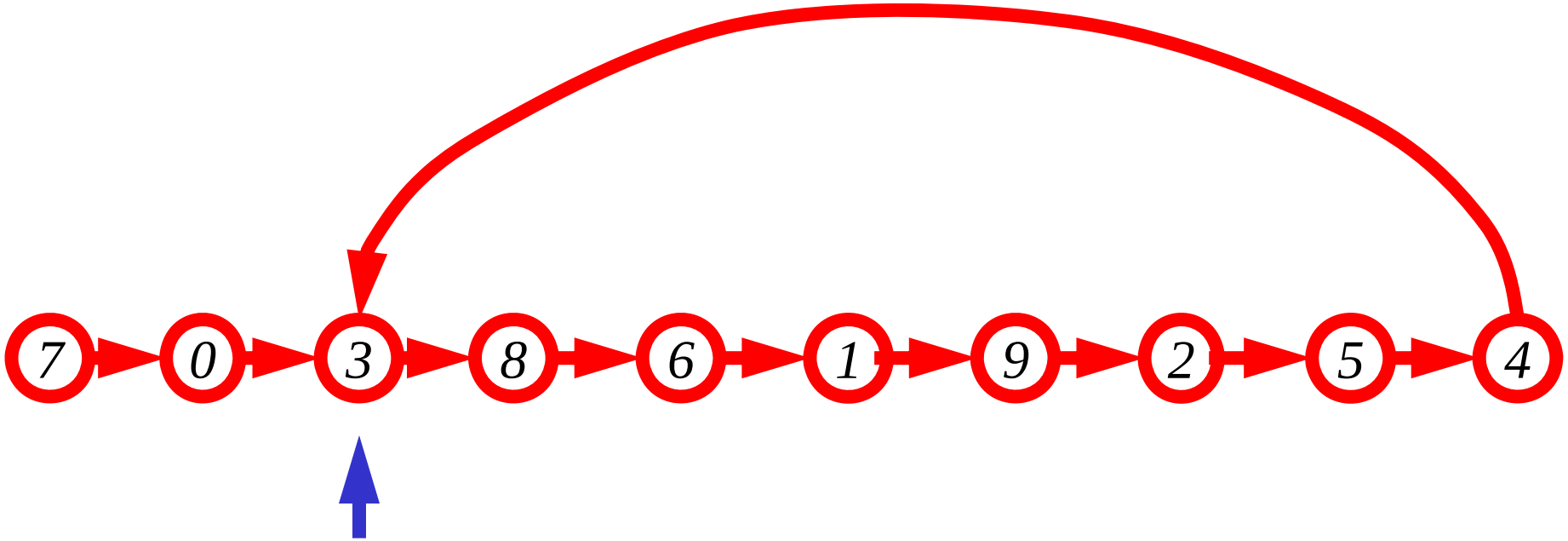
The Stack Algorithm:





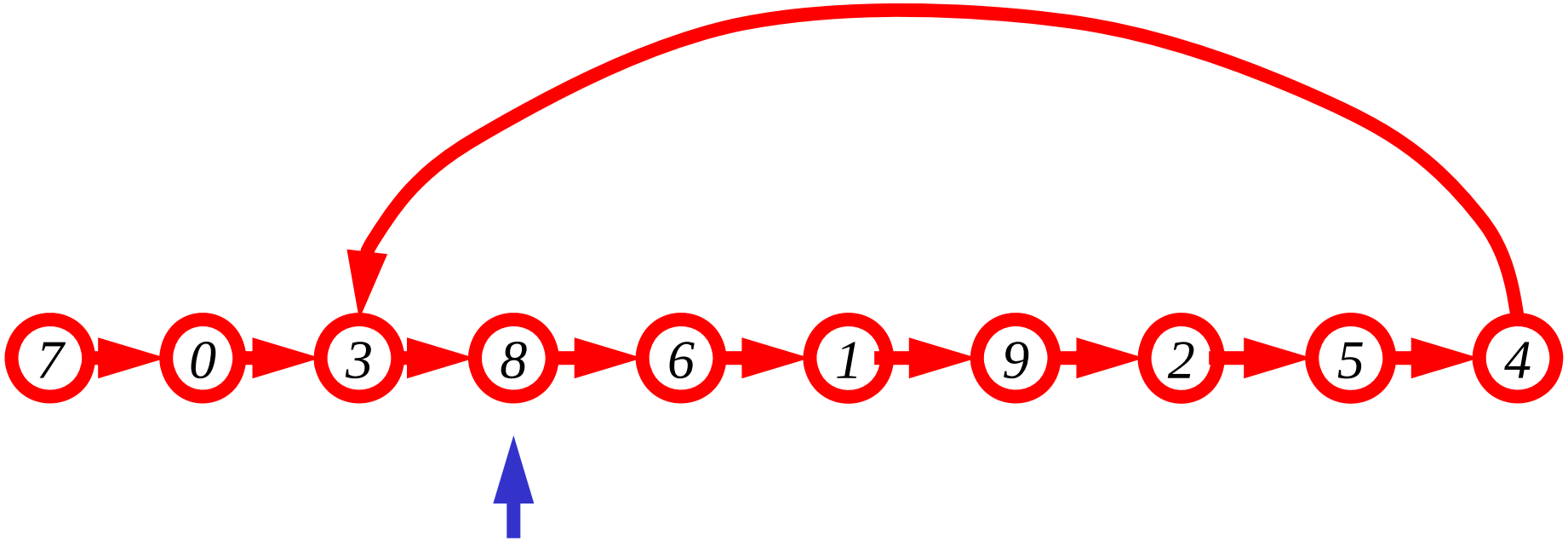
The Stack Algorithm:





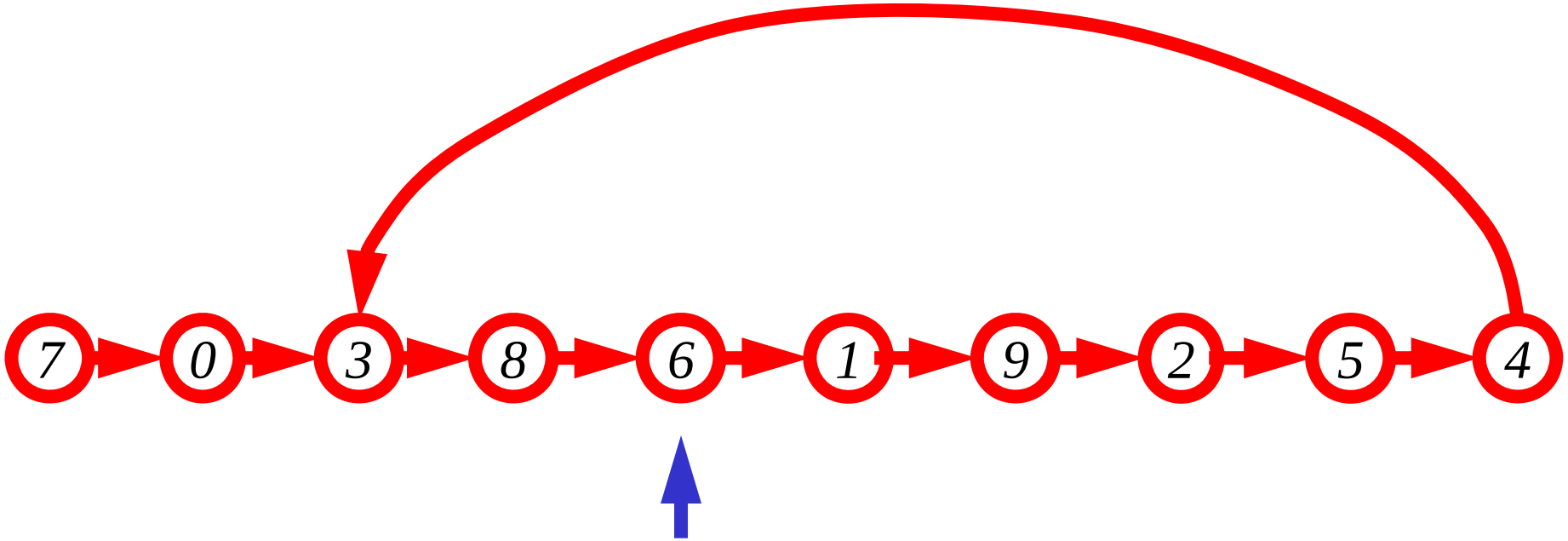
The Stack Algorithm:

0	3	8				



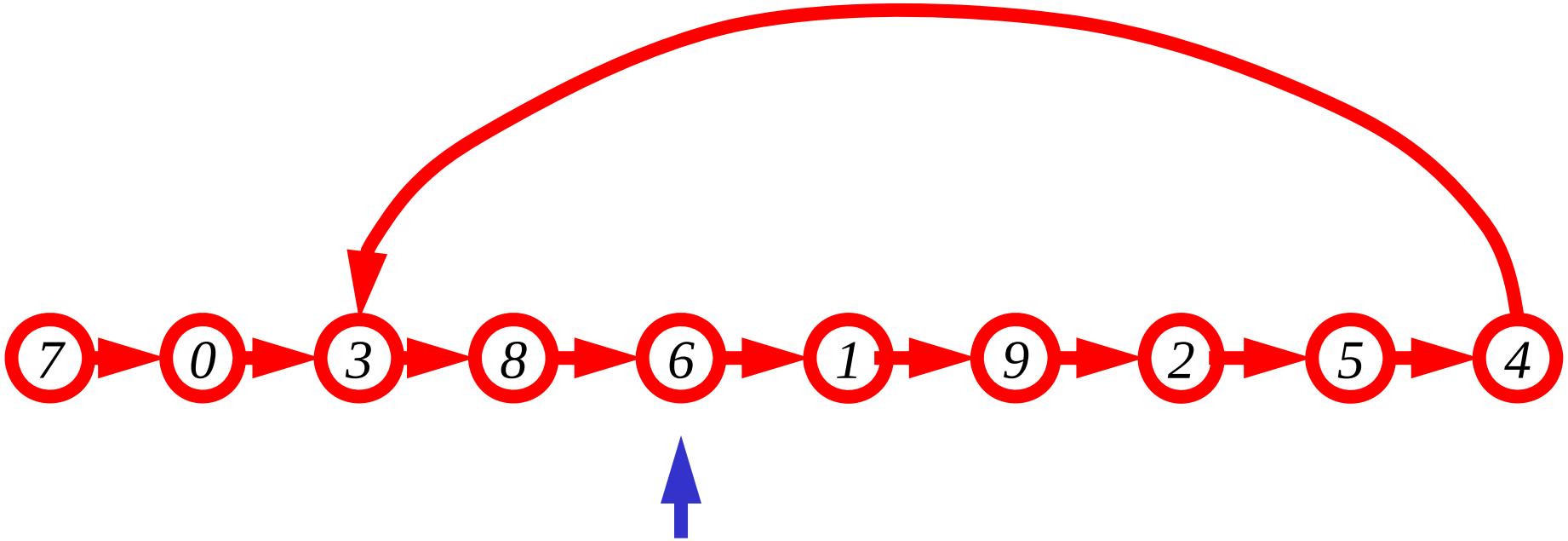
The Stack Algorithm:

0	3	8	6			



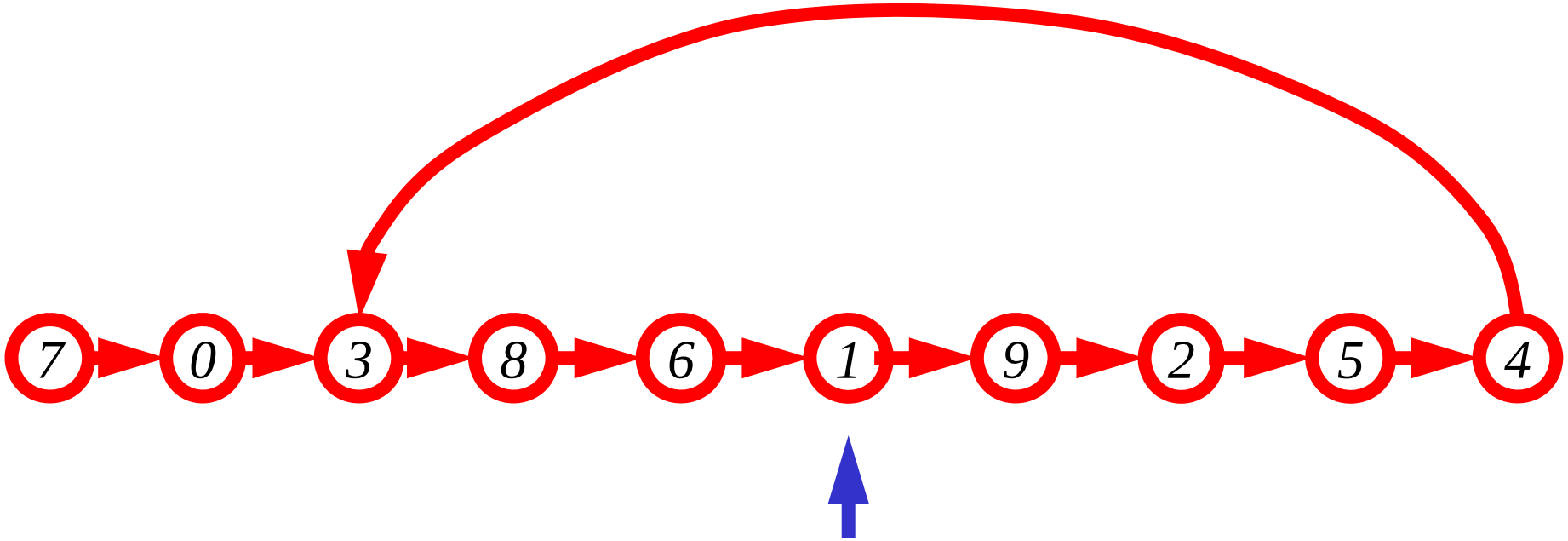
The Stack Algorithm:

0	3	6				



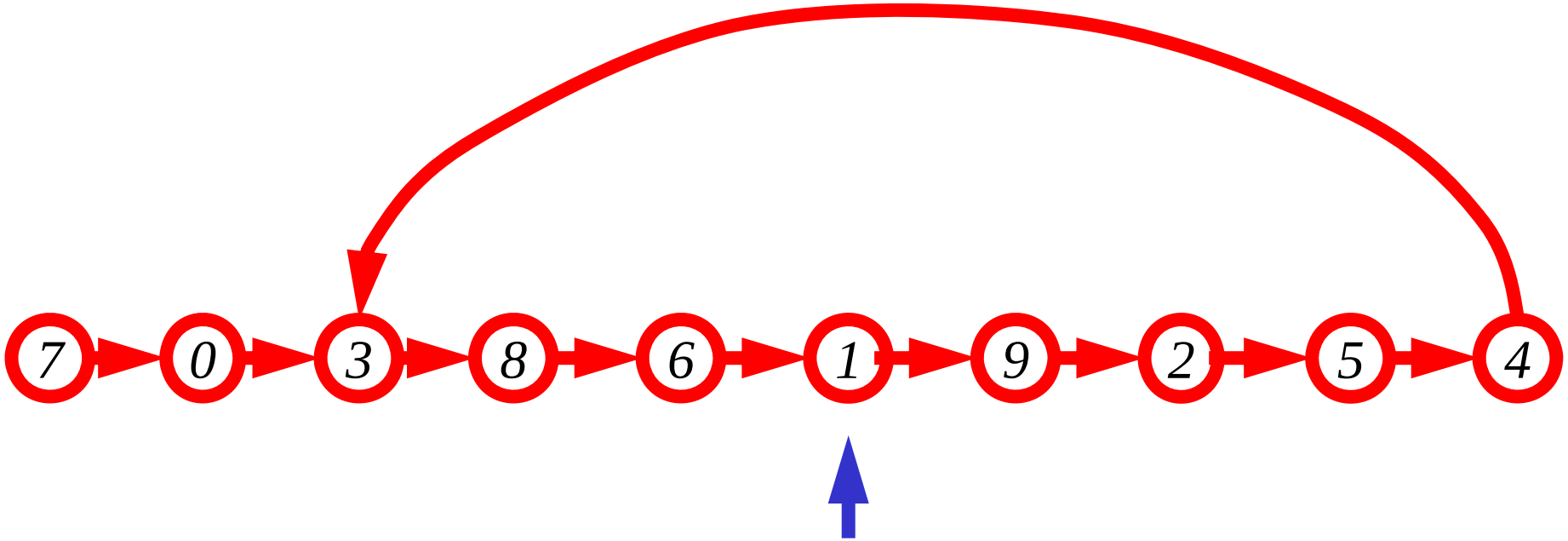
The Stack Algorithm:

0	3	6	1			

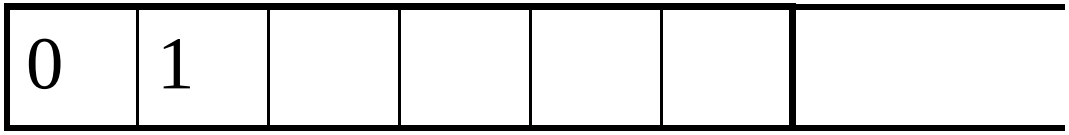


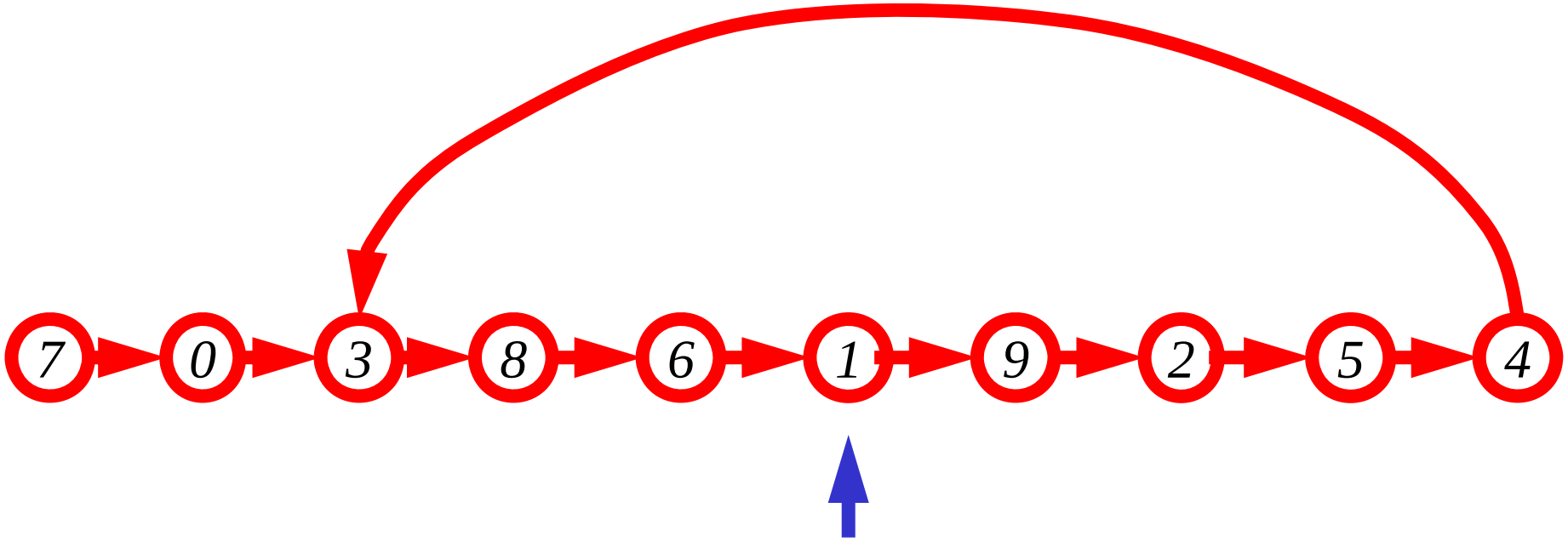
The Stack Algorithm:

0	3	1				



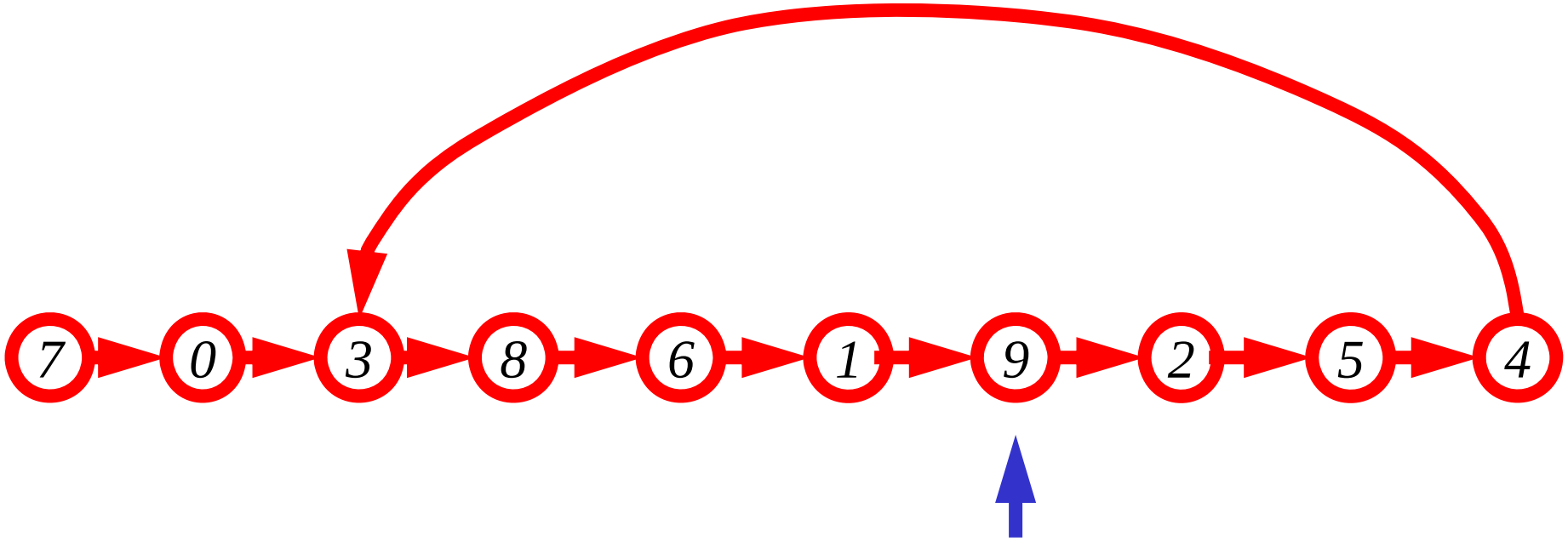
The Stack Algorithm:





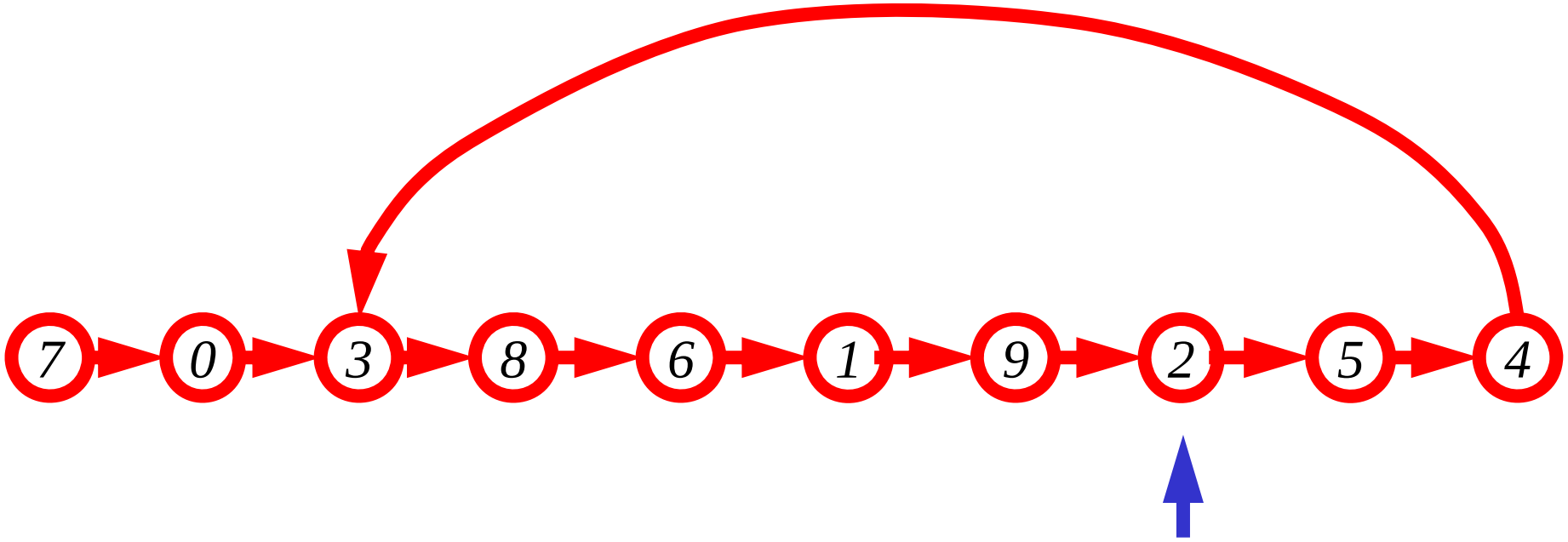
The Stack Algorithm:

0	1	9				

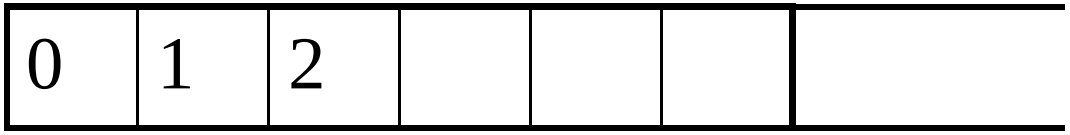


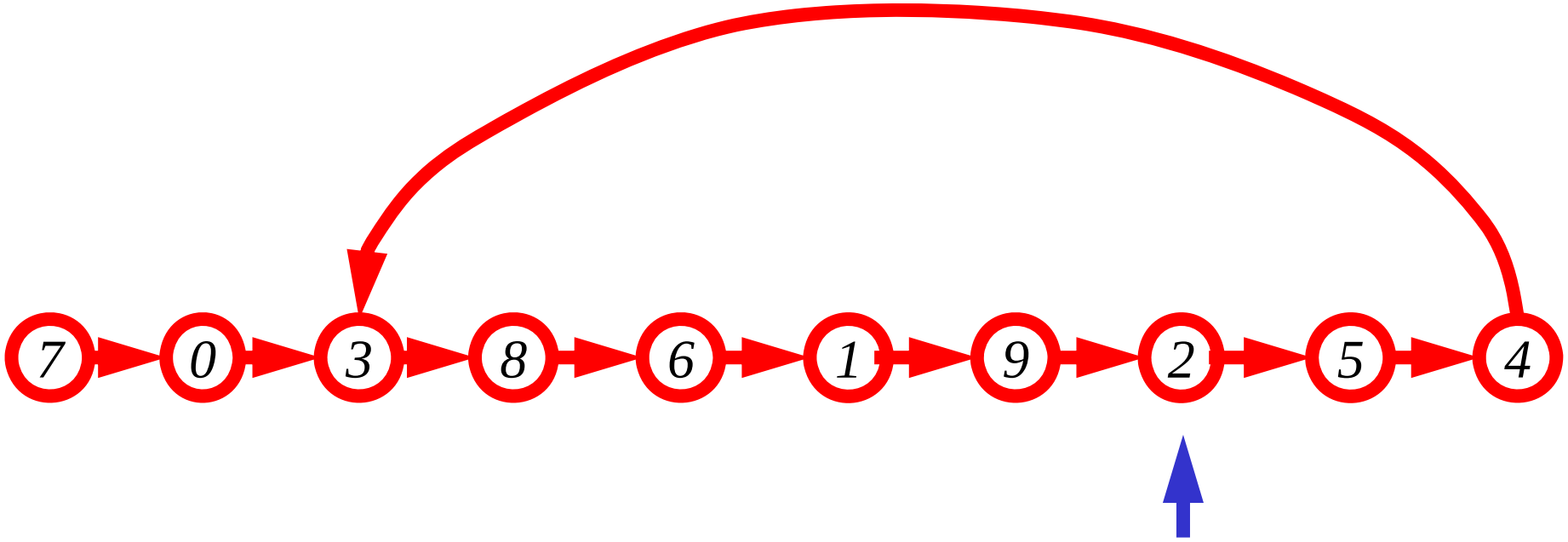
The Stack Algorithm:

0	1	9	2			



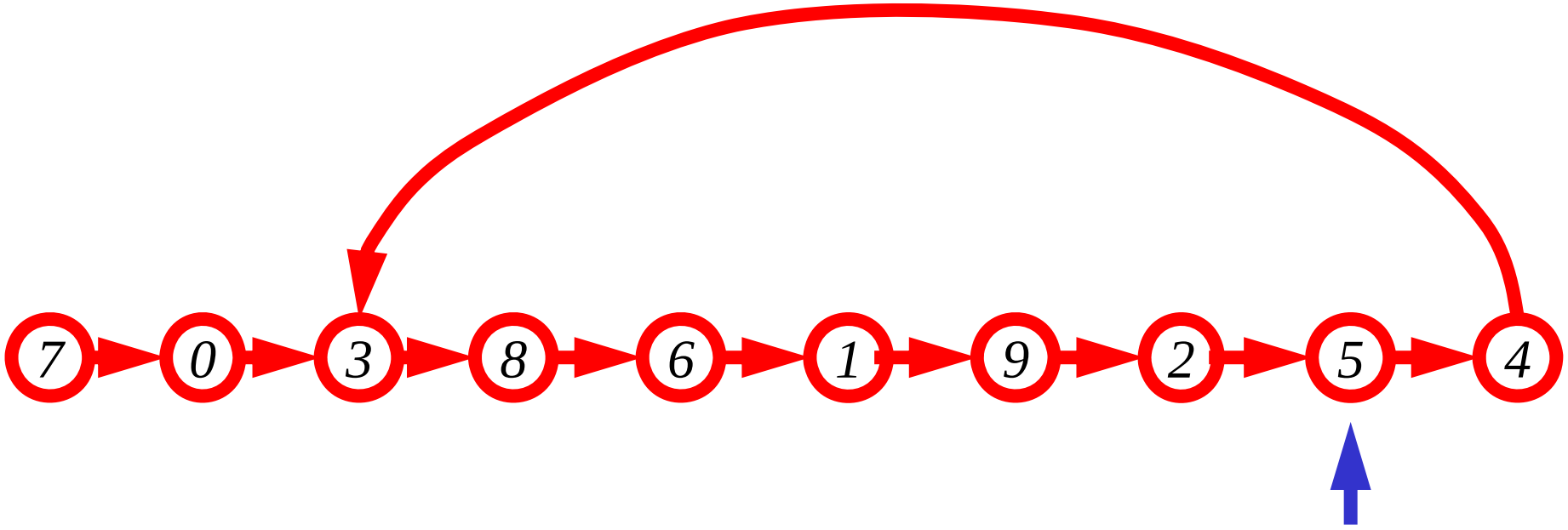
The Stack Algorithm:





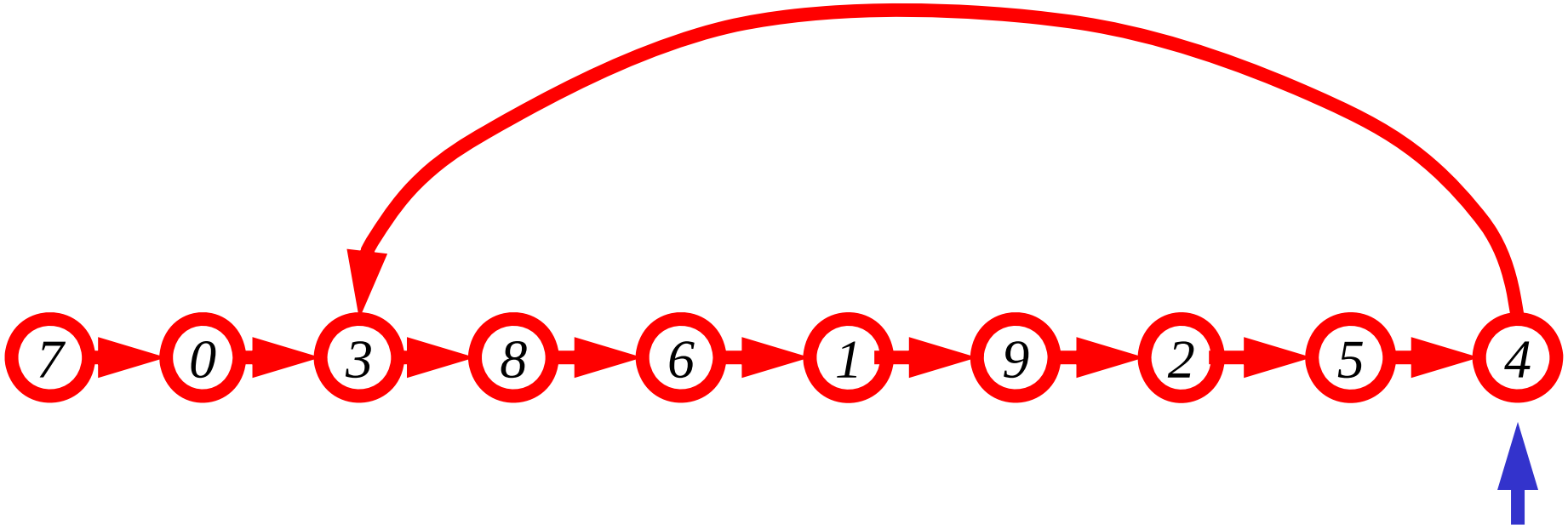
The Stack Algorithm:

0	1	2	5			



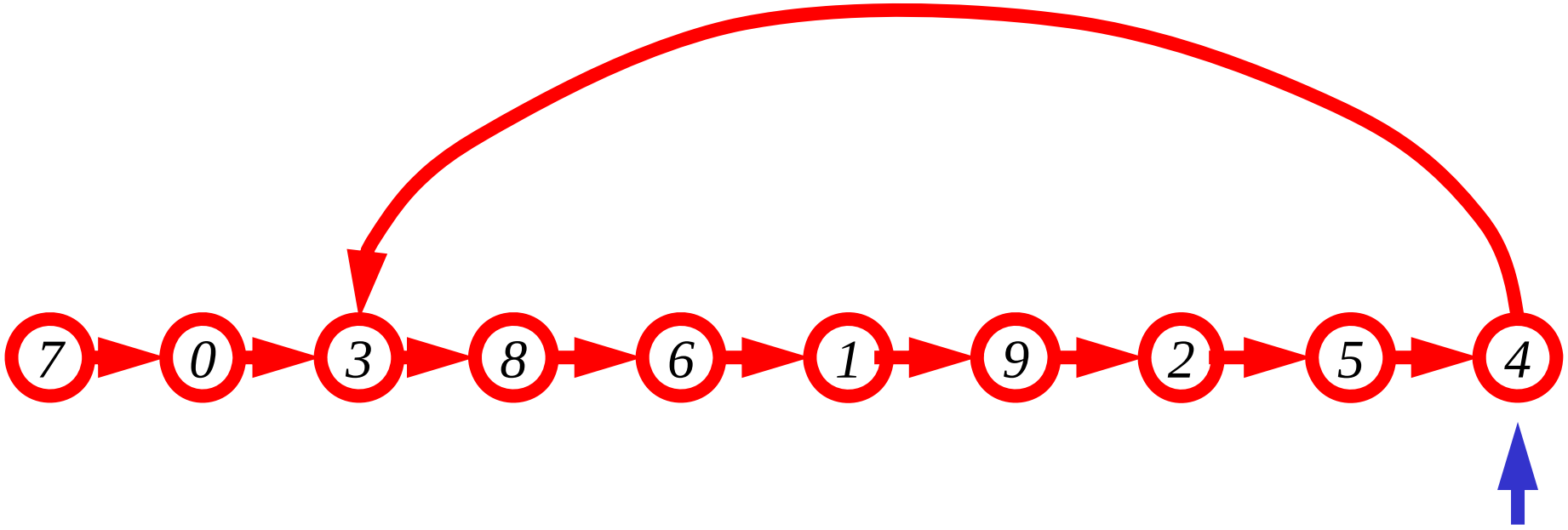
The Stack Algorithm:

0	1	2	5	4		



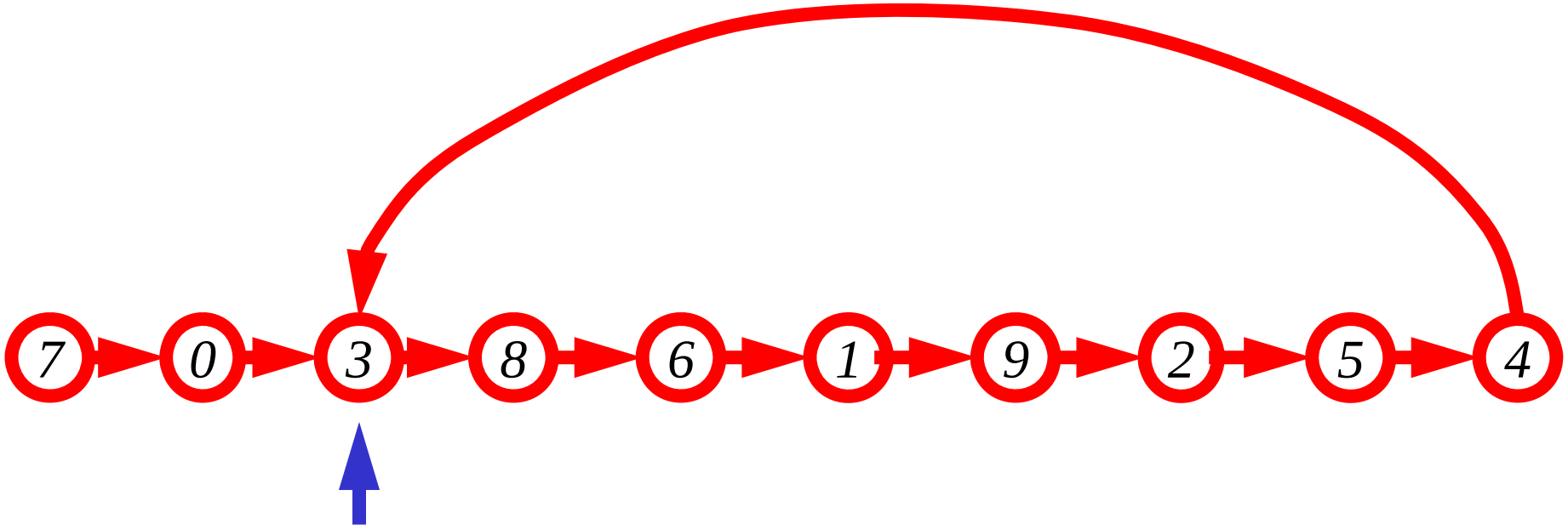
The Stack Algorithm:

0	1	2	4			

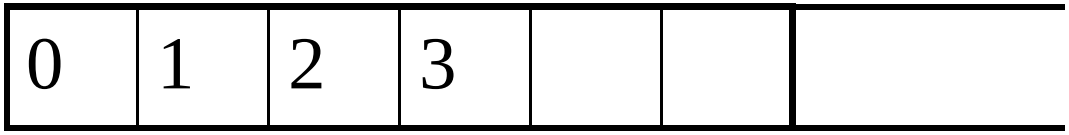


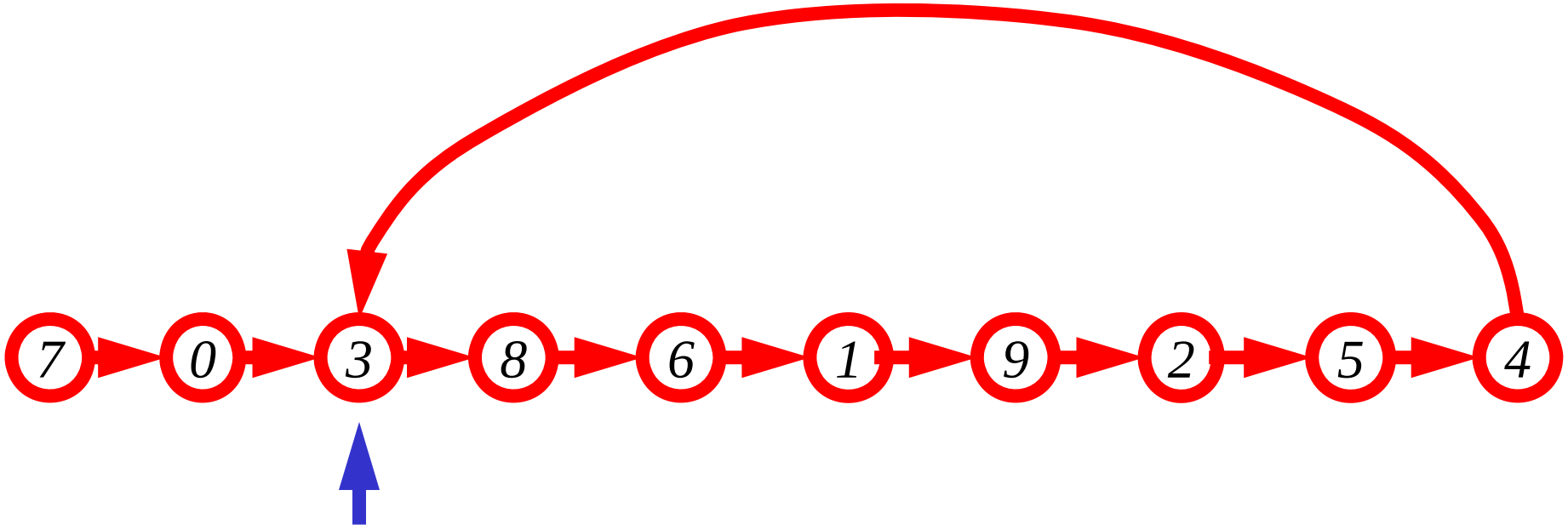
The Stack Algorithm:

0	1	2	4	3		



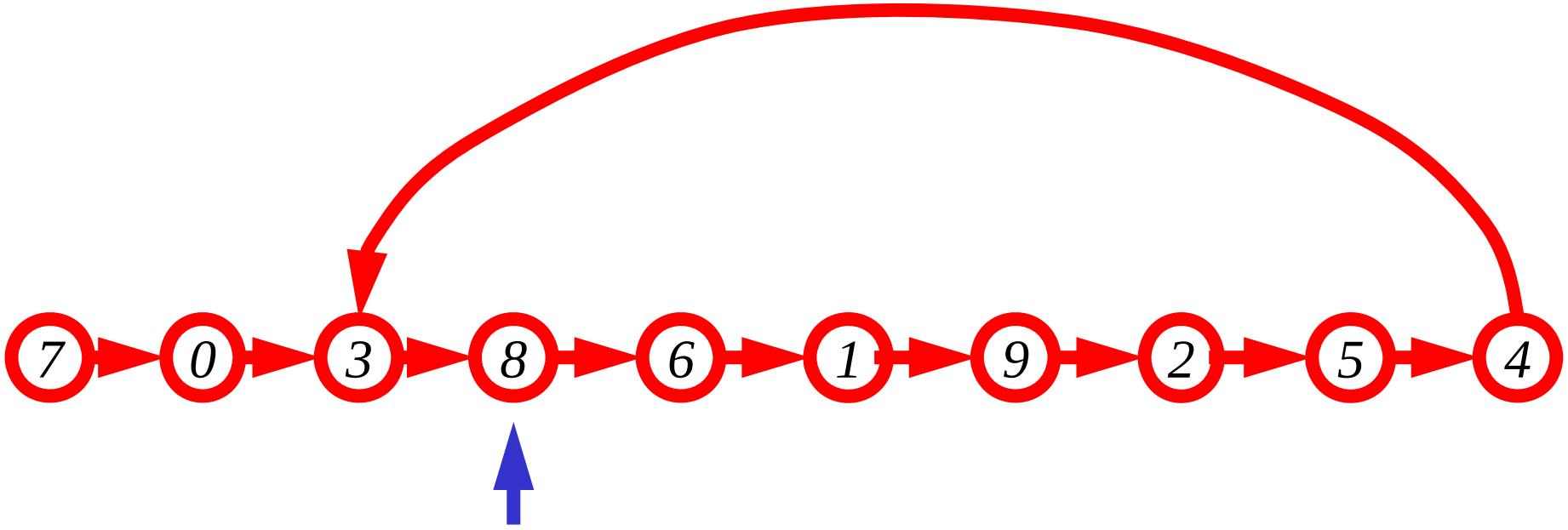
The Stack Algorithm:





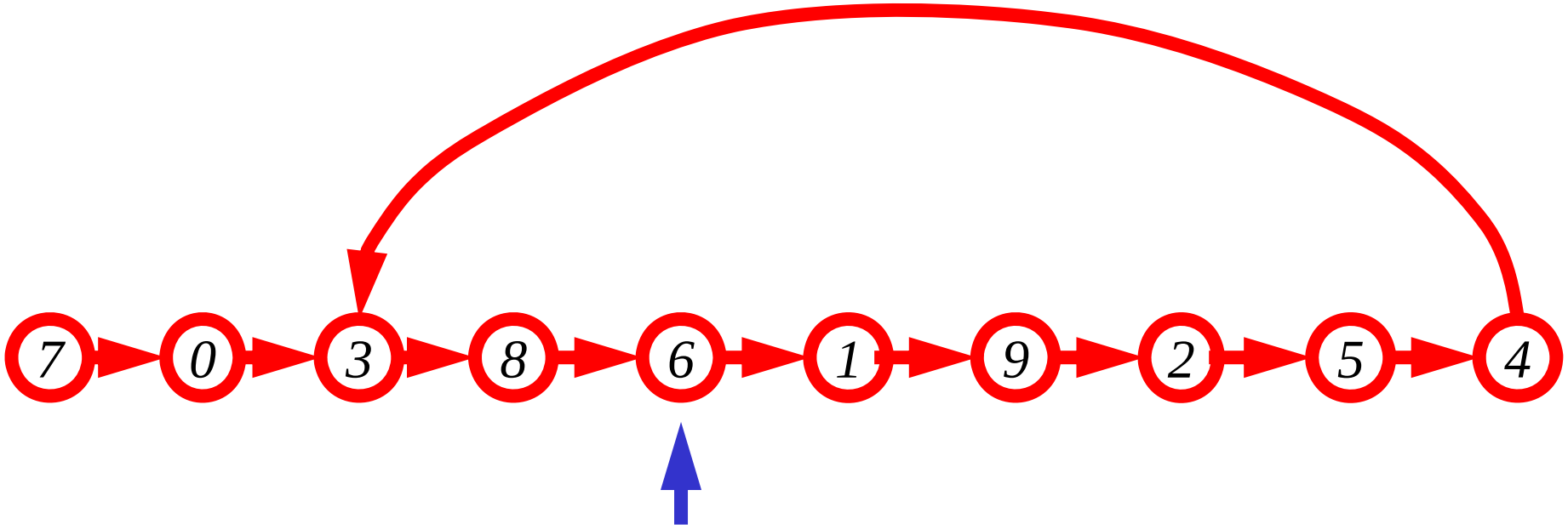
The Stack Algorithm:

0	1	2	3	8		



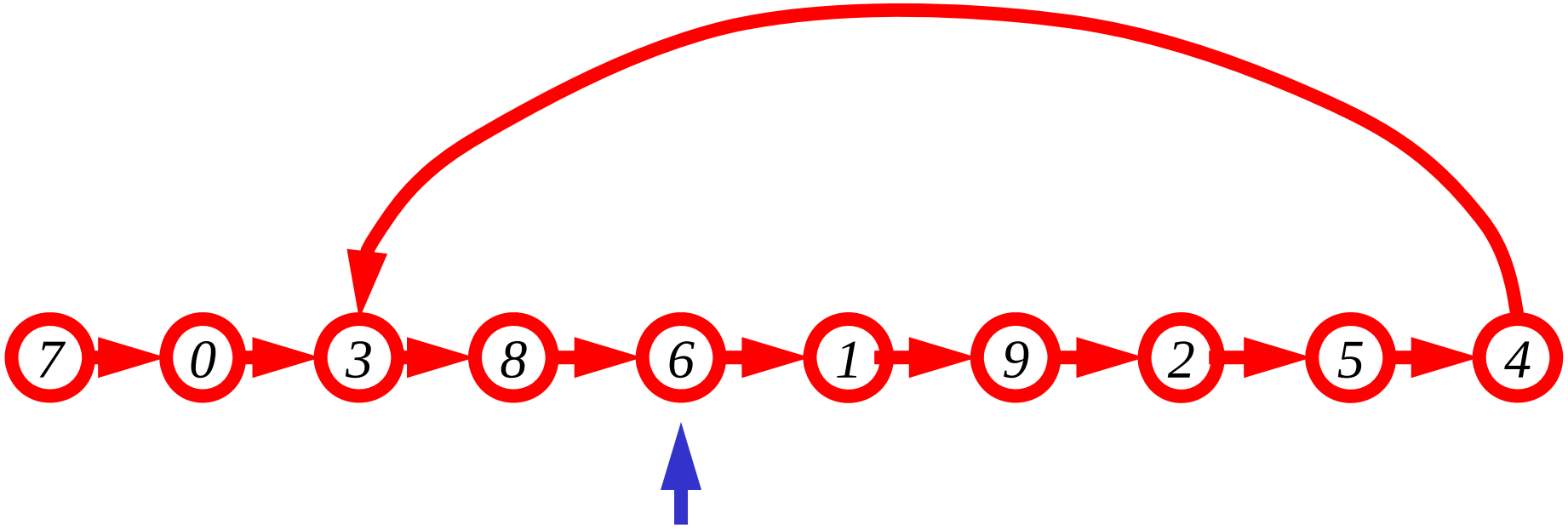
The Stack Algorithm:

0	1	2	3	8	6	



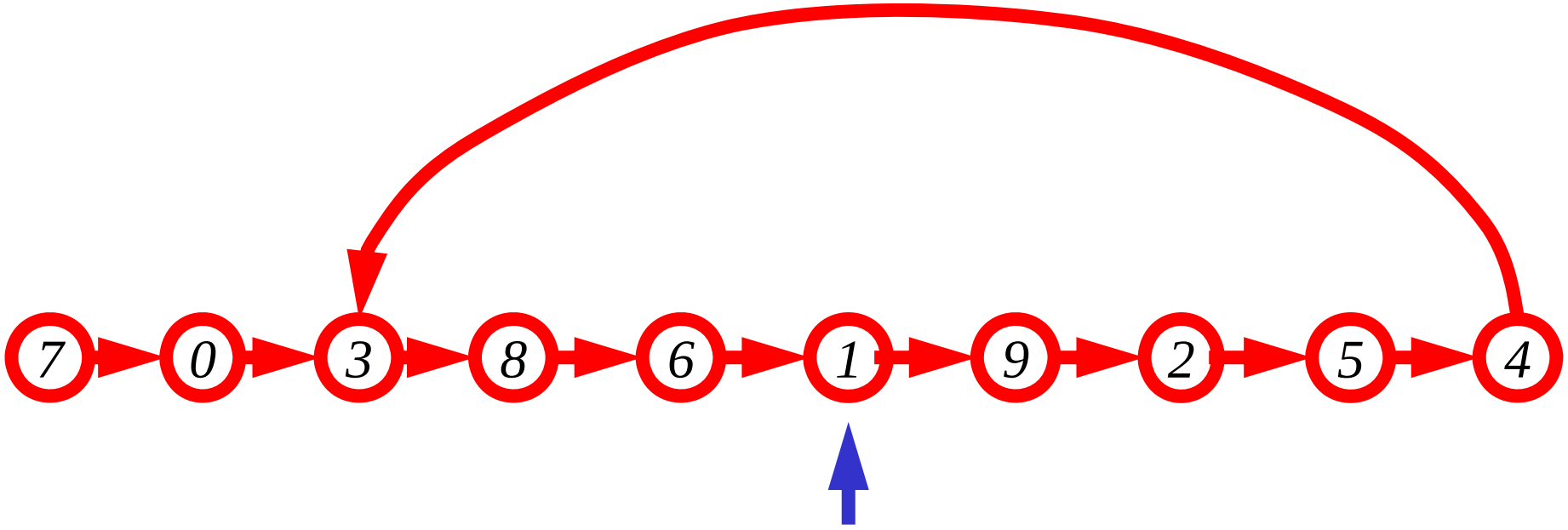
The Stack Algorithm:

0	1	2	3	6		



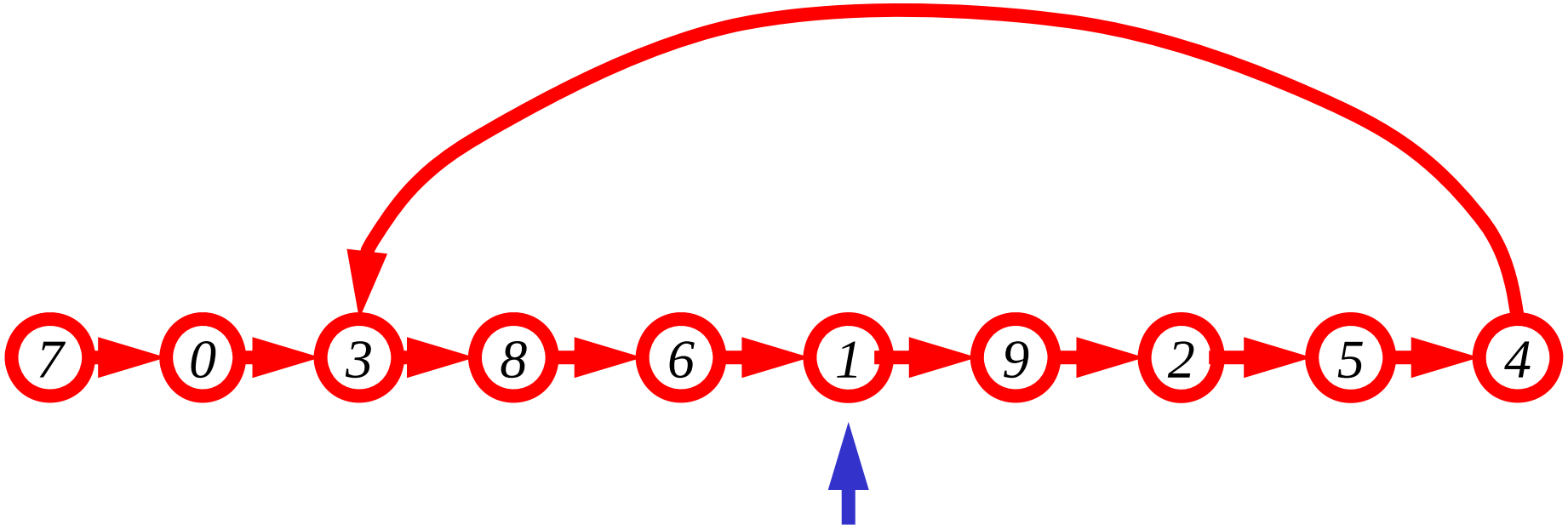
The Stack Algorithm:

0	1	2	3	6	1	



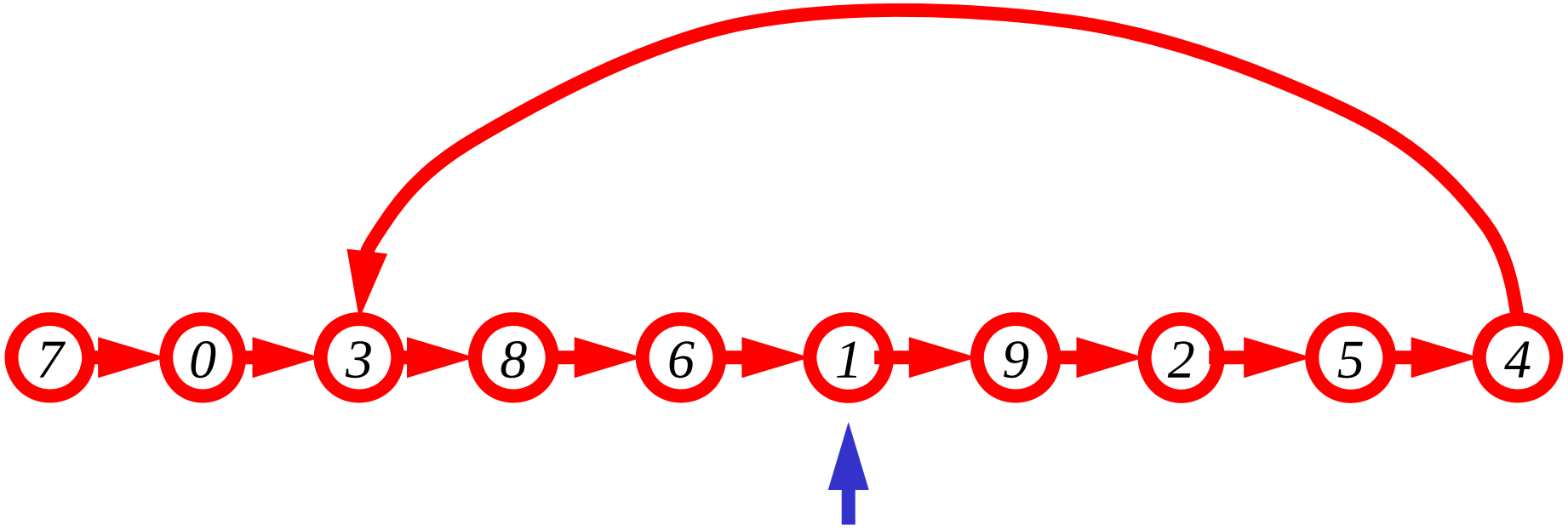
The Stack Algorithm:

0	1	2	3	1		



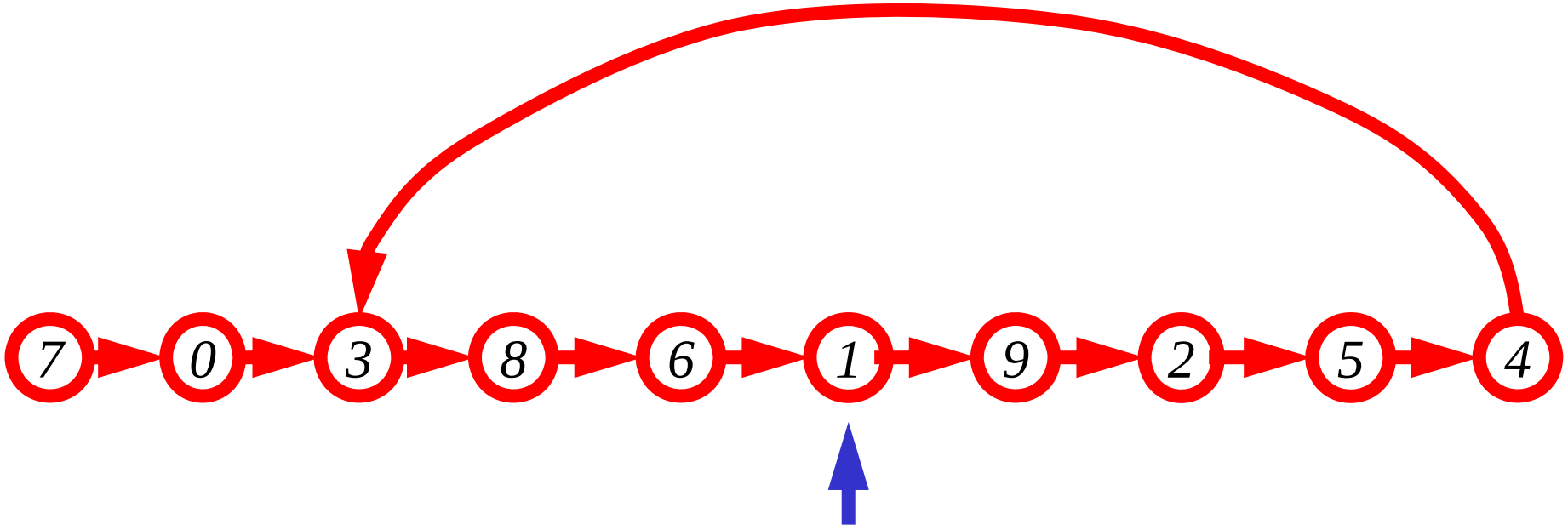
The Stack Algorithm:

0	1	2	1			



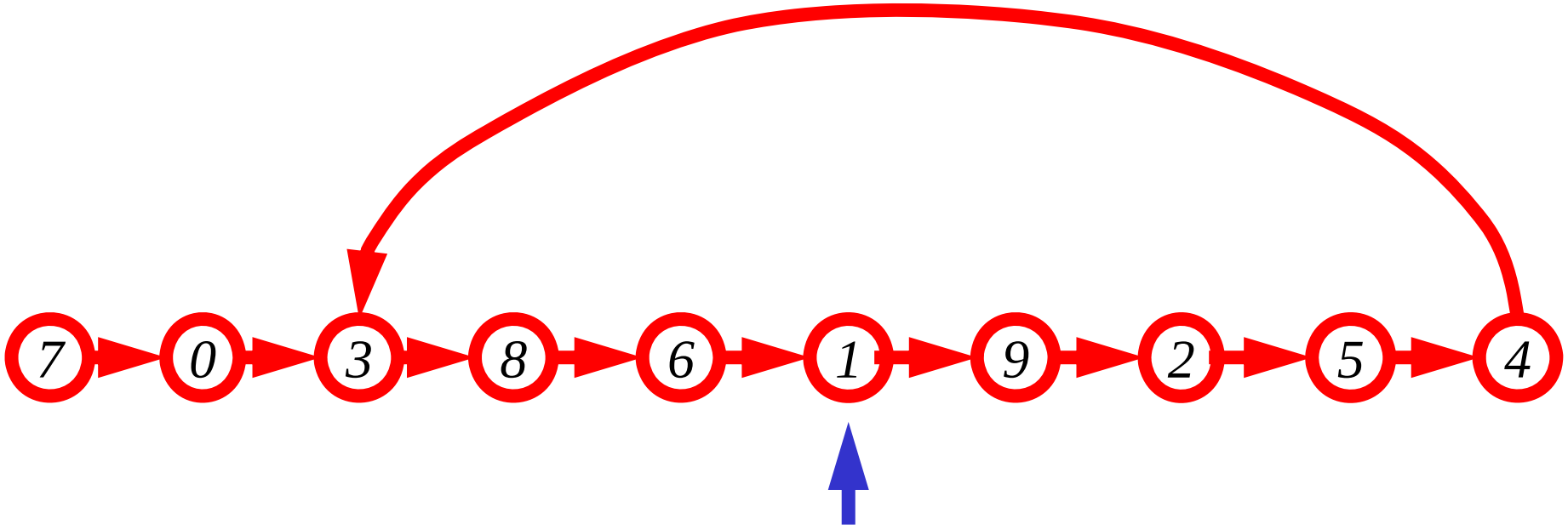
The Stack Algorithm:

0	1	1				

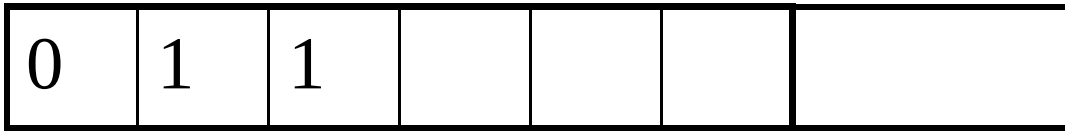


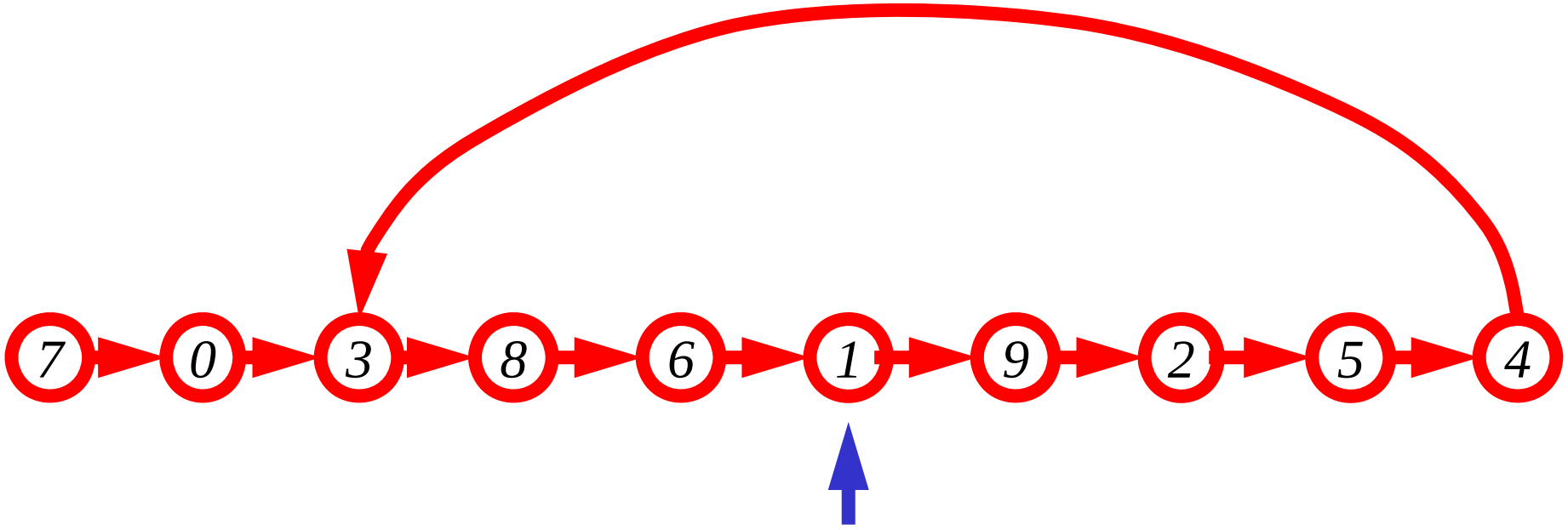
Stop when two identical values appear at the top of the stack

0	1	1				



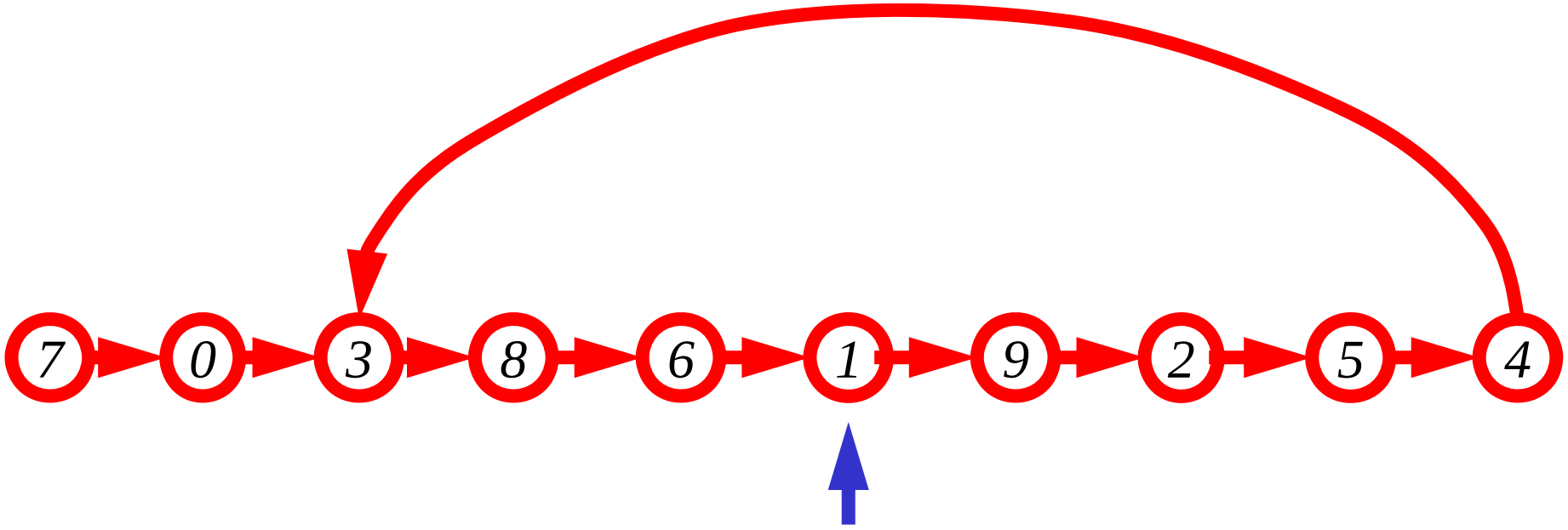
Claim: The maximal size of the stack is expected to be only logarithmic in the path length, requiring negligible memory





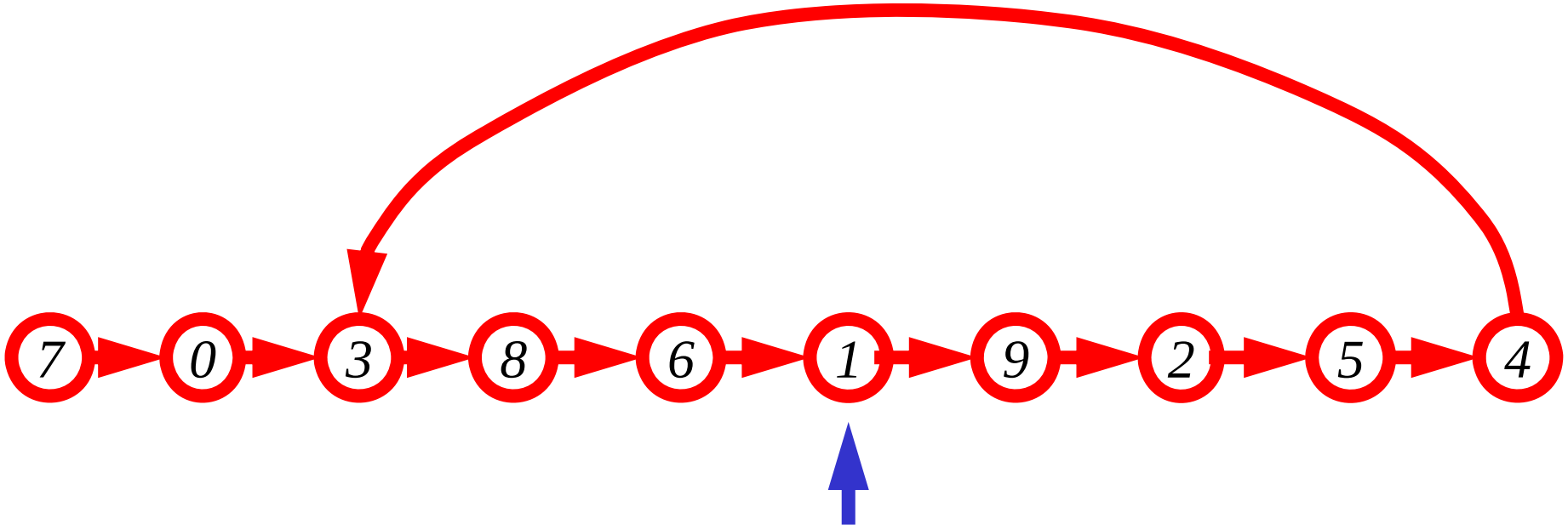
Claim: The stack algorithm always stops during the second cycle, regardless of the length of the cycle or its tail

0	1	1				



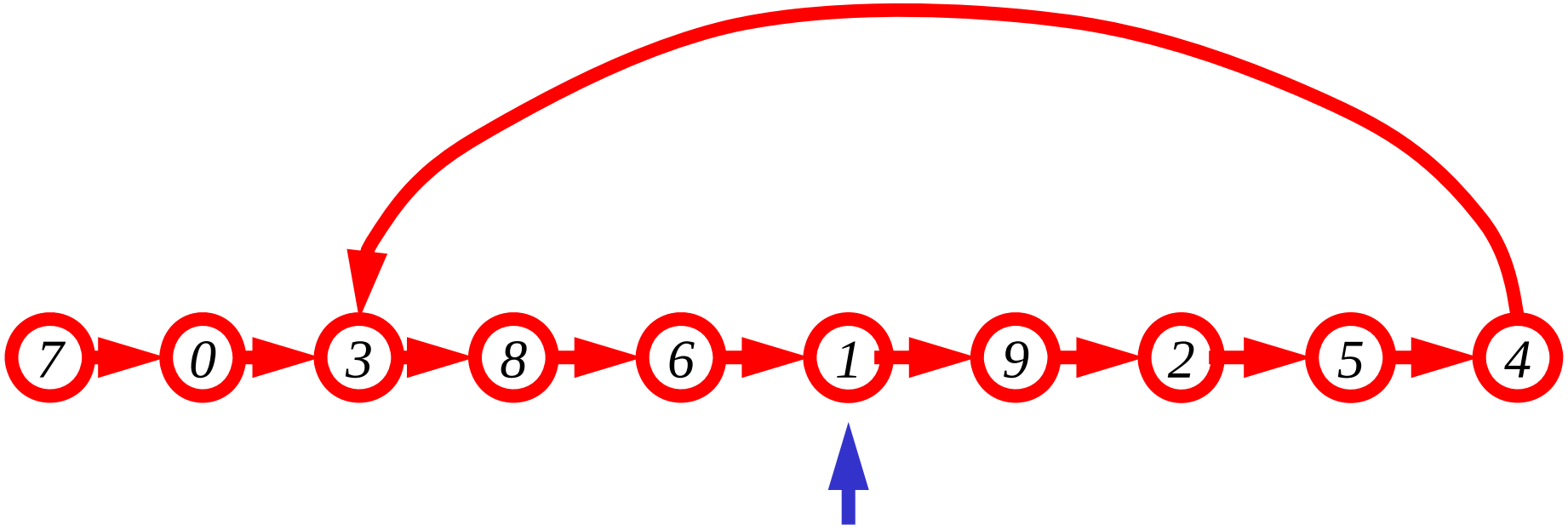
Proof: The smallest value on the cycle cannot be eliminated by any later value. Its second occurrence will eliminate all the higher values separating them on the stack.

0	1	1				

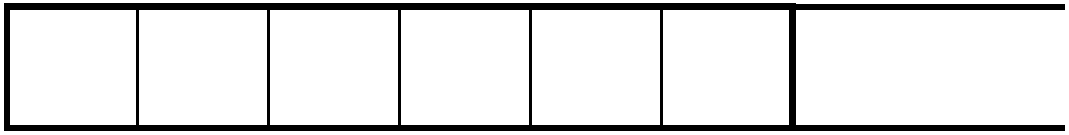


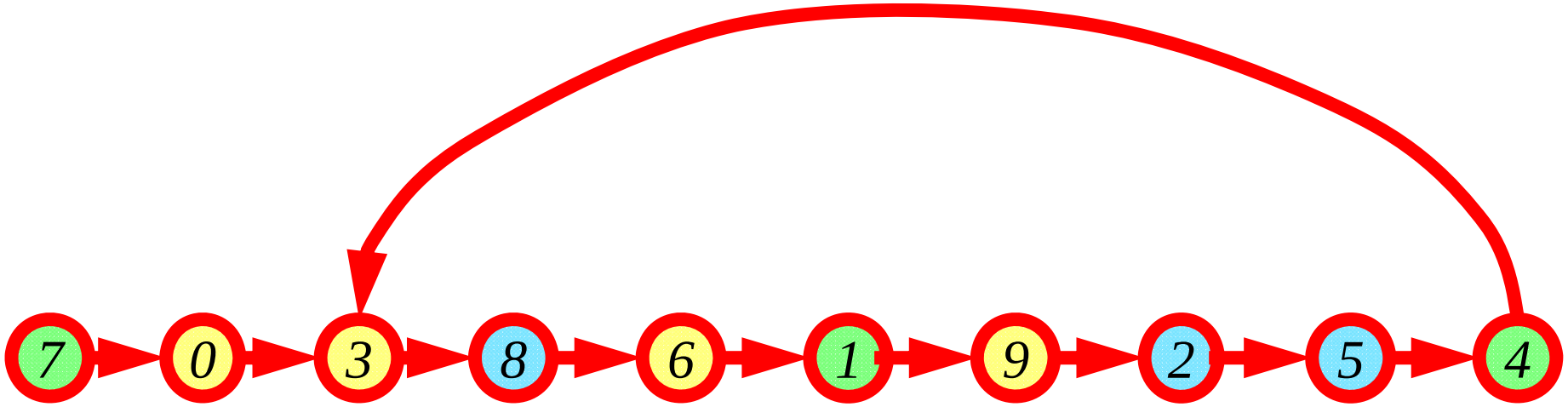
The smallest value in the cycle is located at a random position, so we expect to go through the cycle at least once and at most twice (1.5 times on average)

0	1	1				



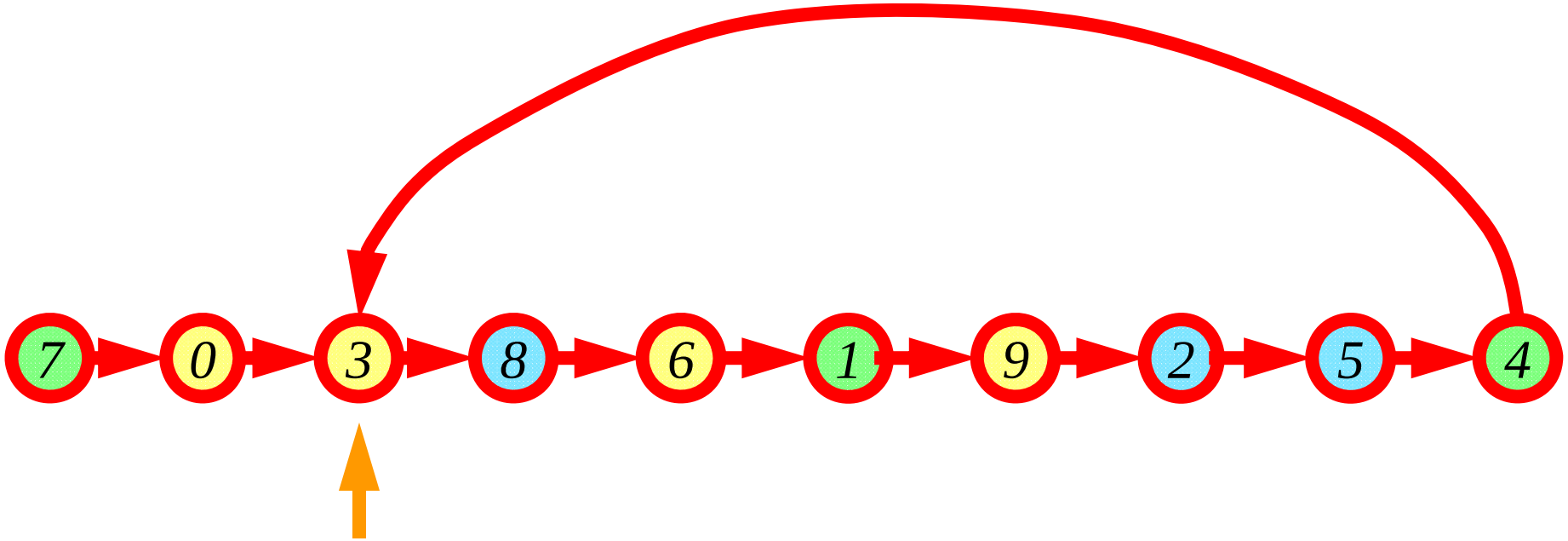
Improvement: Partition the values into k types, and use a different stack for each type. Stop the algorithm when repetition is found in some stack.





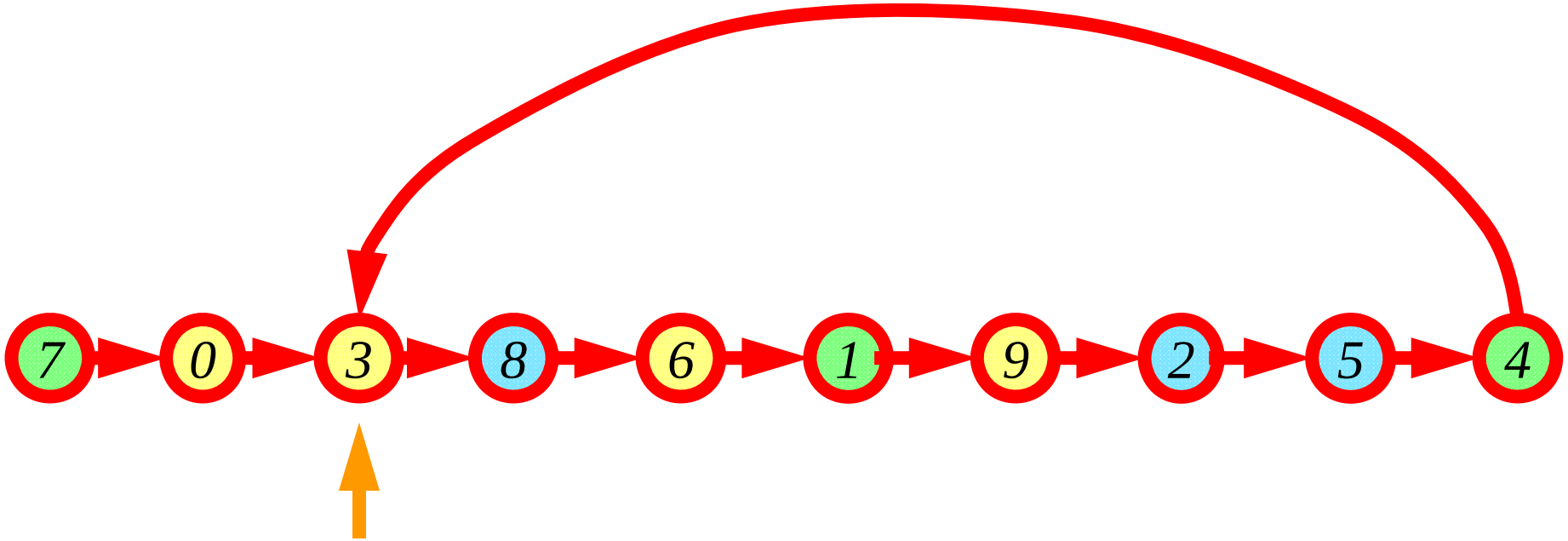
The new expected running time: (1+1/k)*n. Note that n is the minimum possible running time of any cycle detecting algorithm, and for k=100 we exceed it by only 1%

0	3	3				



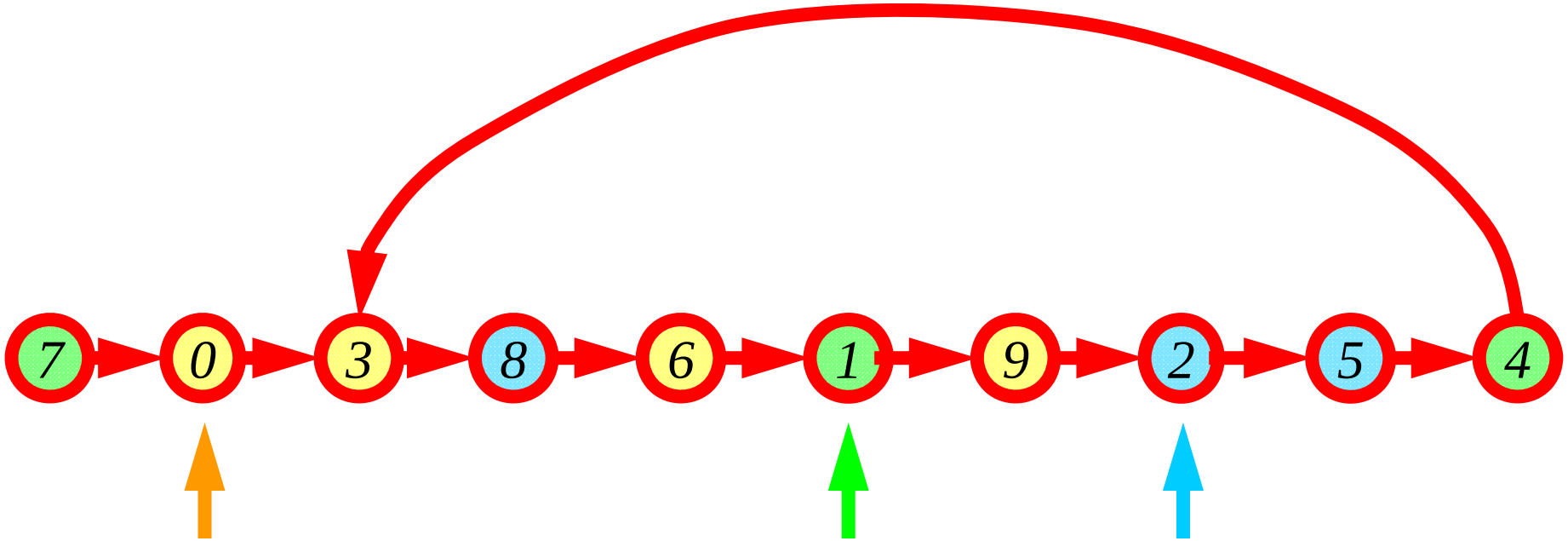
Unlike Floyd's algorithm, the Nivasch algorithm provides excellent approximations for the length of the tail and cycle as soon as we find a repeated value, with no extra work

0	3	3				



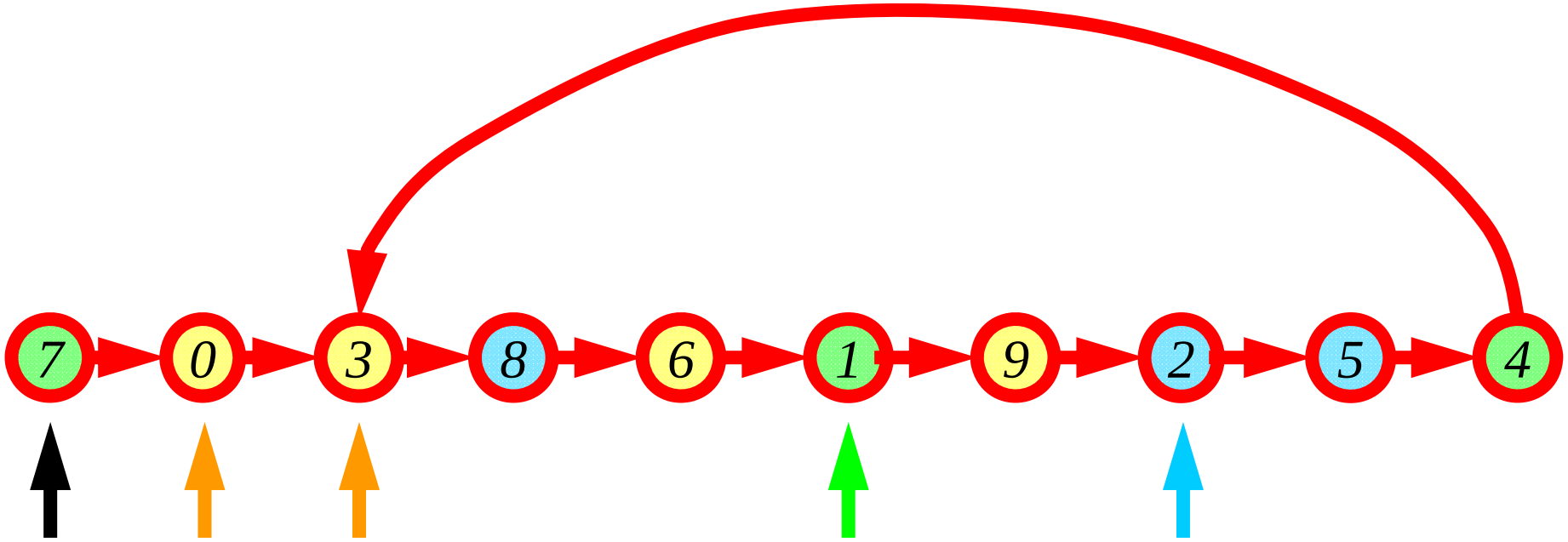
Note that when we stop, the bottom value in each stack contains the smallest value of that type, and that these k values are uniformly distributed along the tail and cycle

0	3	3				



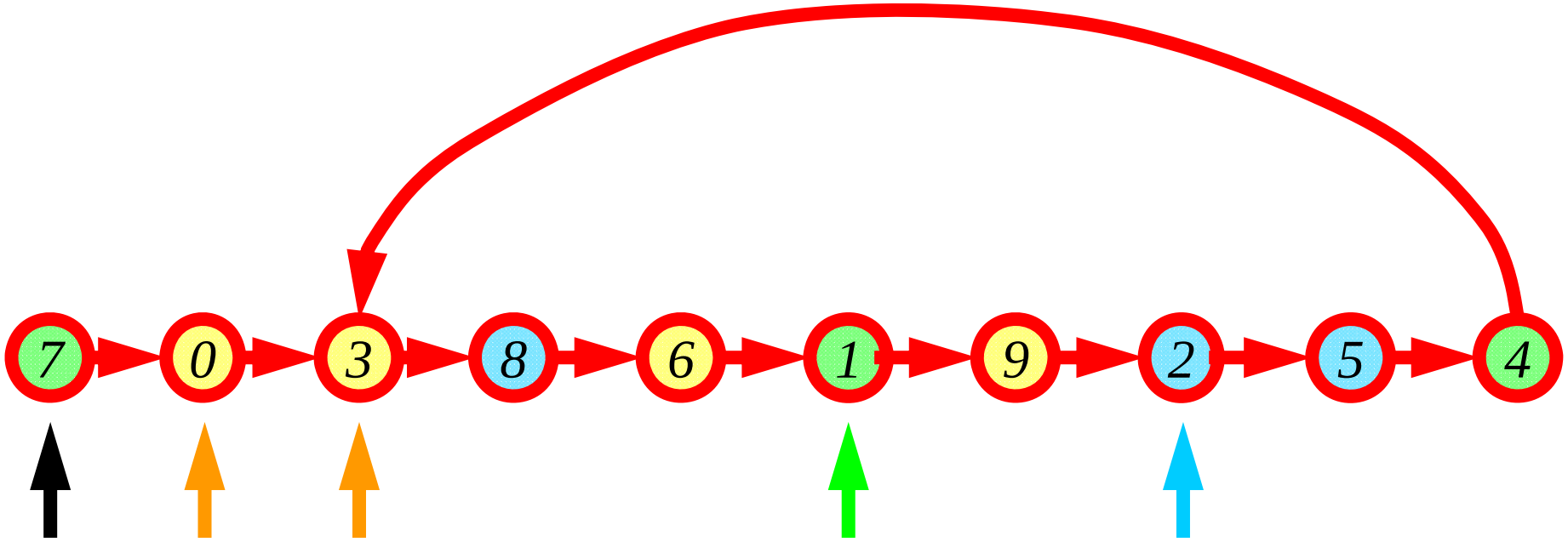
Adding two special points to the k stack bottoms, at least one must be in the tail and at least one must be in the cycle, regardless of their sizes

0	3	3				



We can now find the two closest points (e.g., 0 and 2) which are just behind the collision point. We can thus find the collision after a short synchronized walk

0	3	3				



Finding Multicollisions in Random Graphs:

A beautiful new result will be presented by Joux and Lucks at Asiacrypt 2009:

3-way collisions can be found in time $O(N^{2/3})$ and space $O(N^{1/3})$

Time and space can be traded off along the curve $TM=N$ for $M<N^{1/3}$

The tradeoff can be generalized from 3-collisions to r-collisions for any $r>3$

Lower bounds on Multicollision Finding:

Note that for 2-way collisions, we can use a constant amount of memory and get a $N^{1/2}$ time bound.

An unpublished lower bound I recently obtained while working on the same problem proves the optimality of the Joux and Lucks algorithm for 3-collisions

The Model of Computation:

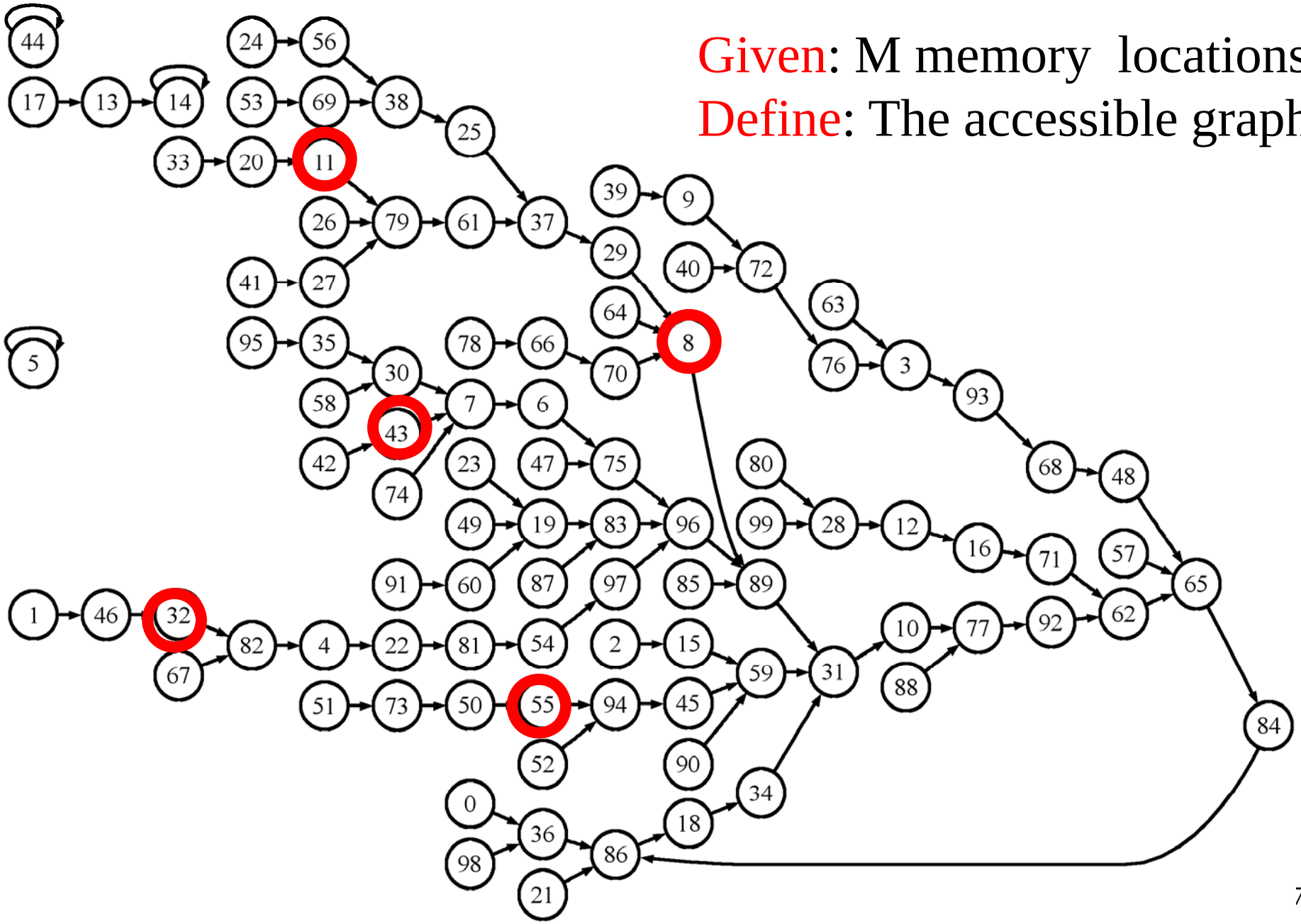
At any moment, the attacker can:

- Store a fresh random vertex in some memory location, replacing its old contents
- Copy one memory location into another
- Replace the vertex stored in some memory location by its successor vertex

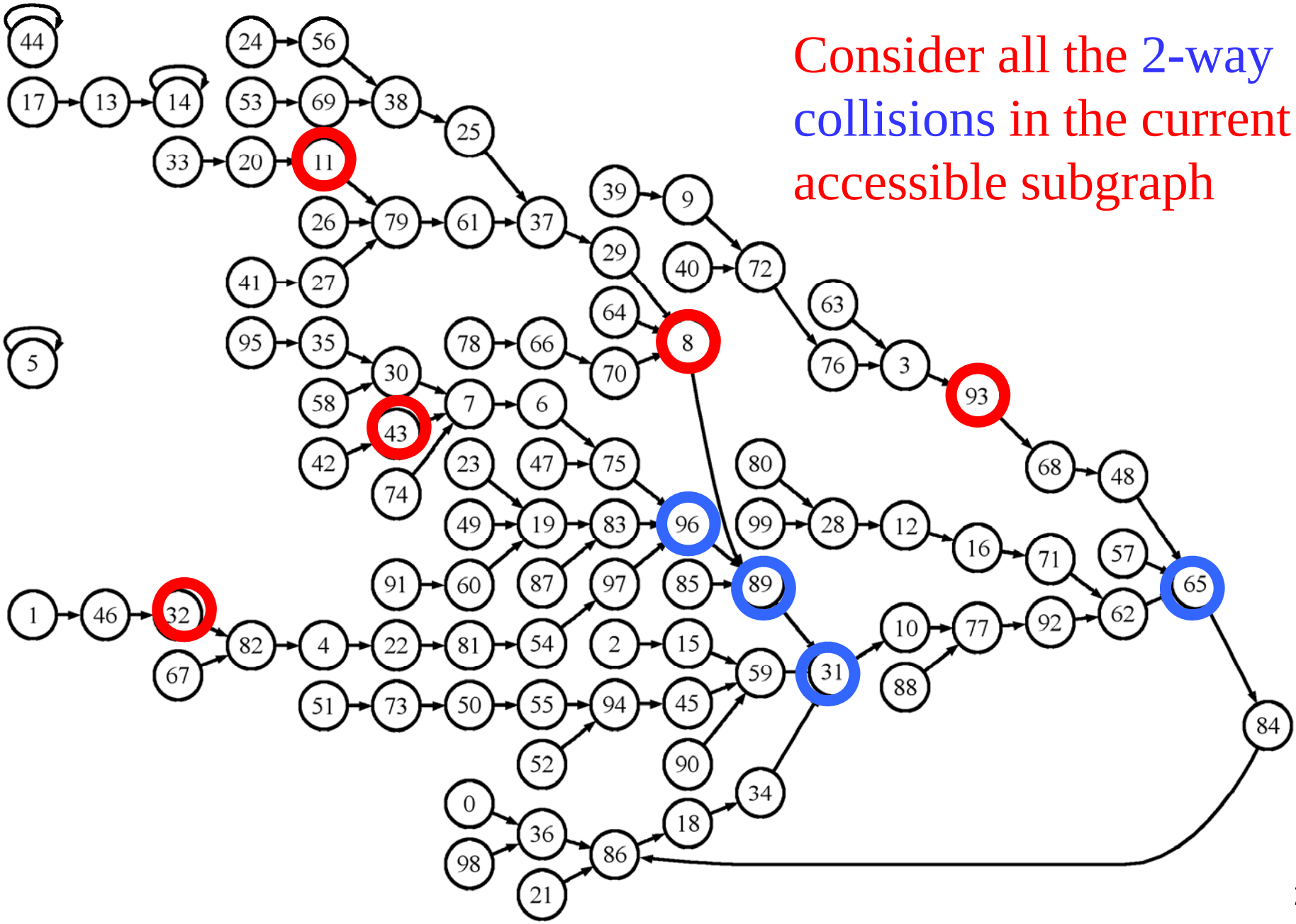
The Basic Idea of the Lower Bound:

Given: M memory locations

Define: The accessible graph



The Basic Idea of the Lower Bound:



The Lower Bound Proof:

The accessible graph is a dynamic, time-dependent subgraph of the full random graph.

The main observation: The accessible subgraph defined by M stored points can contain at most M 2-way collisions, and every 3-way collision was at some stage a 2-way collision which was hit by a new edge from a third direction

The attacker might not be currently aware of most of the 2-way collisions in his current accessible subgraph, but he could find them later by following some paths in a particular order from the stored vertices.

The Lower Bound Proof:

At the end of the 3-way collision finding algorithm, the attacker is fully aware of the 3-way collision since he has to supply its 3 predecessors

Consider the first point in time in which the attacker traversed an edge whose head is an implicit 2-way collision defined by the currently stored vertices (such a time must exist)

Since the number of 2-way collisions is bounded by O(M), this is unlikely to happen if he traverses fewer than O(N/M) edges altogether in the whole algorithm

A Different Problem: Inverting Edges

The Fundamental Problem of Cryptanalysis:

Given a ciphertext, find the corresponding key

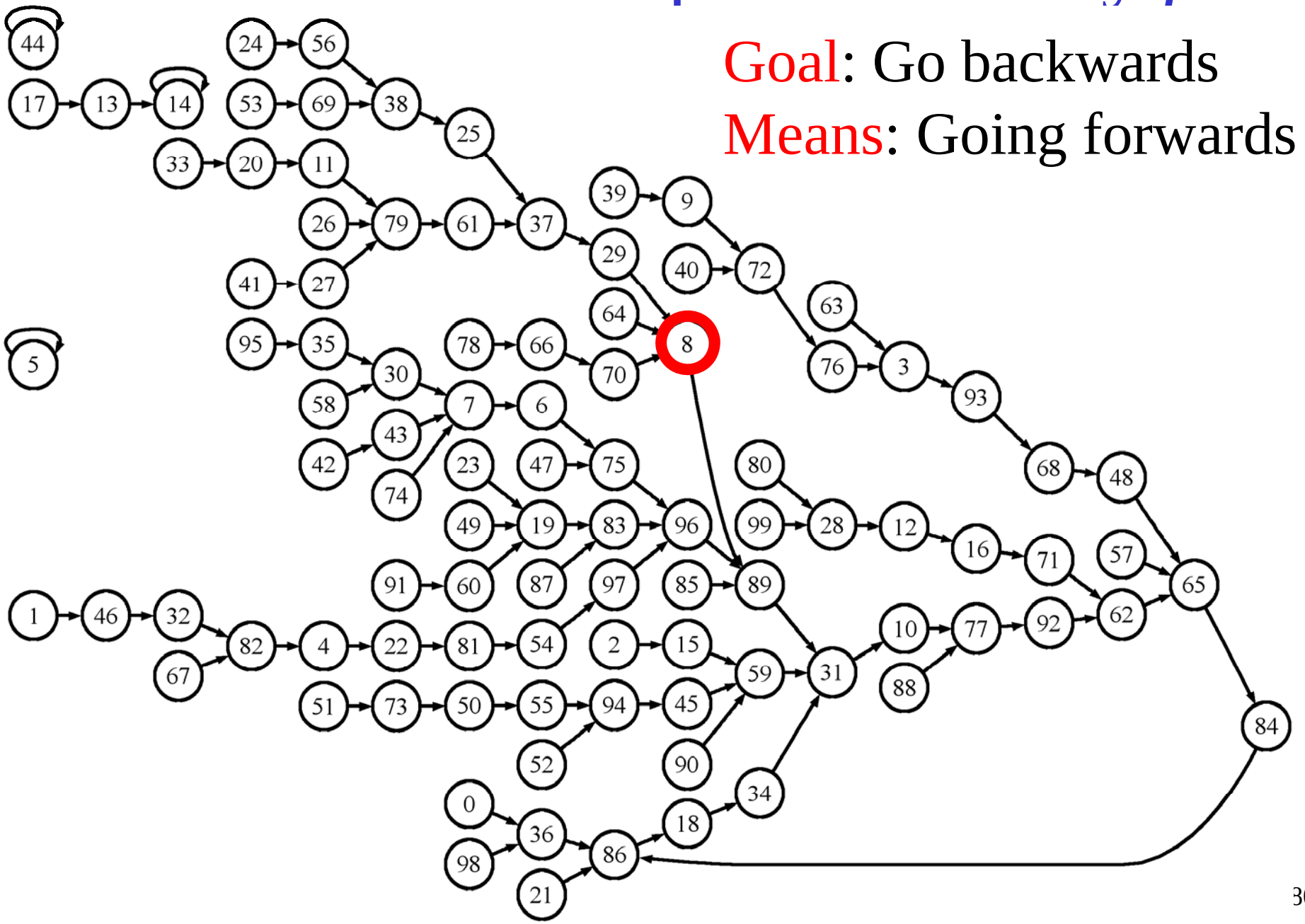
Given a hash value, find a first or second preimage

The mathematical problem: Invert an easily computed random function f where $f(x)=E_x(0)$ or $f(x)=H(x)$

The Random Graph Defined by *f*:

Goal: Go backwards

Means: Going forwards

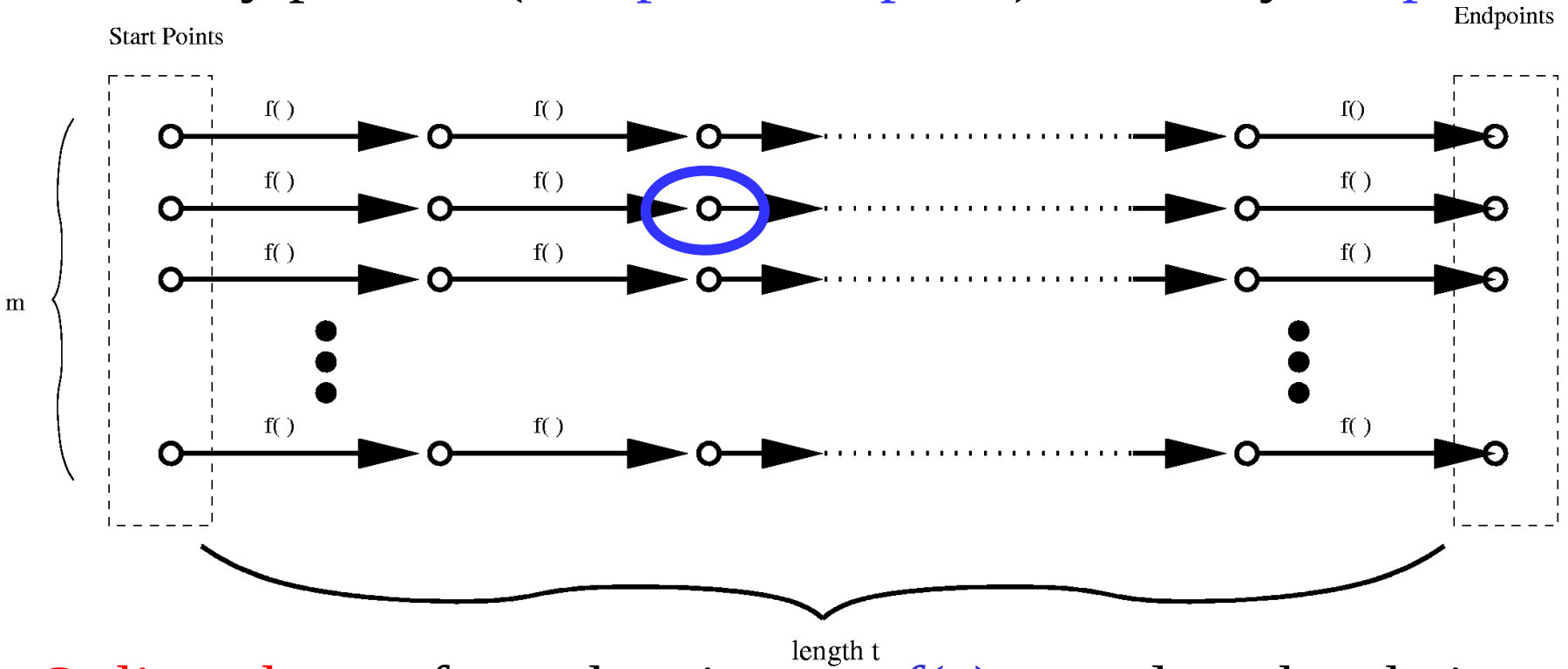


Hellman's T/M Tradeoff (1979)

Preprocessing phase:

Choose *m* random starting points, evaluate chains of length *t*.

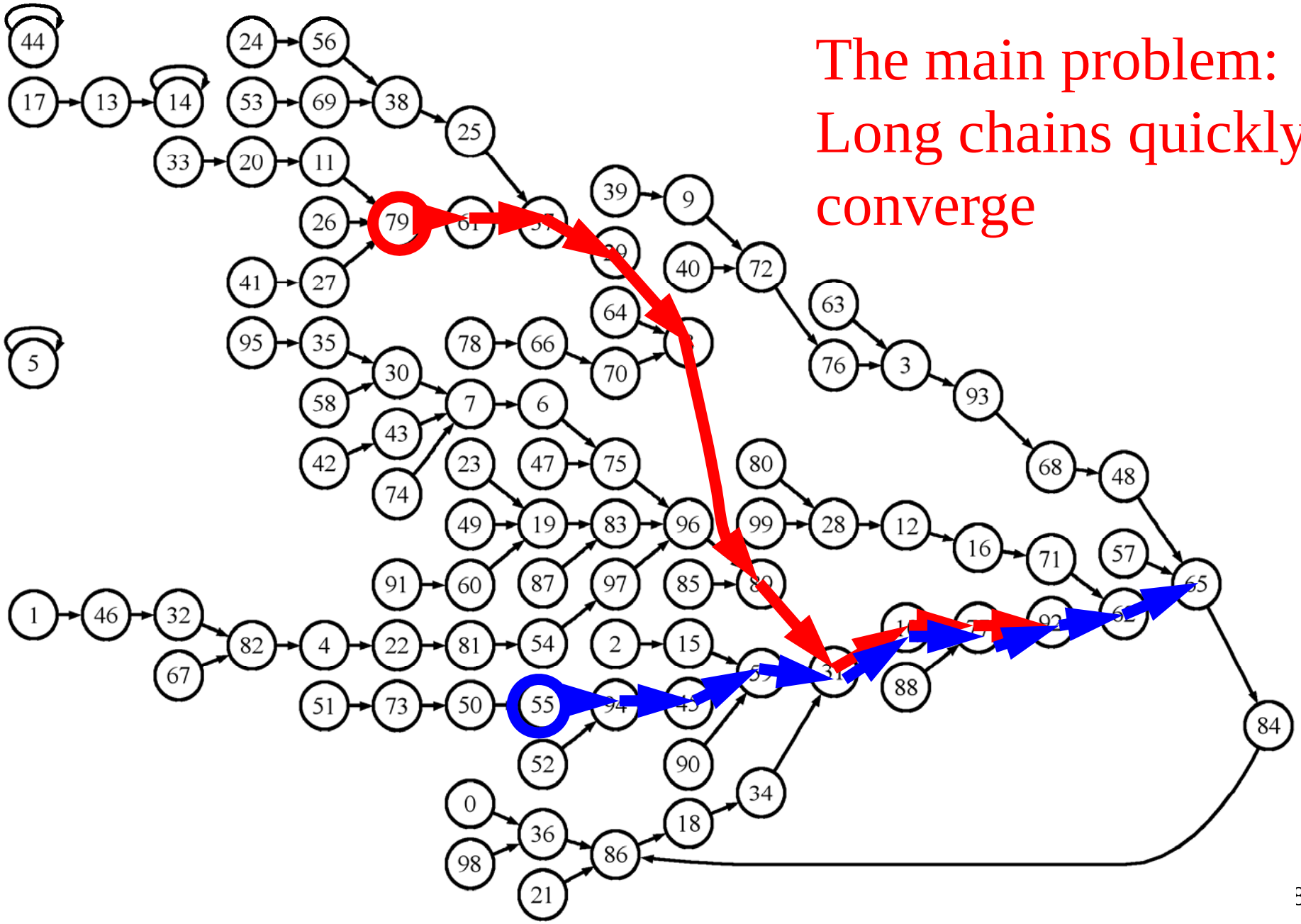
Store only pairs of (startpoint,endpoint) sorted by endpoints.



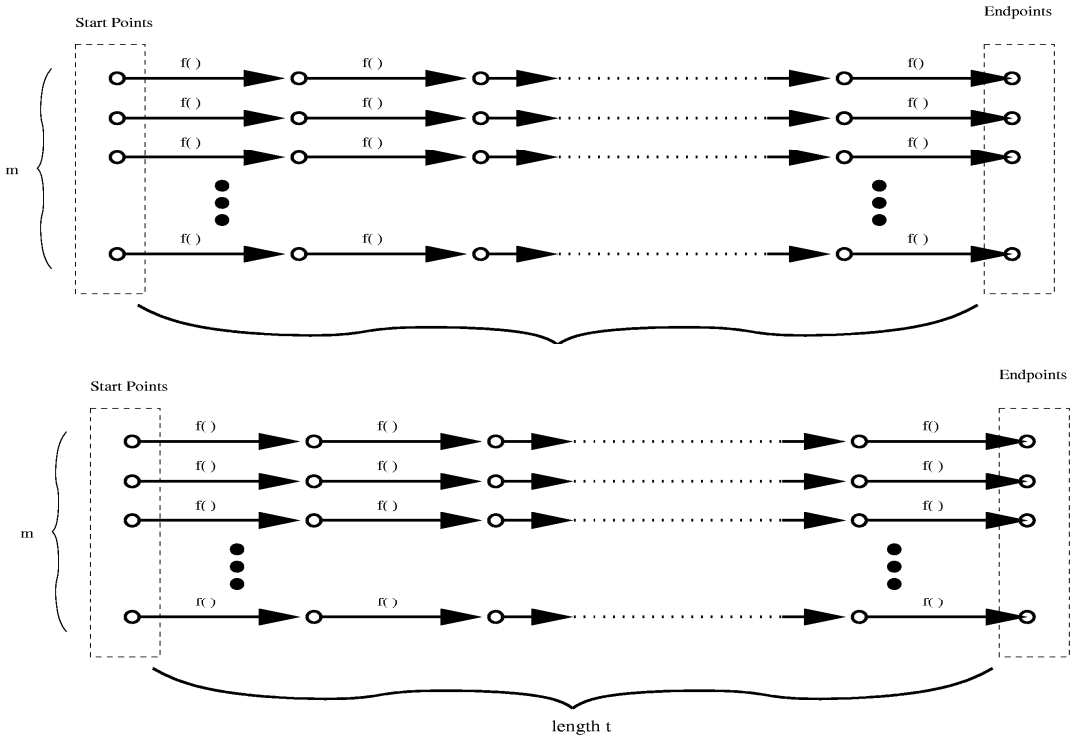
Online phase: from the given $y=f(x)$ complete the chain.

Find *x* by re-calculating the chain from its startpoint.

How can we cover this graph by chains?



Hellman's Solution:



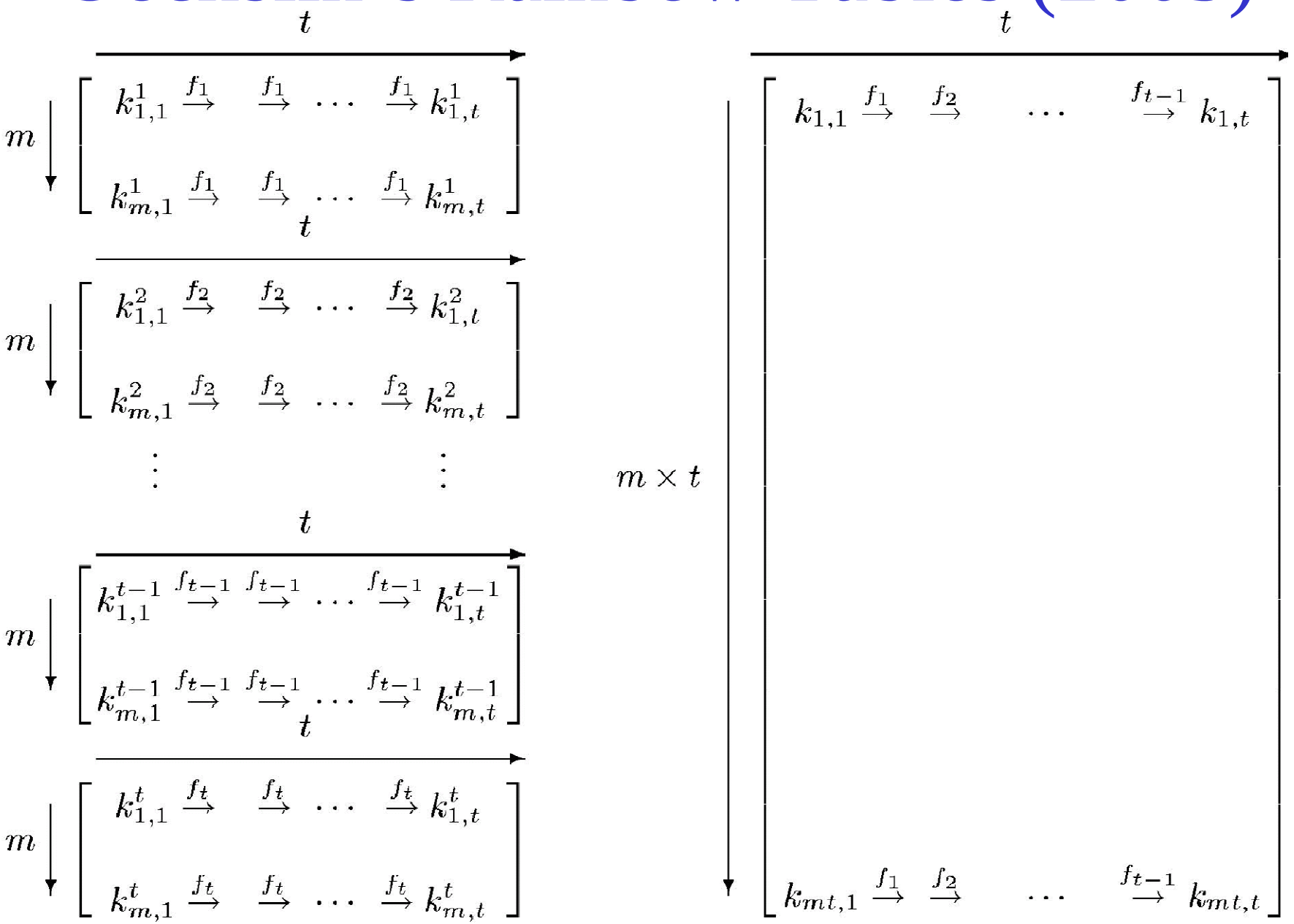
Use t "independent" tables from t "related" functions $f_i(x)=f(x+ i \bmod N)$

– note that inversion of $f_i \Rightarrow$ inversion of f.

Yields a general T/M tradeoff: $TM^2=N^2$.

Typical complexities: Time $T=N^{2/3}$, space $M=N^{2/3}$

Oechslin's Rainbow Tables (2003)



There are many other possible tradeoff schemes:

Use a different sequence of functions along each path, such as:

111222333 or 123123123 or
pseudorandom e.g. 1221211

Make the choice of the next function dependent on previous values

What kind of random graph are we working with in such schemes?

There was already a slight problem with the multiple graphs of Hellman's scheme, since they are not really independent, and there are subtle relationships between their structures

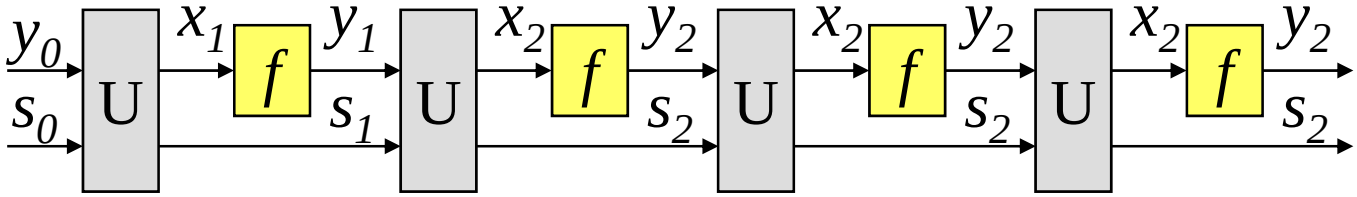
Oechslin's graphs are even weirder, since their multiple functions and layered structure does not look like a random graph at all

Barkan, Biham, and Shamir (Crypto 2006):

Introduced a new notion of random graph called Stateful Random Graph

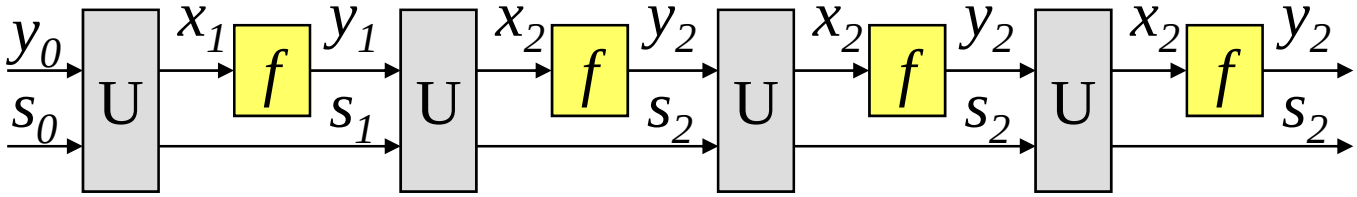
Used it to prove rigorous lower bounds on the achievable time/memory tradeoffs of any scheme which is based on such graphs, including Hellman, Oechslin, and all their many known variants and extensions

The Random Stateful Graph Model



- The nodes in the graph are pairs (y_i, s_i), with N possible images y_i and S possible states s_i.
- The scheme designer can choose any U, then random f is given.
- The increased number of nodes (NS) can reduce the probability of collisions and a good U can create more structured graphs.
- Examples of states: Table# in Hellman, column# in Oechslin.
- We call it a *hidden state*, since its value is *unknown* to the attacker and has to be guessed when he tries to invert an image y .

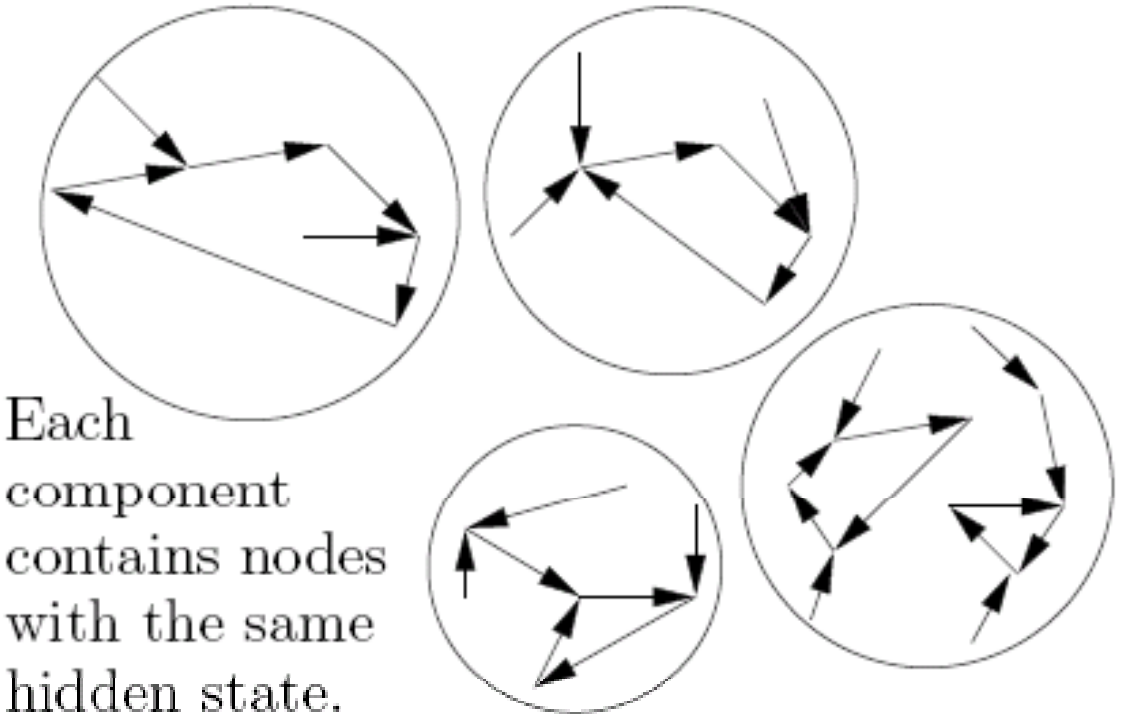
The Stateful-Random-Graph Model – cont



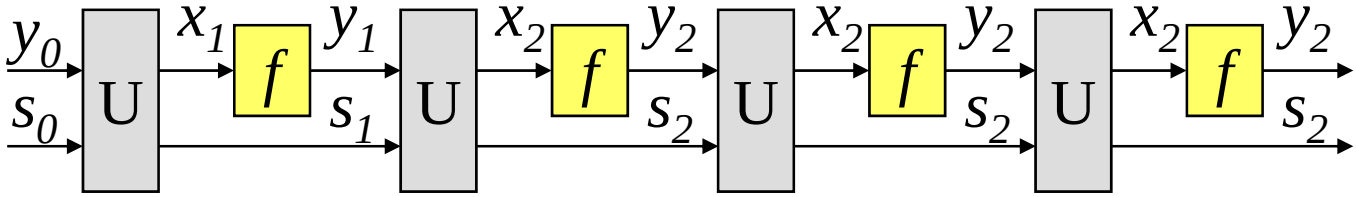
U in Hellman:

$x_i = y_{i-1} + s_{i-1} \bmod N$

$s_i = s_{i-1}$



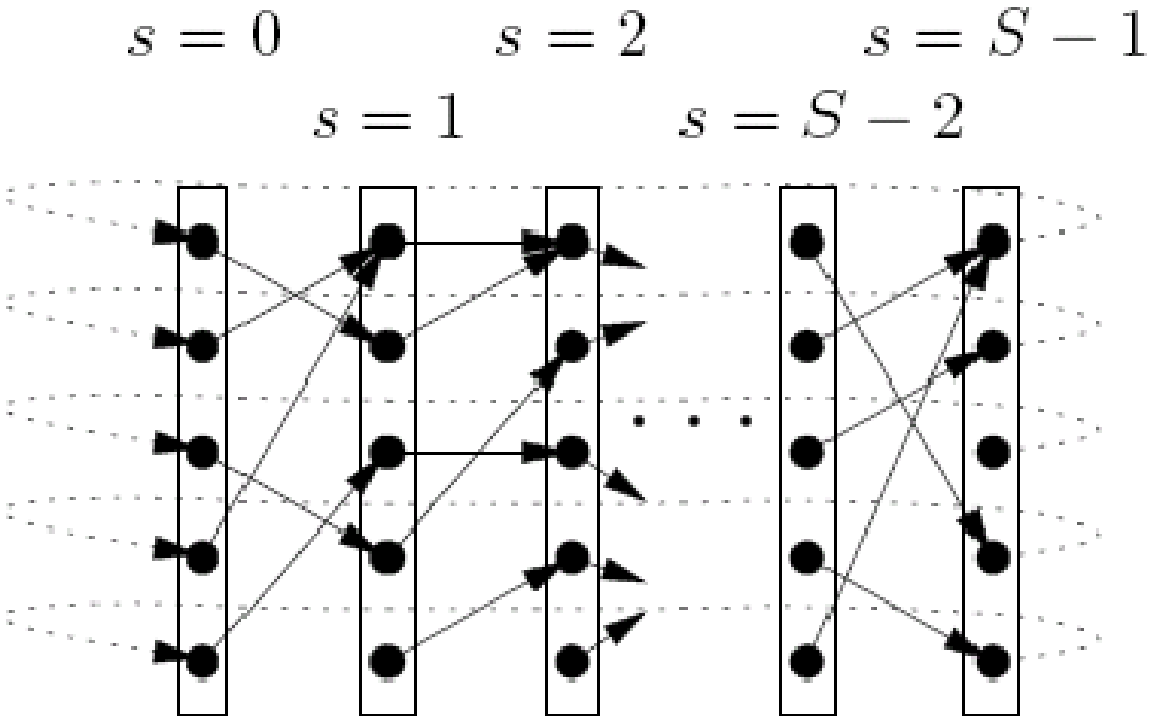
The Stateful-Random-Graph Model – cont



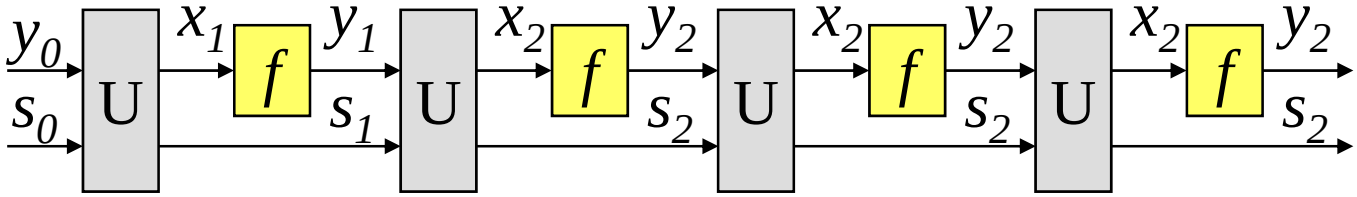
U in Rainbow:

$x_i = y_{i-1} + s_{i-1} \bmod N$

$s_i = s_{i-1} + 1 \bmod S$.



The Stateful-Random-Graph Model – cont

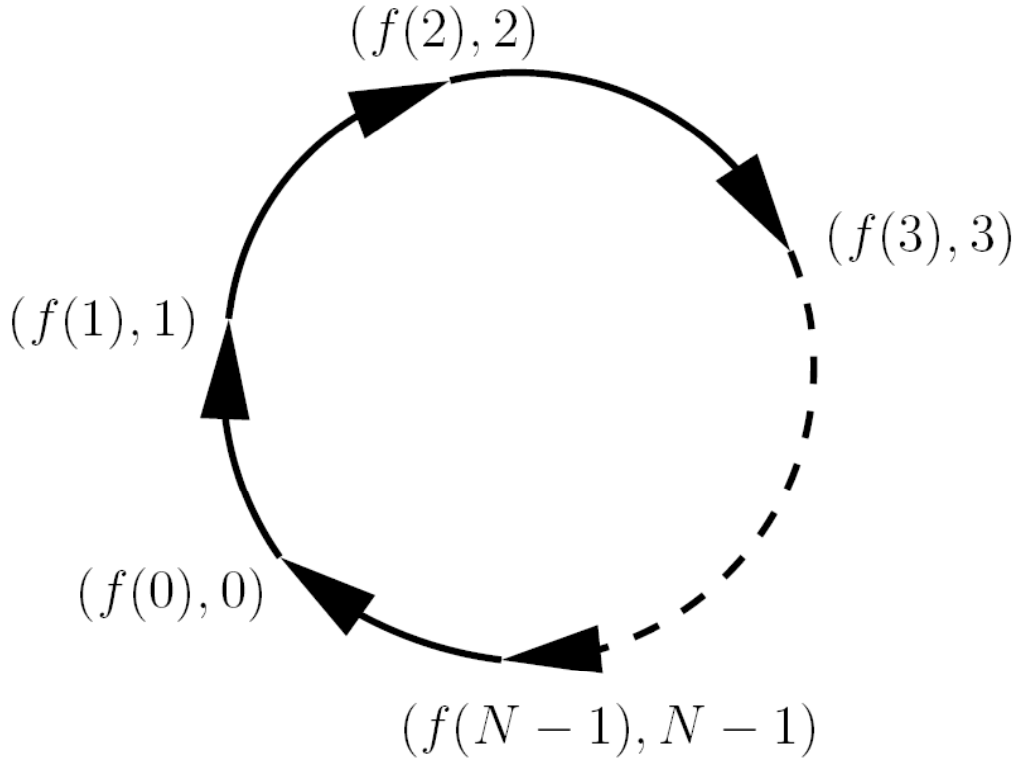


U in exhaustive search

$x_i = s_{i-1}$

$s_i = s_{i-1} + 1 \bmod N$,

which goes over
all the preimages
of f in a single cycle



The rigorously proven Coverage Theorem (exact statement, with no hidden constants):

For any U with S hidden states,

with overwhelming probability over random f's,

the coverage of any collection of M paths of any length in the stateful random graph defined by U

is bounded from above by 2A, where

$$A = \sqrt{SNM \ln(SN)},$$

Corollaries:

To cover most of the vertices of any stateful random graph, you have to use a sufficiently large number of hidden states, whose guessing determines the minimal possible running time of the online phase of the attack in any such scheme.

This lower bound is applicable to Hellman's scheme, to the Rainbow scheme, and to all their known variations, and proves their optimality up to logarithmic factors

Which Time/Memory Tradeoff Scheme Has the Best Constants?

- Oechslin claimed that his TMTO is far better than Hellman's due to its lower number of false alarms

- On the other hand, Hellman's startpoints can be represented by half the number of bits, saving a lot of memory

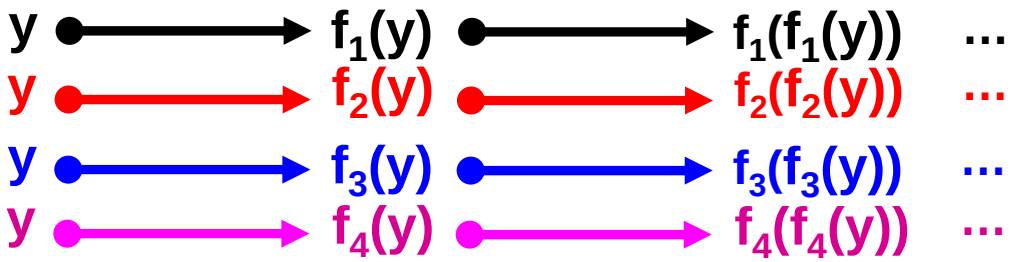
Meet the New World Champion:

- Hoch and Shamir (2009): A novel variant called *pH* (abbreviated from *P*arallel *H*ellman) for the online stage of Hellman's TMTO

- For the same memory and coverage, *pH* can be up to twice as efficient as the Hellman and Oechslin TMTO's

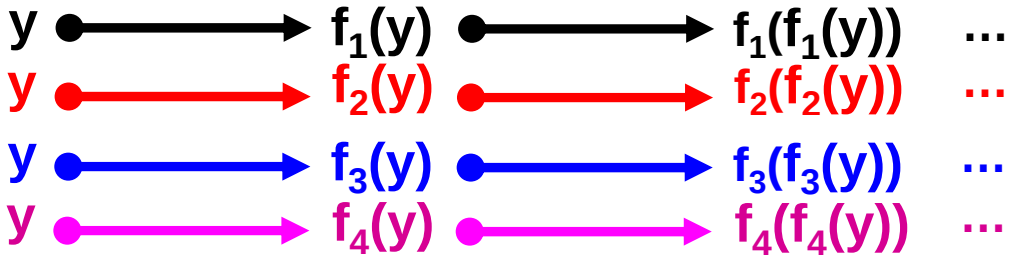
The standard *H* Algorithm

- Instead of searching for the inverse in each table sequentially…



The New *pH* Algorithm

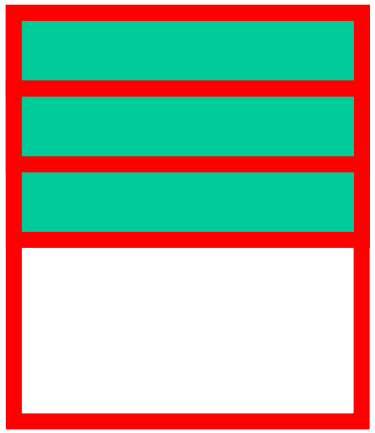
- Instead of searching for the inverse in each table sequentially…
- …We search in all tables in parallel (with a single processor and additional fast memory)



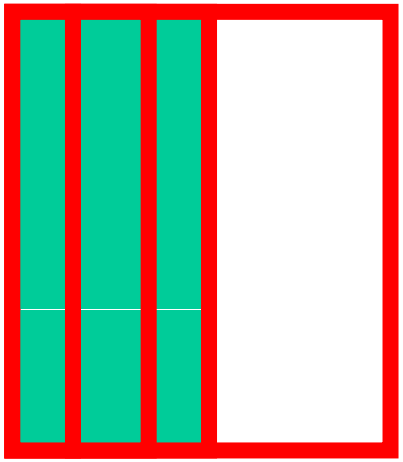
Where does the Improvement Come from?

- A random target point occurs near the middle column and the middle row in all the tables
- The search algorithms cover the following areas:

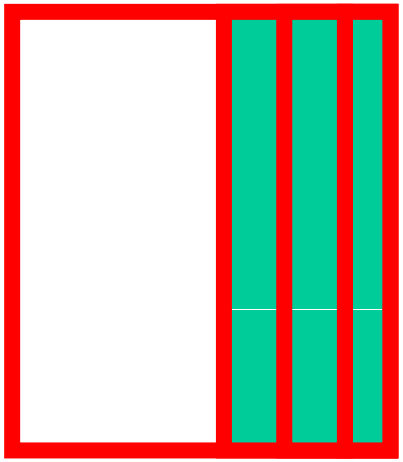
Hellman:



Parallel Hellman:



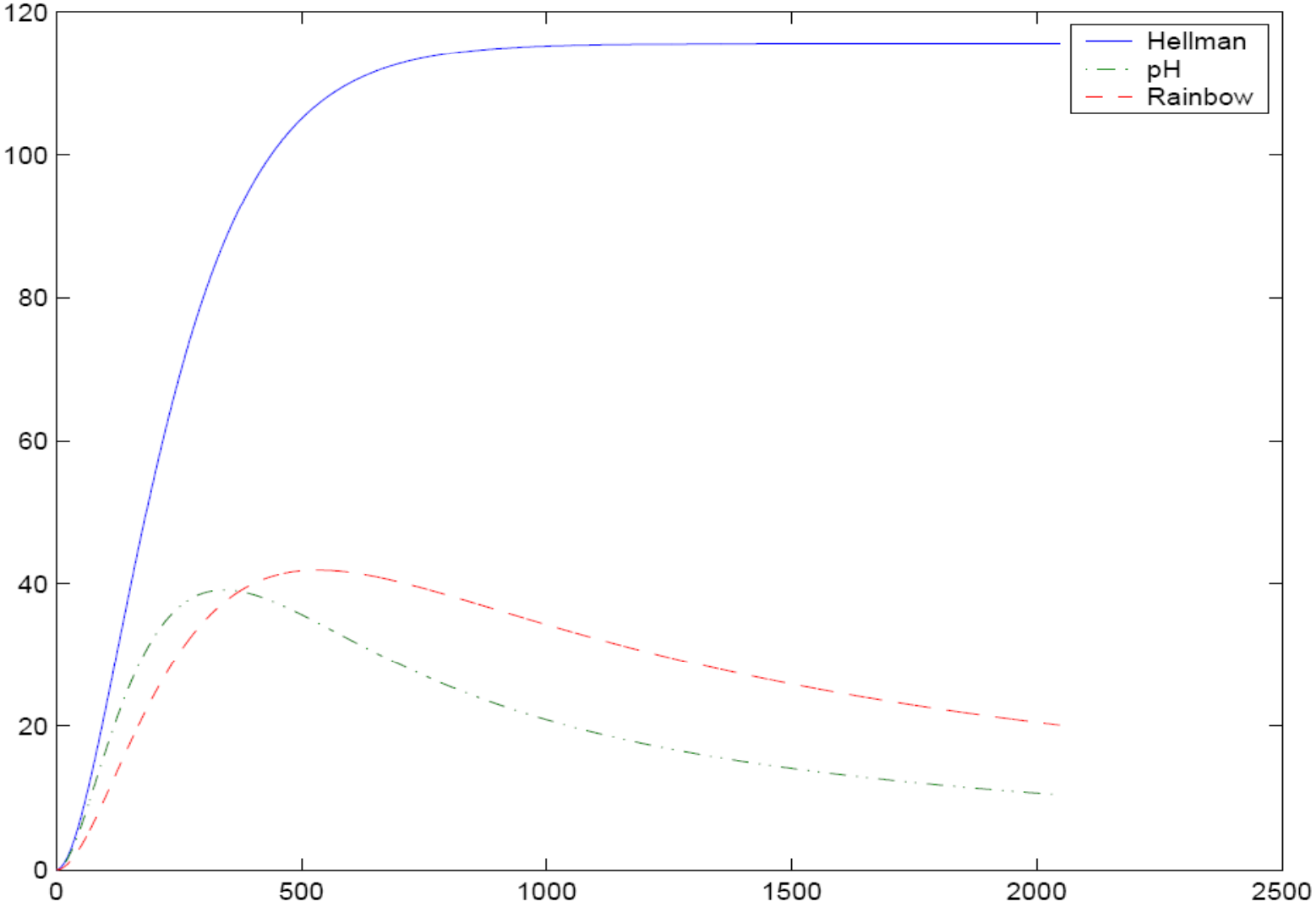
Oechslin:



Where does the Improvement Come from?

- The number of false alarms rapidly increases with chain length. Since pH examines chains of about half length compared to Hellman, it combines the main benefits of Oechslin's rainbow tables (fewer false alarms) and Hellman's tables (more compact endpoint representation)
- To run the pH algorithm, we need $N^{1/3}$ fast sequential memory for intermediate values (one per table) in addition to the $N^{2/3}$ slow random access memory for the endpoints of all the precomputed tables
- We can equalize the processor and memory speeds by evaluating k edges rather than a single edge in each step and storing the values in consecutive memory locations. For small k this has negligible effect on the number of false alarms, and runs the ALU and memory at full speed

False alarms vs. # of tables



Numerical Results, Comparing Classical Hellman with pH for the Same Memory and Coverage Parameters

Algorithm	Coverage (percent)	#False alarms	False work (iterations)	Total work (iterations)
Hellman	86.5	62	19150	31100
pH	86.5	42.5	13000	24576
Hellman	96.6	73	22678	47314
pH	96.6	36	11300	36000
Hellman	99.9	77.22	23785	57249
pH	99.9	17.65	5758	42142

Figure 3.4: Results of simulations with t=m=256. The table compares the classical Hellman and pH algorithms for three coverage fractions. The parameters compared are the number of false alarms, the number of f iterations used to rule out false alarms and the total number of f iterations.

Numerical Results, Comparing pH with Oechslin's Rainbow Tables for the Same Memory and Coverage Parameters

Algorithm	Coverage	False alarms	False work	Total work
Parallel Hellman full end points	86	86.5	52711	97022
Rainbow full end points	87	170	103500	177000
Parallel Hellman 13 bit end points	85	160	104000	134130
Rainbow 22 bit end points	86	272	194000	244000
Parallel Hellman full end points	99.5	46	28500	165000
Rainbow full end points	99.5	215	129000	409000
Parallel Hellman 13 bit end points	99.5	178	111000	246000
Rainbow 22 bit end points	99.5	343	241000	520000

Figure 3.5: 30-bit function, 0.86 and 0.995 coverage

To conclude, our implementation of pH shows an improvement of approximately 50% in the total running time over the online phase of the rainbow algorithm. Thus our parallel online algorithm, pH, together with the savings on start and end point memory gives the best performance of any variant of time/memory tradeoff proposed so far.

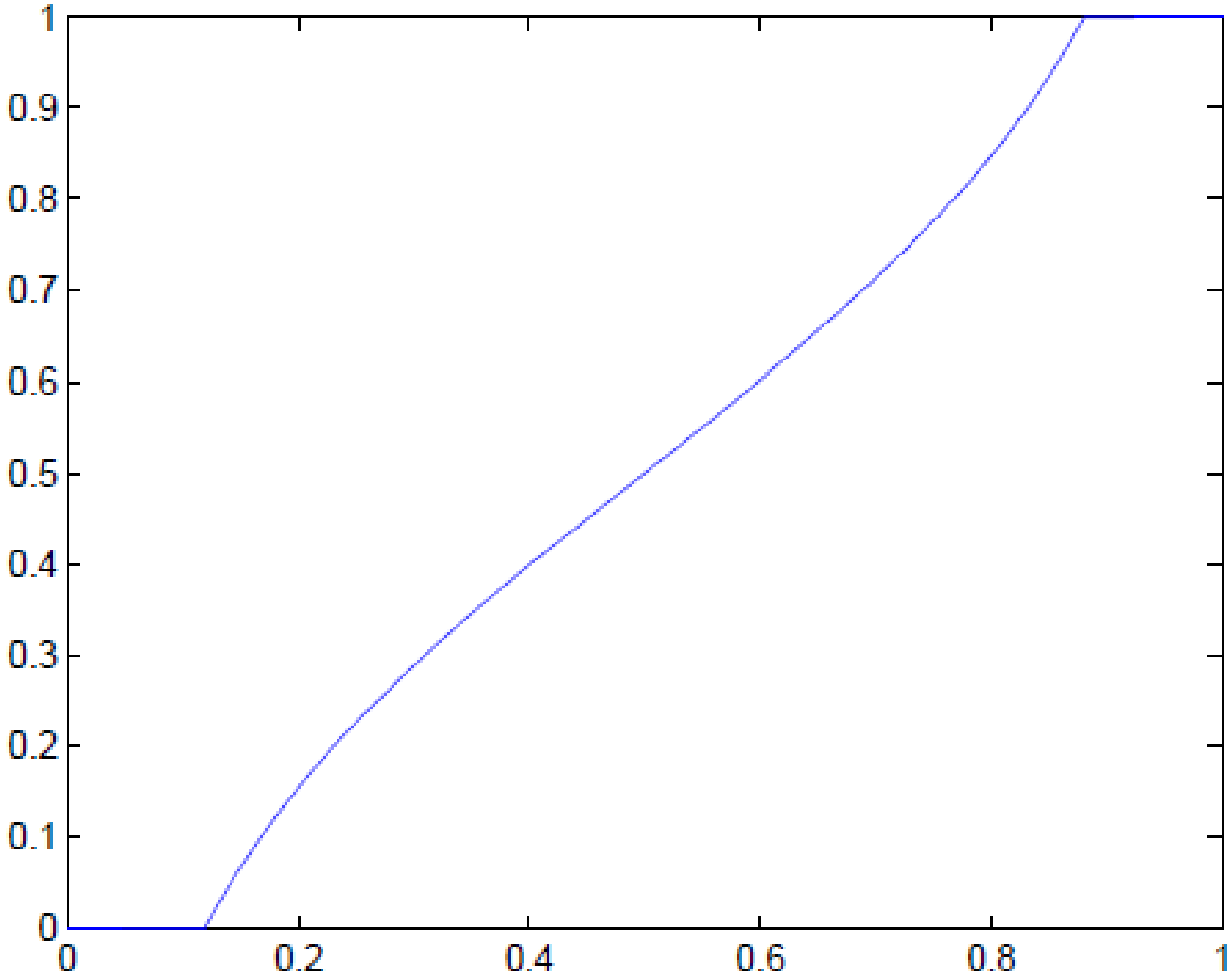
Non-Uniform Point Distributions

- TMTOs are used mostly in password cracking
- Passwords are NOT distributed uniformly
- Hoch and Shamir (2009) developed improved TMTO's for non-uniform input distributions

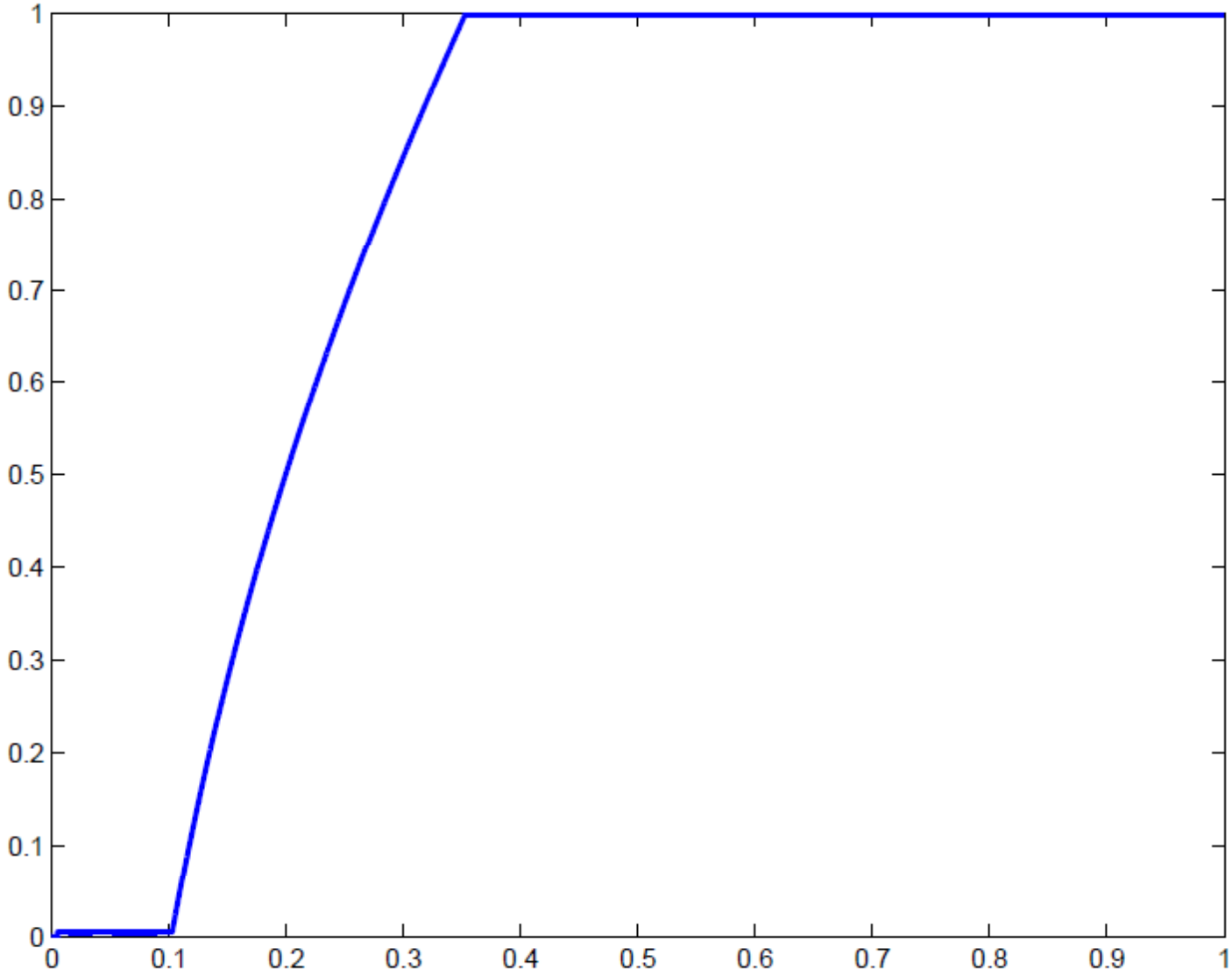
The Models

- Every application of the function is followed by a ‘sampling’ of the input
- We examined two models
  - We can choose a probability for each element to be sampled
  - We can choose a probability for each element to be sampled in each column (only analyzed two populations)

Optimal Sampling Probability vs. Weight for Two Populations



Optimal sampling probability vs. weight for a single column



Adding Privacy To Biometric Databases: Using Random Graphs to Identify People

- Many governments (including in Israel) plan to issue new ID cards in the near future
- They are facing strong public opposition mainly due to privacy concerns
- The five possible solutions:

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

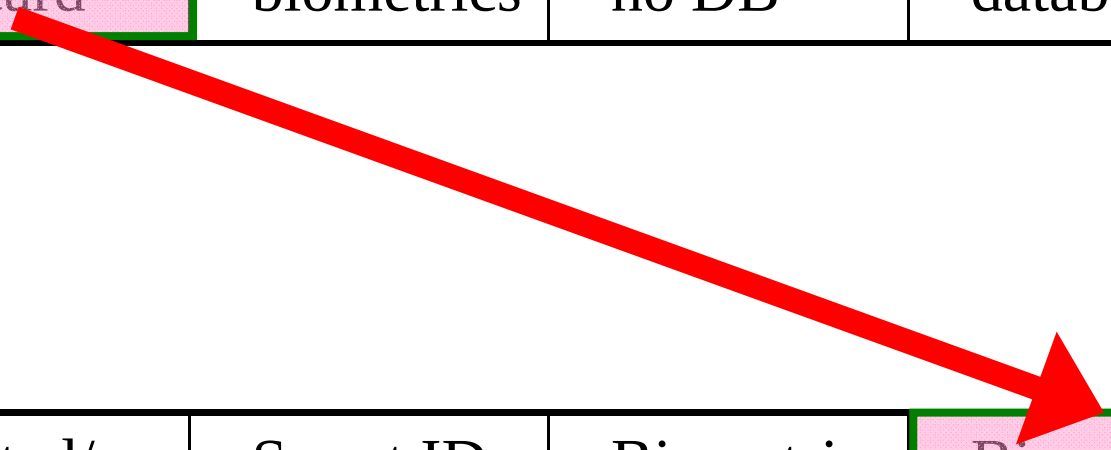
The Planned Transition in Israel

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

The Planned Transition in Israel

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database



No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

The Planned Transition in Israel

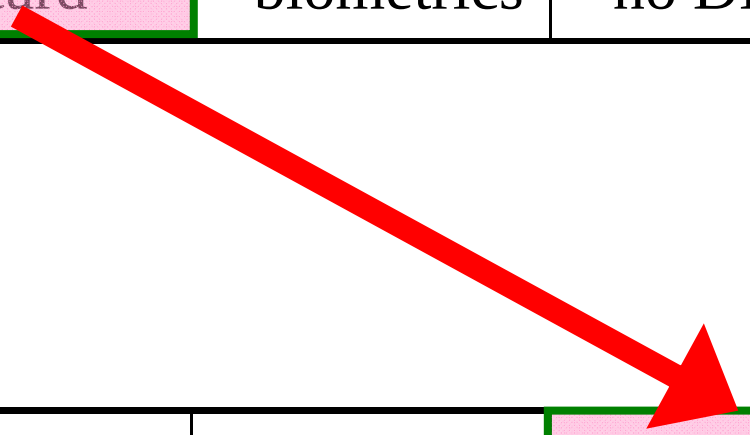
No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

preferred by authorities,
strongly opposed by public

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

The Planned Transition in Israel

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database



No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

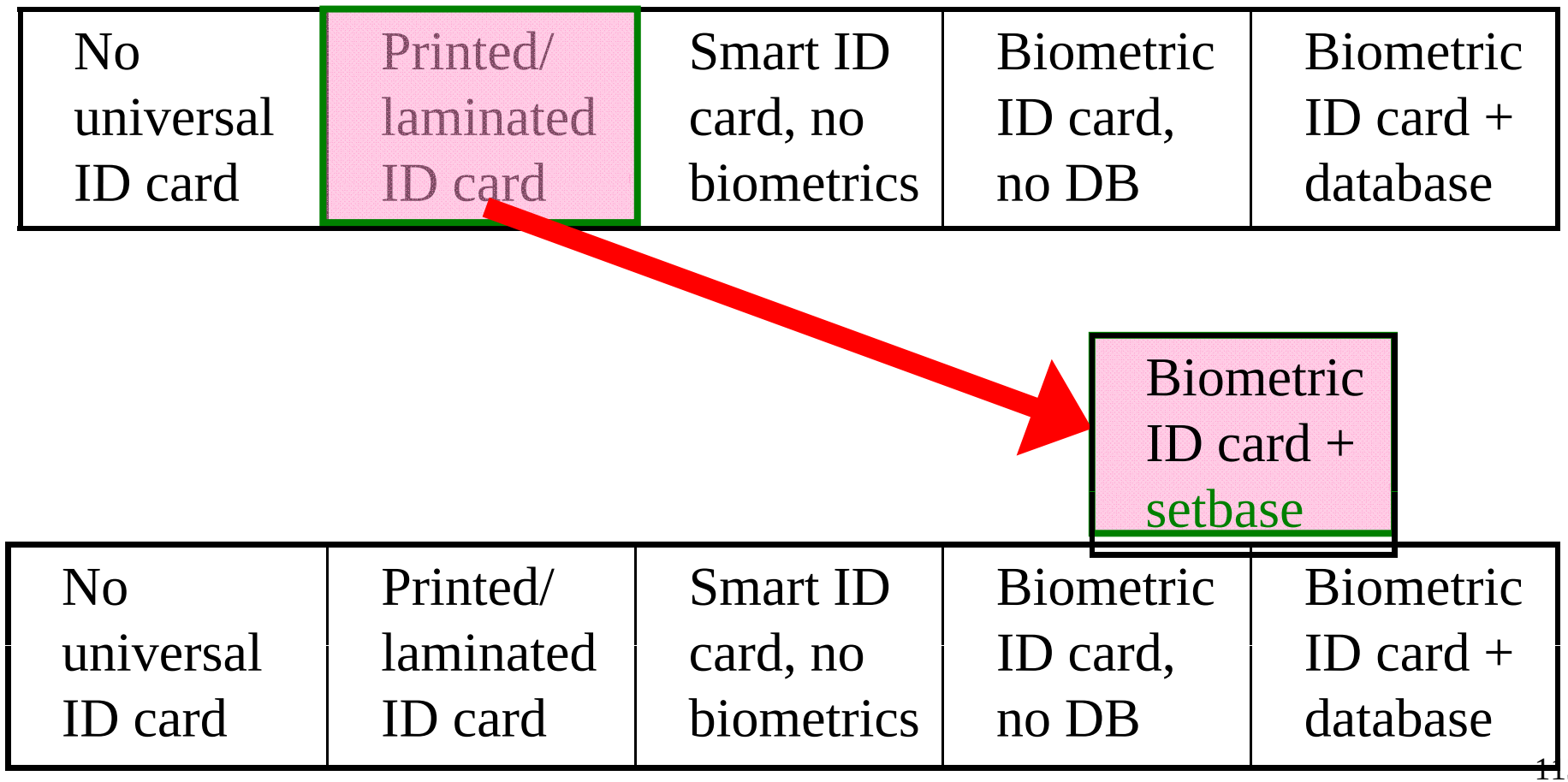
The Planned Transition in Israel

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

rejected by authorities,
almost no public opposition

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

My Proposal: A Biometric Setbase



My Proposal: A Biometric Setbase

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

acceptable to authorities,
solves most privacy concerns

Biometric ID card + setbase

No universal ID card	Printed/ laminated ID card	Smart ID card, no biometrics	Biometric ID card, no DB	Biometric ID card + database

The Official Reasons for Creating a Biometric Database in Israel:

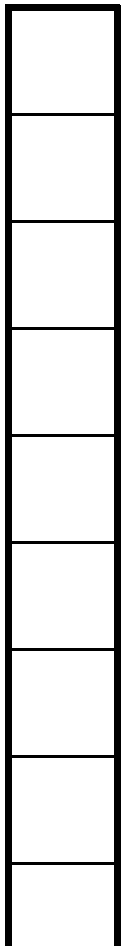
- Major reason: Preventing double issuing of official ID cards to criminals and crooks
- Minor reason: Identifying paperless bodies and solving major crimes – in very rare cases

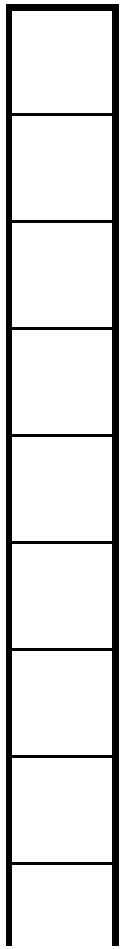
The Main Counterarguments of Privacy Advocates:

- Irreversibility: After the biometrics are collected for one purpose, there will be mission creep
- Mistrust of government: Legal protections are insufficient to prevent possible future misuse
- Insufficiency of Cryptographic Protection: Future governments can force the disclosure of keys
- Potential dangers: identifying troublemakers, entrapping innocents, leakage to outside entities

A Standard Biometric Database:

identities



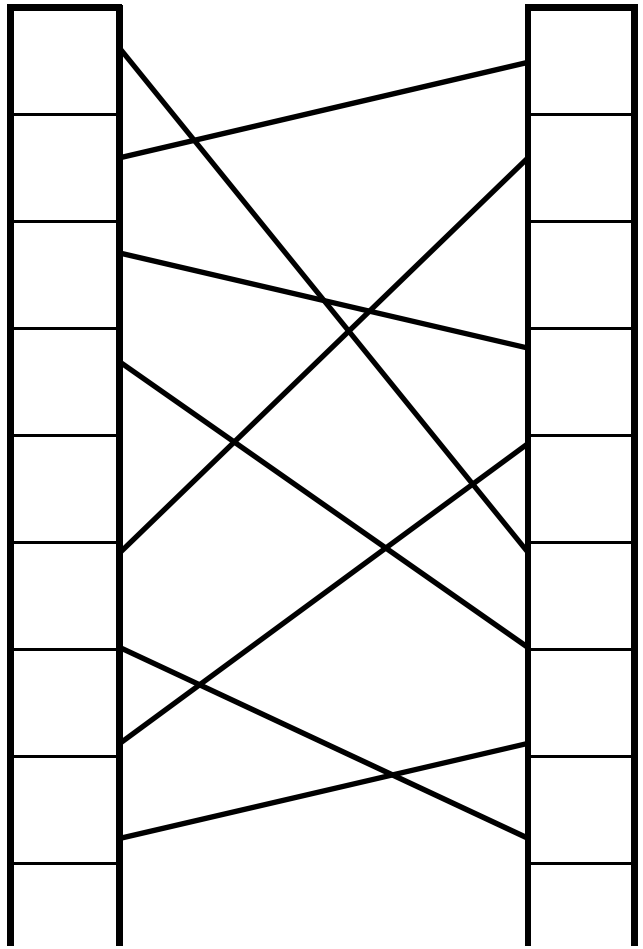


biometrics

A Standard Biometric Database:

a one-to-one correspondence

identities



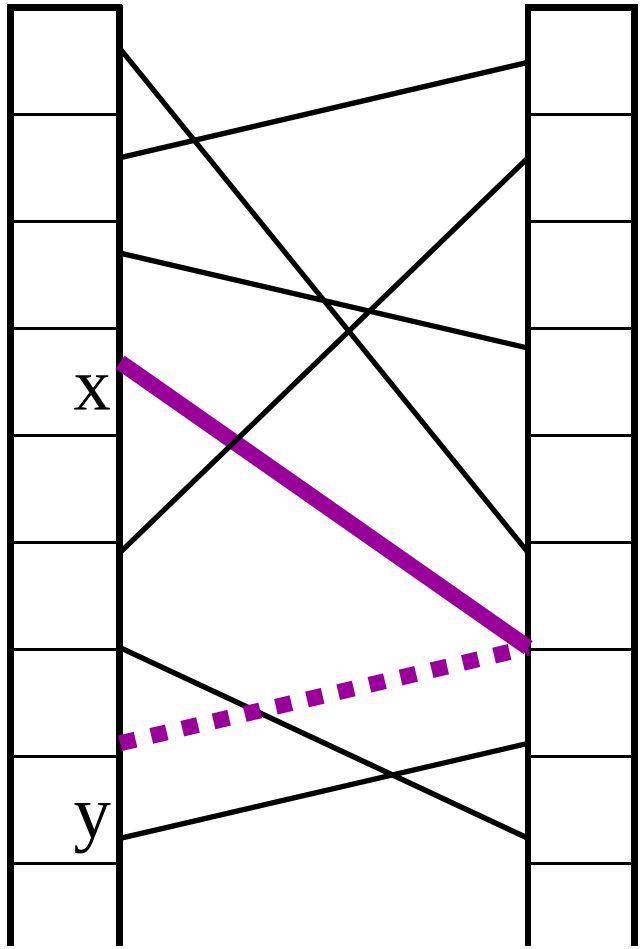
biometrics

A Standard Biometric Database:

a one-to-one correspondence

identities

biometrics



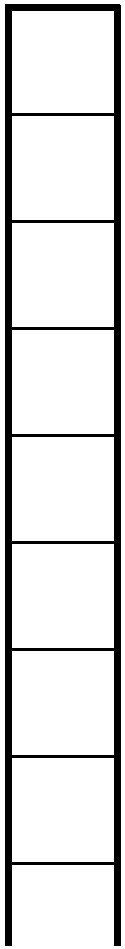
when someone
who is already
registered as Mr X
claims to be Mr Y,
he will be caught
via his biometrics

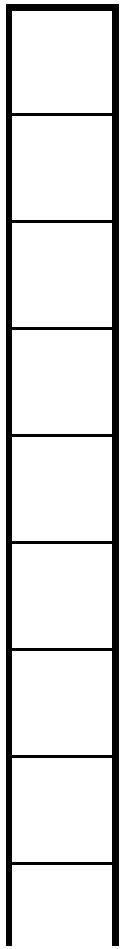
The Main Observation Behind Setbases:

- The database should have:
  - insufficient information to identify a person via his biometrics as Mr X
  - sufficient information to disprove a wrong claim that he is Mr Y
- This separation should remain true even if:
  - the law changes after the database is set up
  - everyone colludes with the government

Using Setbases Instead of Databases:

file cabinet with all the N identities



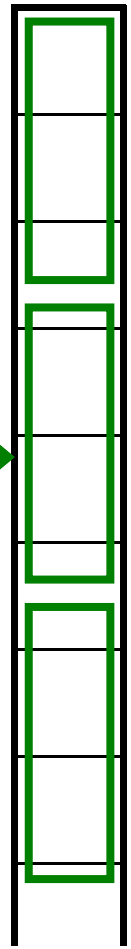


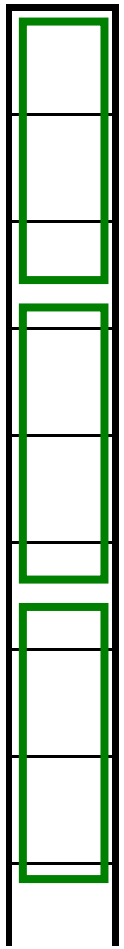
file cabinet with all the N biometrics

Using Setbases Instead of Databases:

file cabinet with all the N identities

secretly and randomly partitioned into drawers with About sqrt(N) files in each



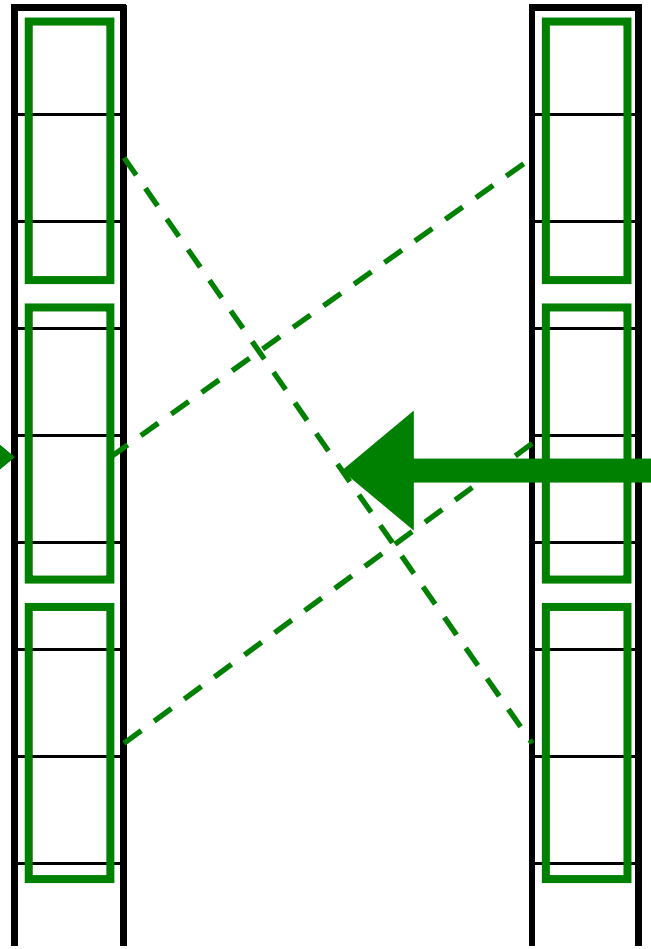


file cabinet with the N biometrics

Using Setbases Instead of Databases:

file cabinet with all the identities

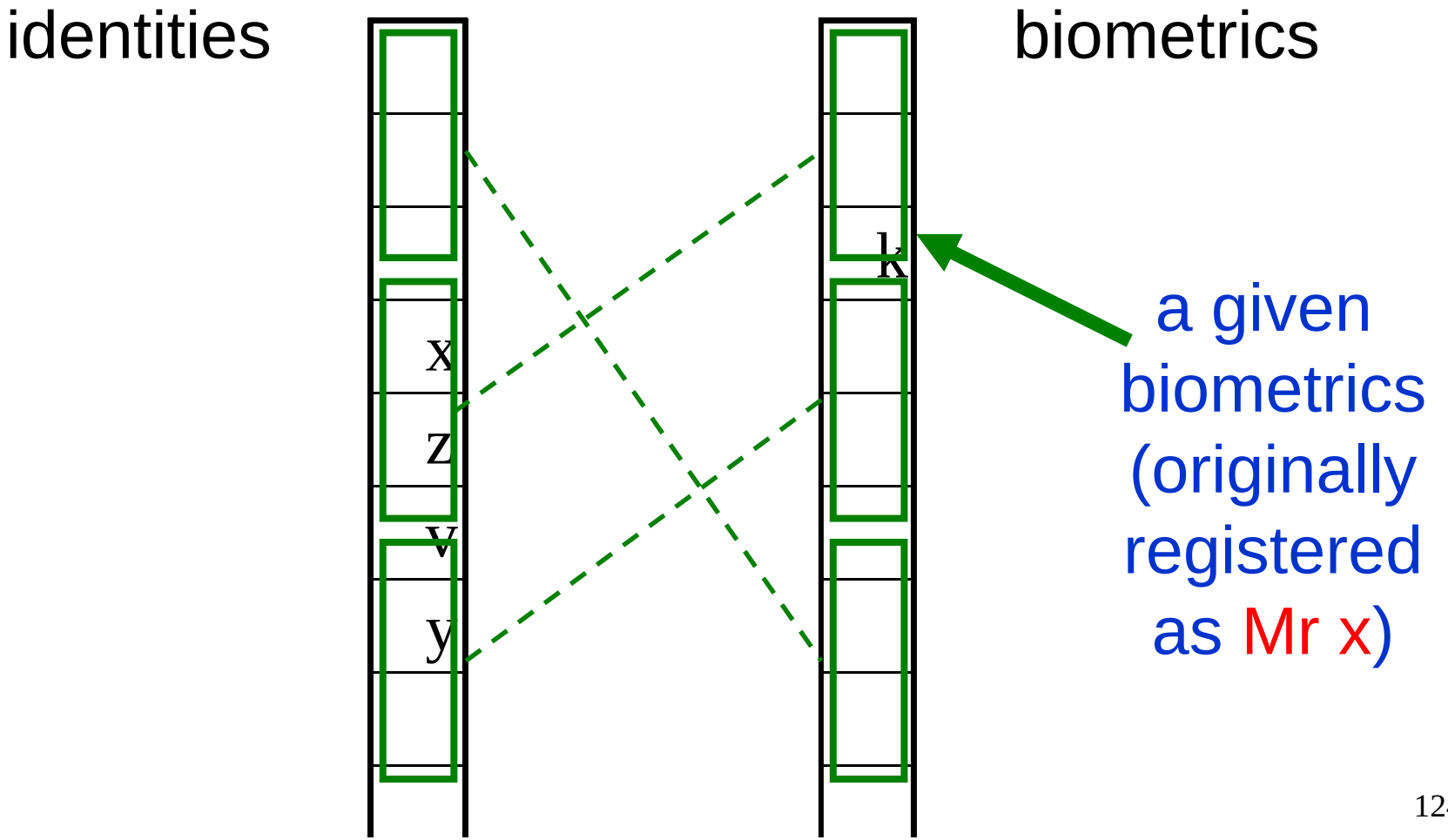
secretly and randomly partitioned into drawers with about 1,000 files in each drawer



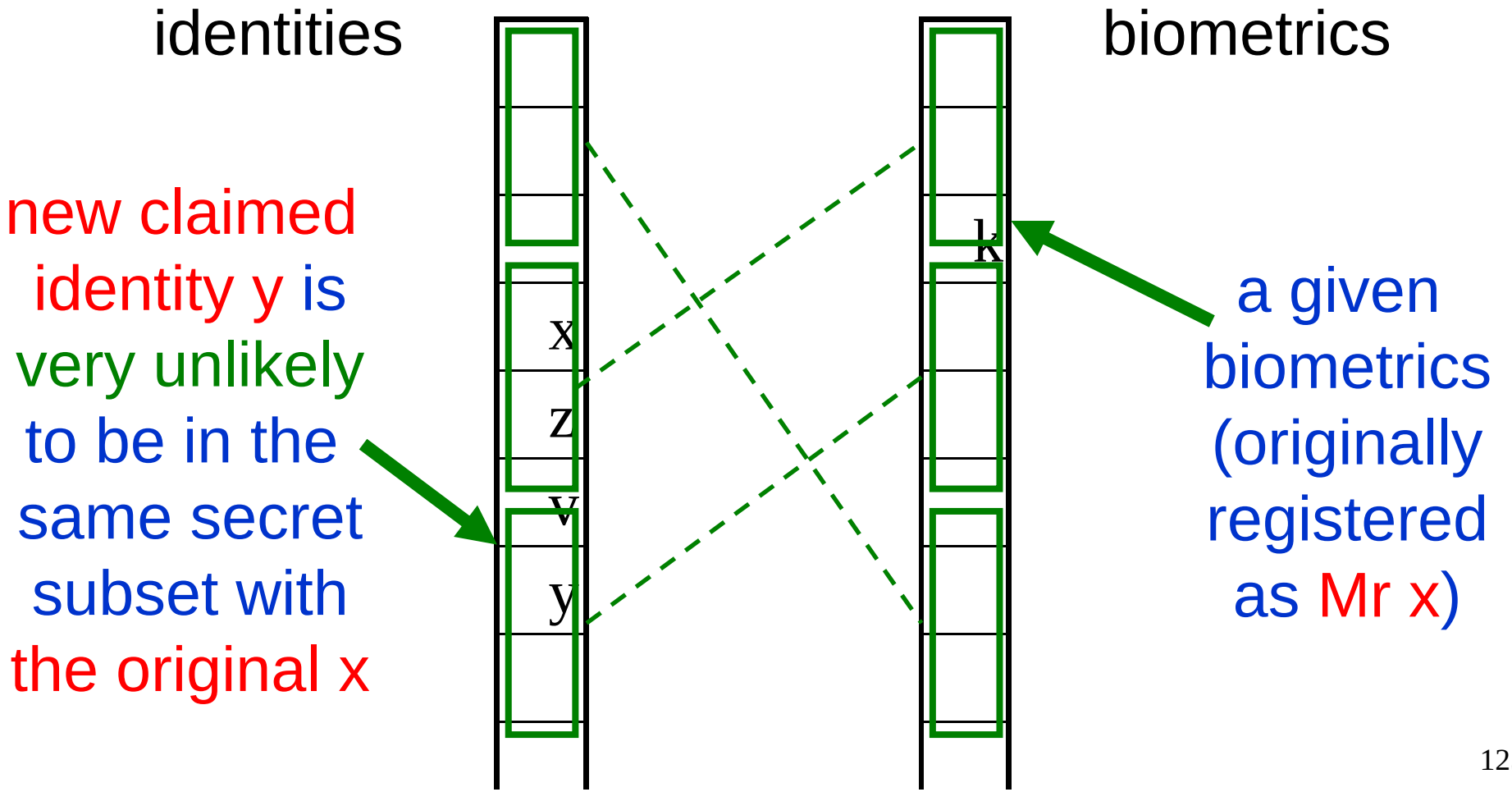
file cabinet with all the biometrics

with secret linking between the drawers, but not between files

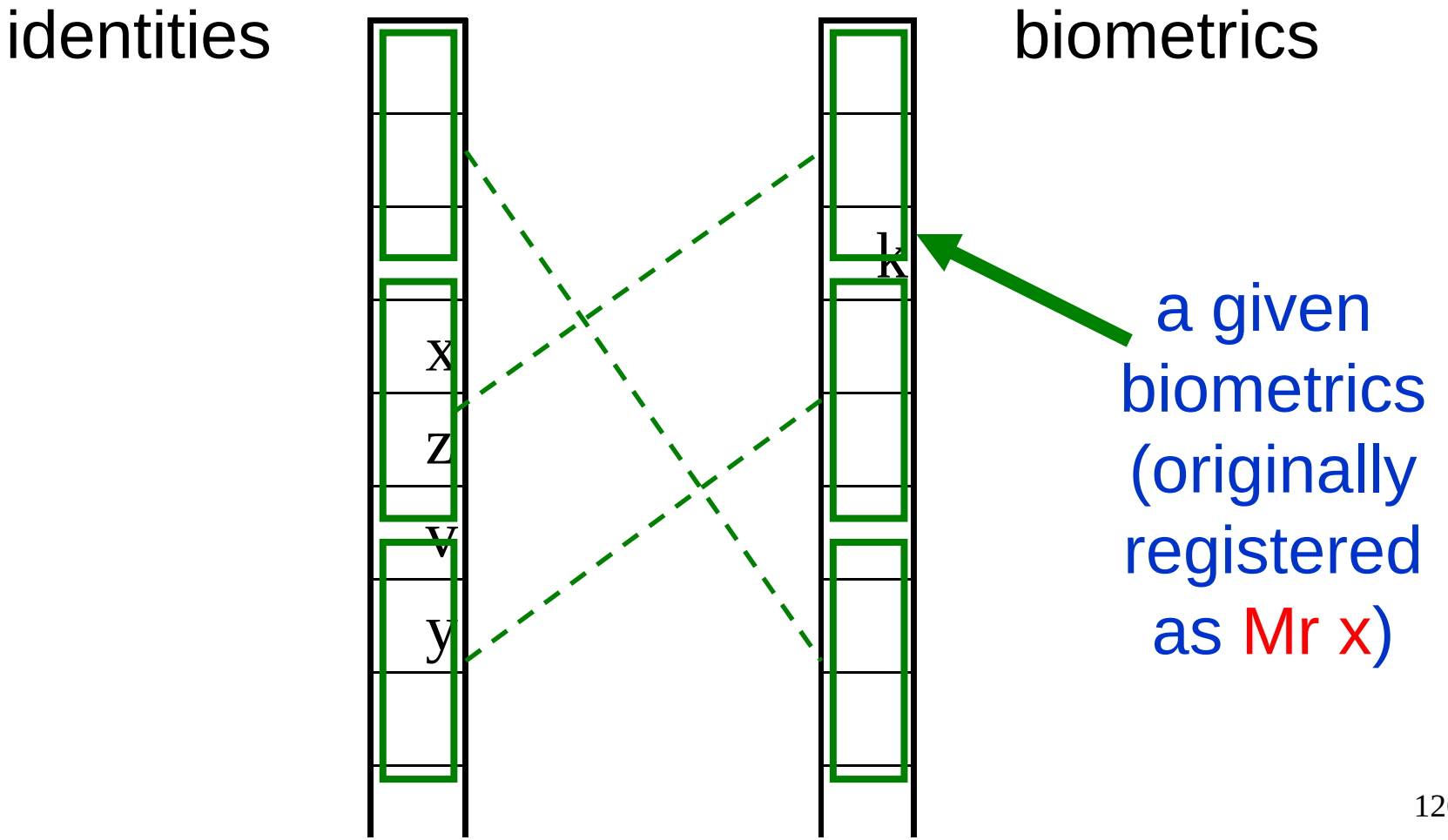
Using Setbases Instead of Databases: How to catch cheaters



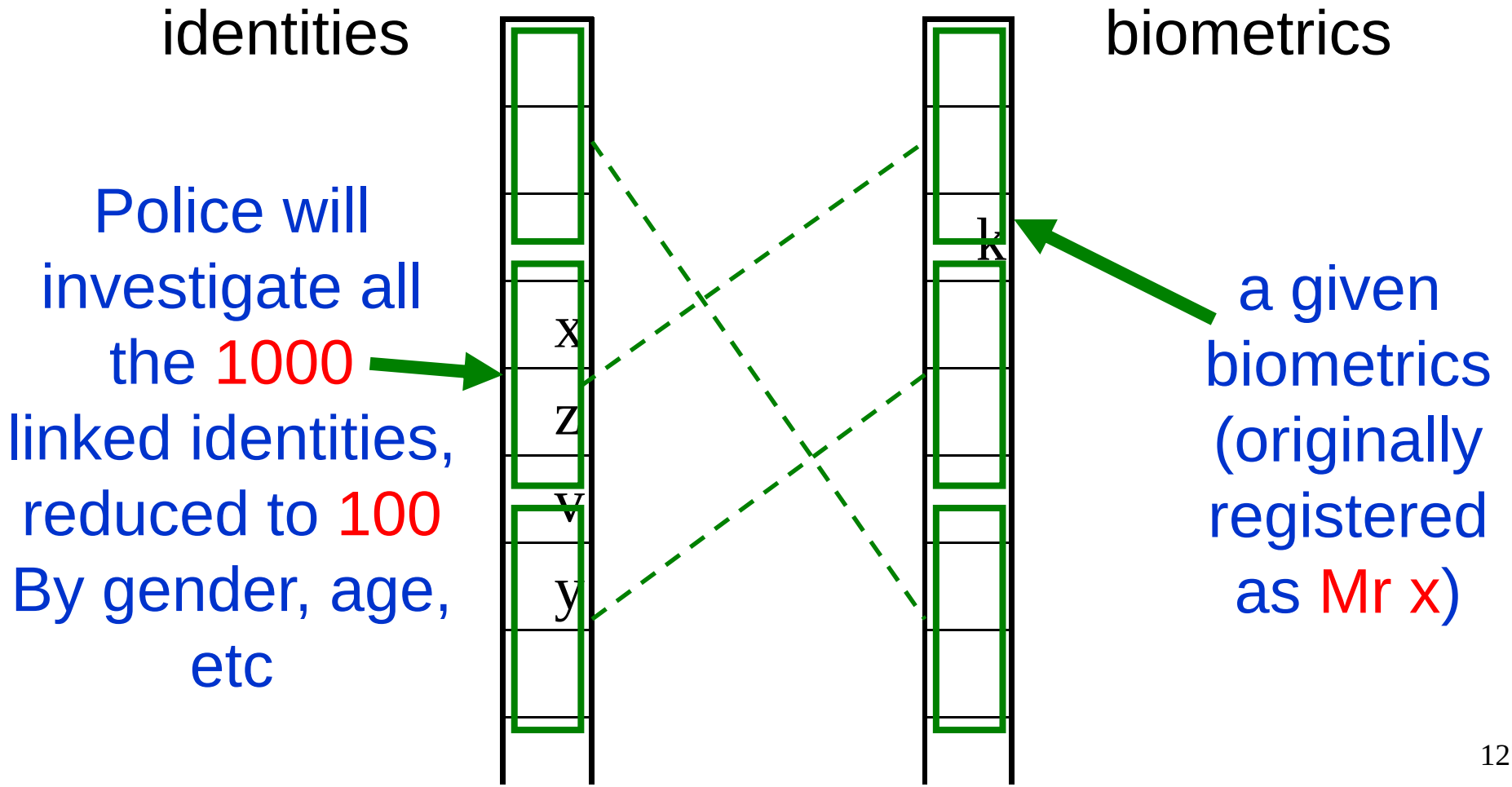
Using Setbases Instead of Databases: How to catch cheaters



Using Setbases Instead of Databases: How to Identify Paperless Bodies



Using Setbases Instead of Databases: How to Identify Paperless Bodies

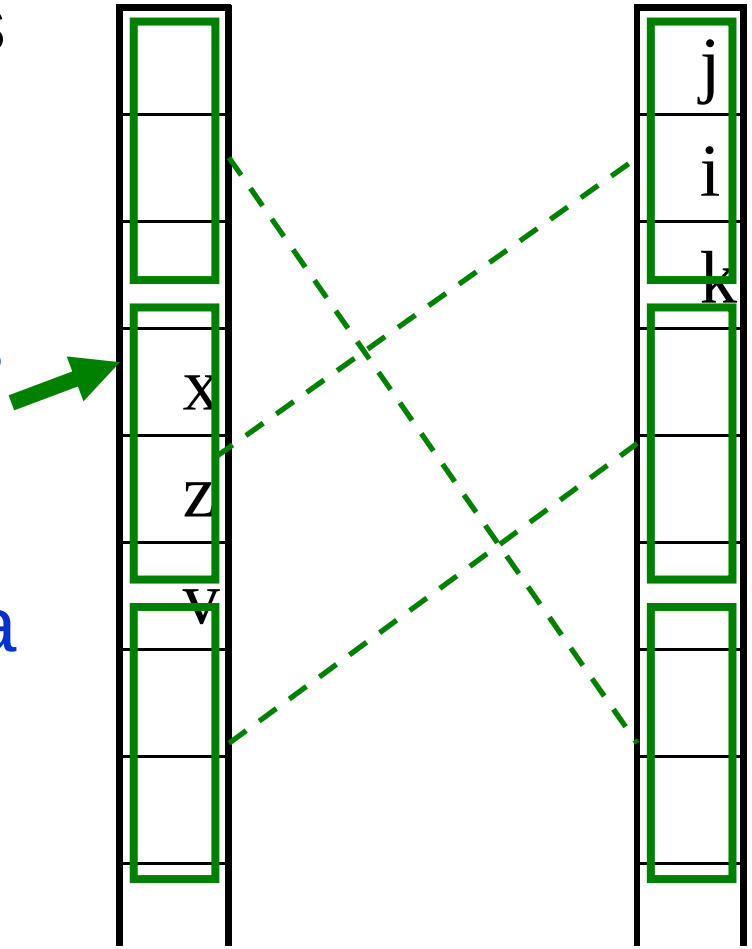


Using Setbases Instead of Databases: Even Fully Leaked Data Cannot Entrap

identities

biometrics

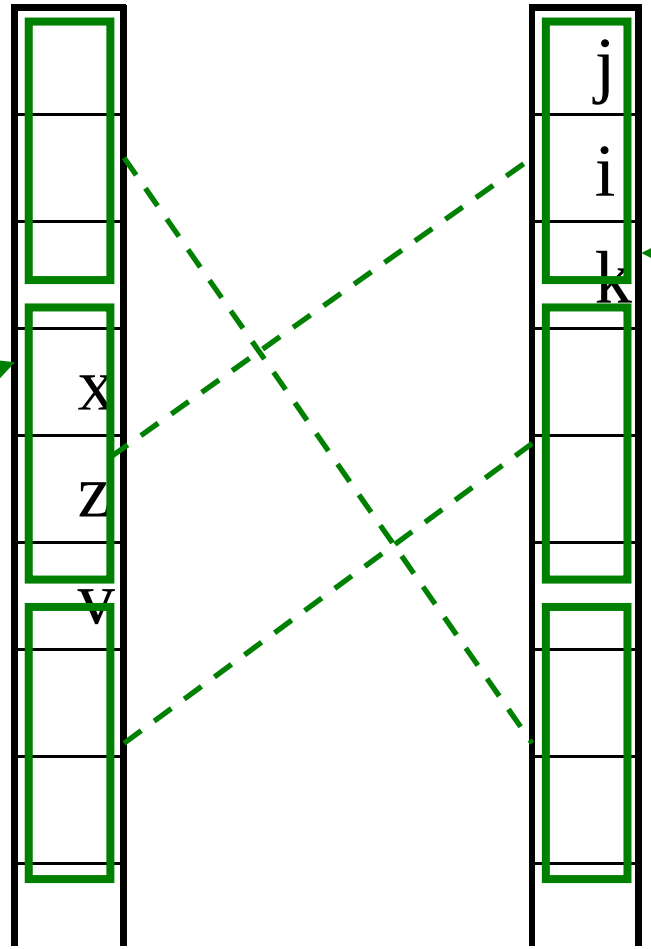
someone with full access to the data wants to entrap x by planting his fingerprints in a crime scene



Using Setbases Instead of Databases: Even Fully Leaked Data Cannot Entrap

identities

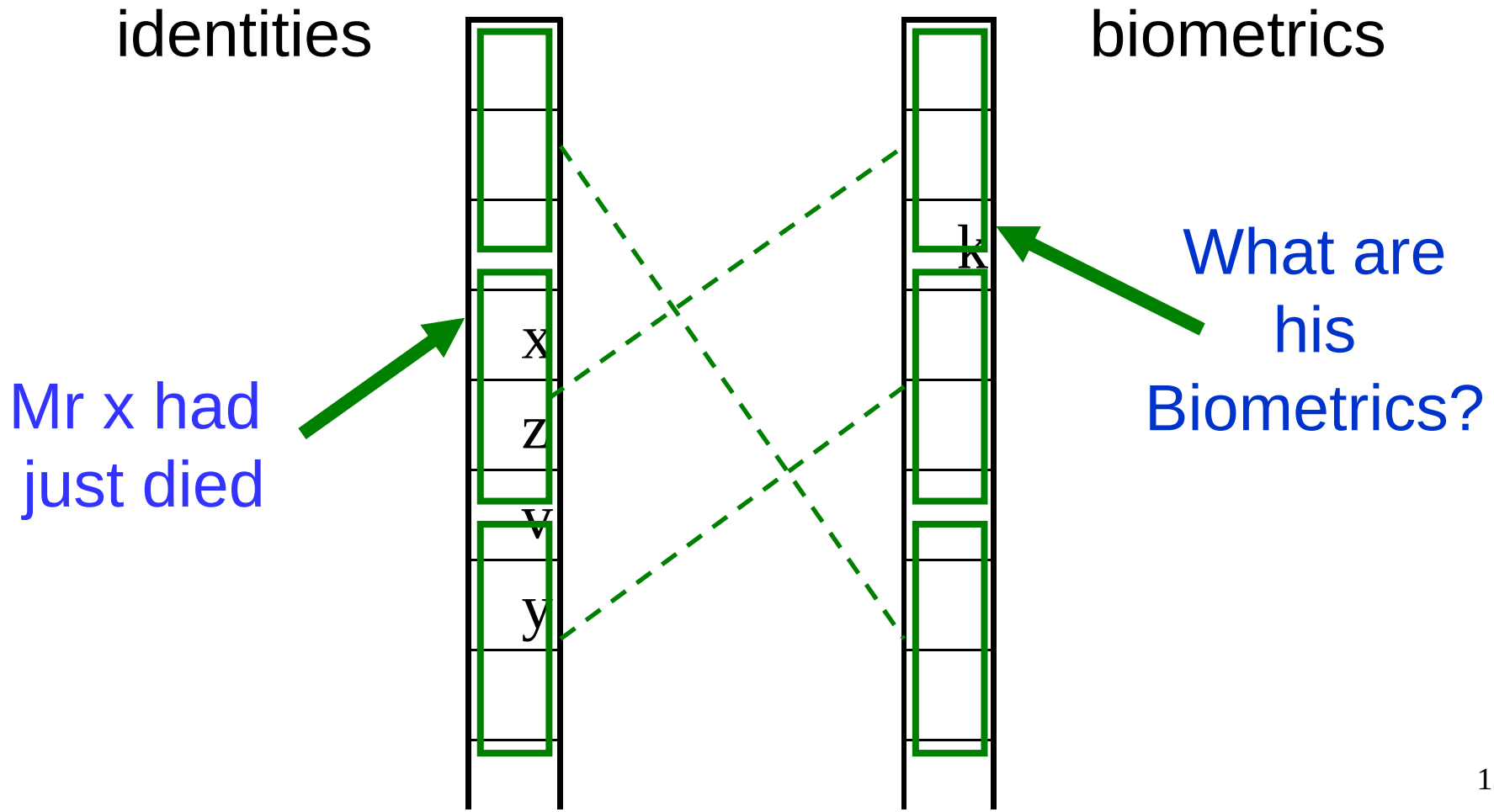
someone with full access to the data wants to entrap x by planting his fingerprints in a crime scene



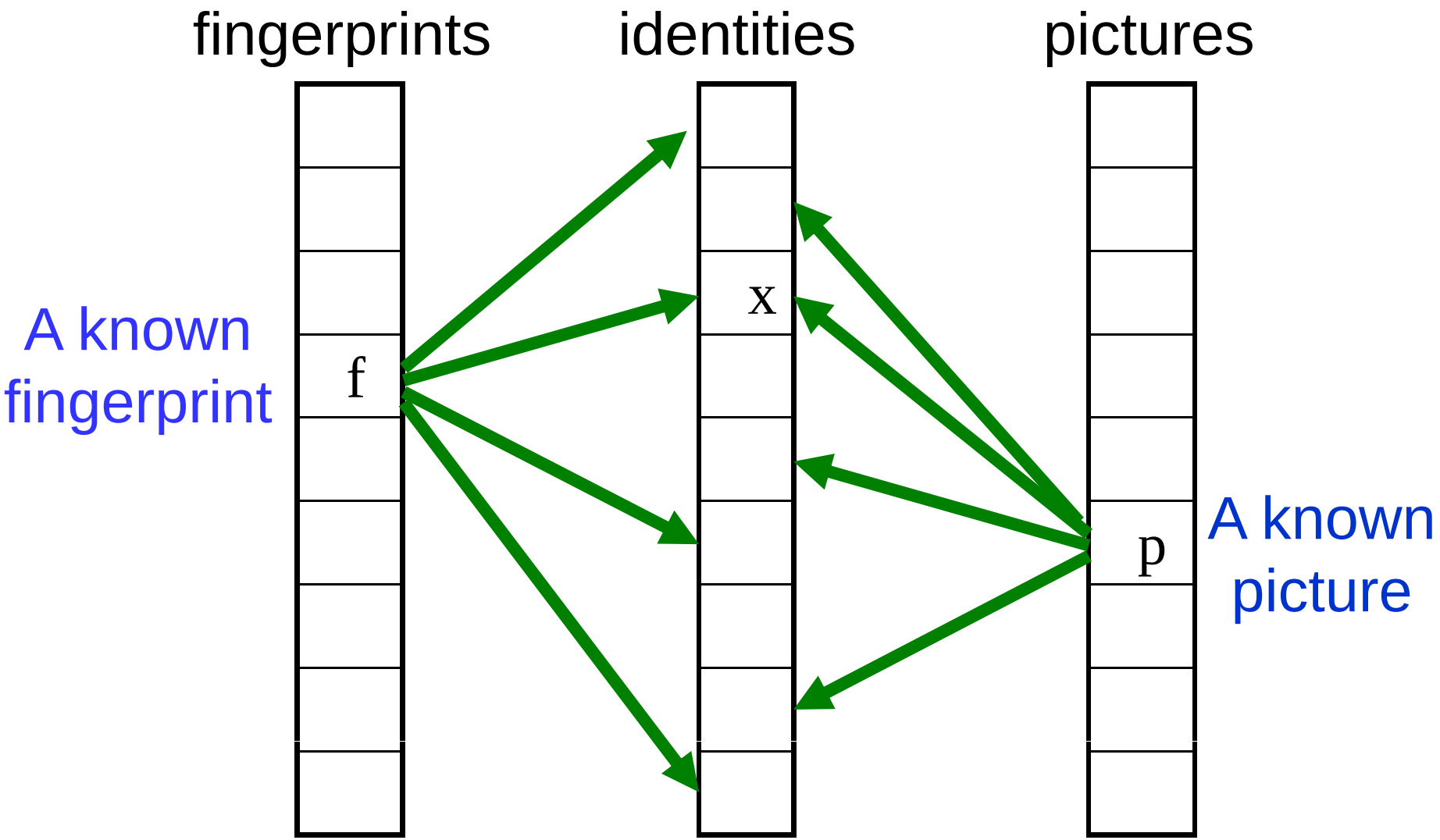
biometrics

planting one fingerprint has probability of 1/1000 to succeed; planting multiple fingerprints will raise alarm

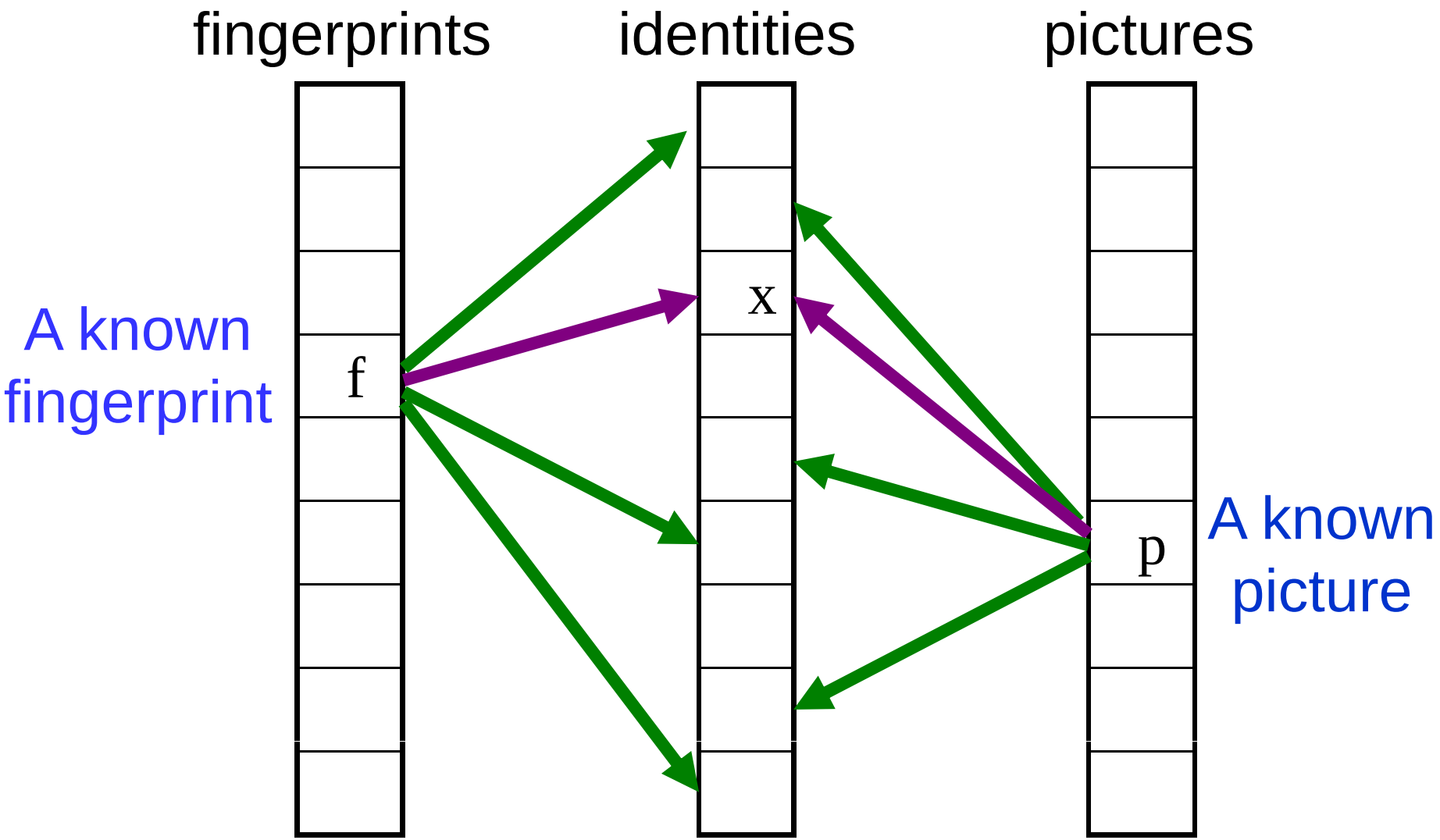
Real life Problems Are More Complicated: Can We Eliminate People who Die or Emigrate?



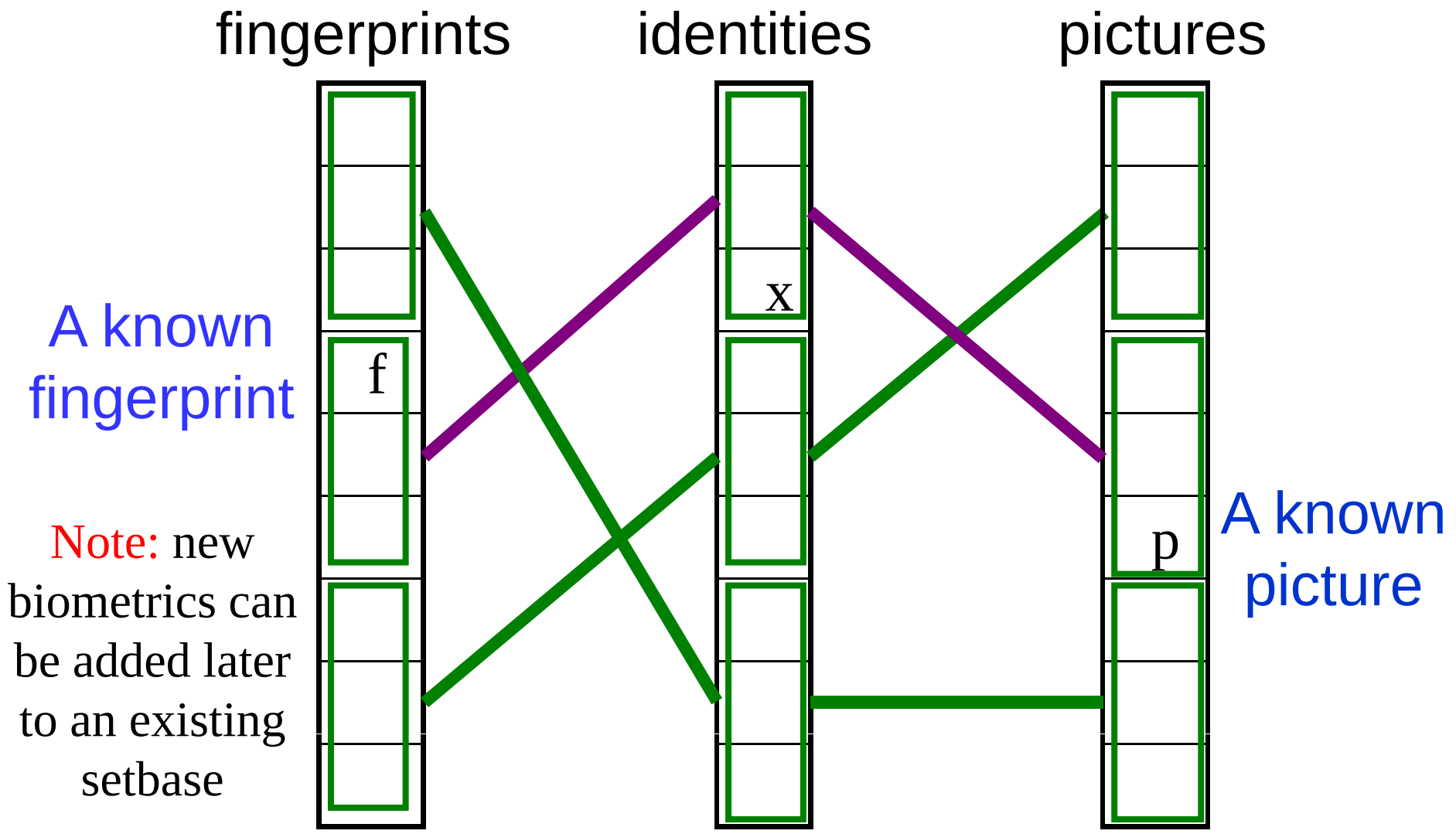
Real life Problems Are More Complicated: How to Deal With Multiple Biometrics?



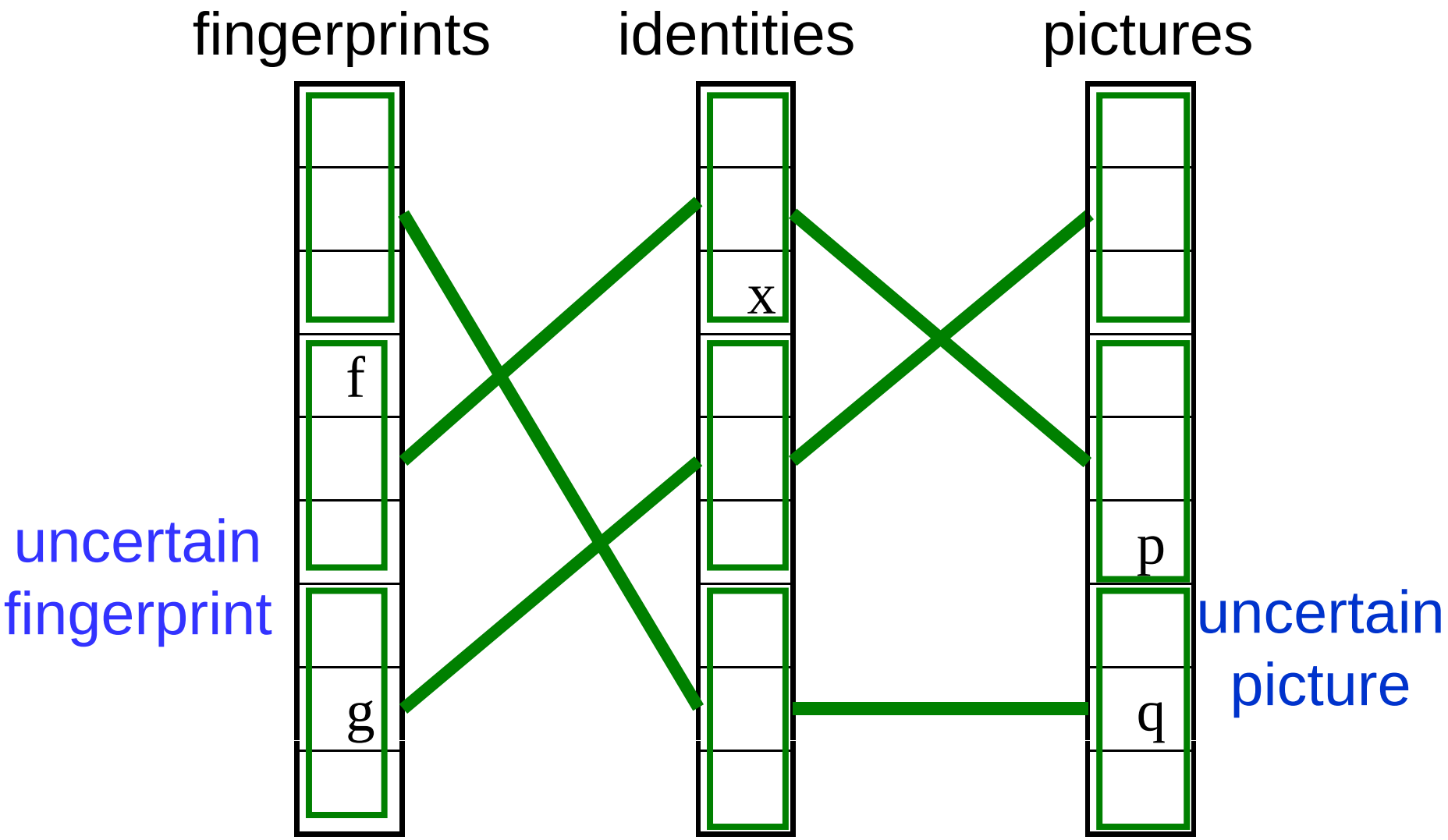
Real life Problems Are More Complicated: Multiple Biometrics Can Identify a Person



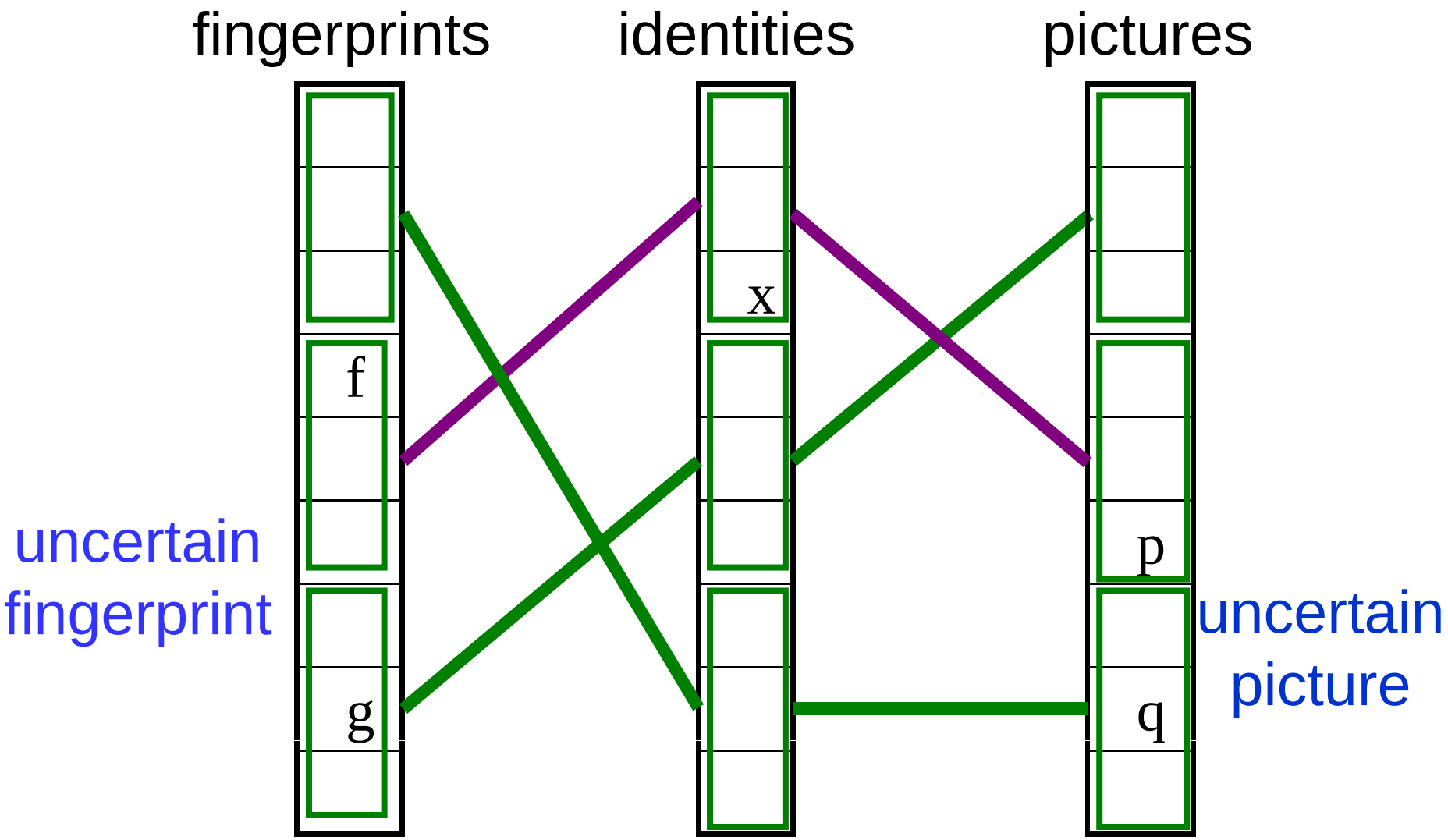
Real life Problems Are More Complicated: Correct Implementation of Hypergraph Setbases



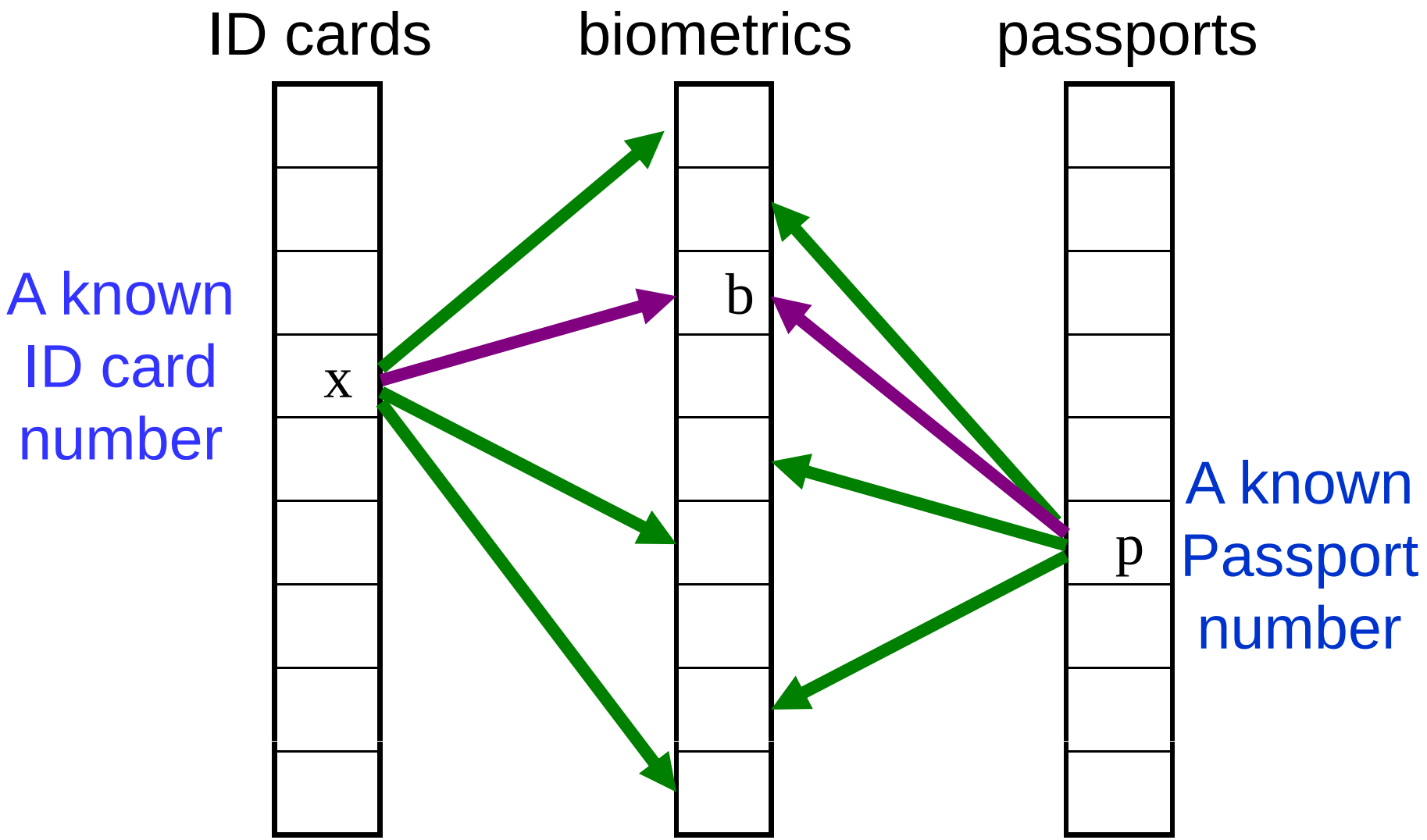
Real life Problems Are More Complicated: The Advantages of Hypergraph Setbases



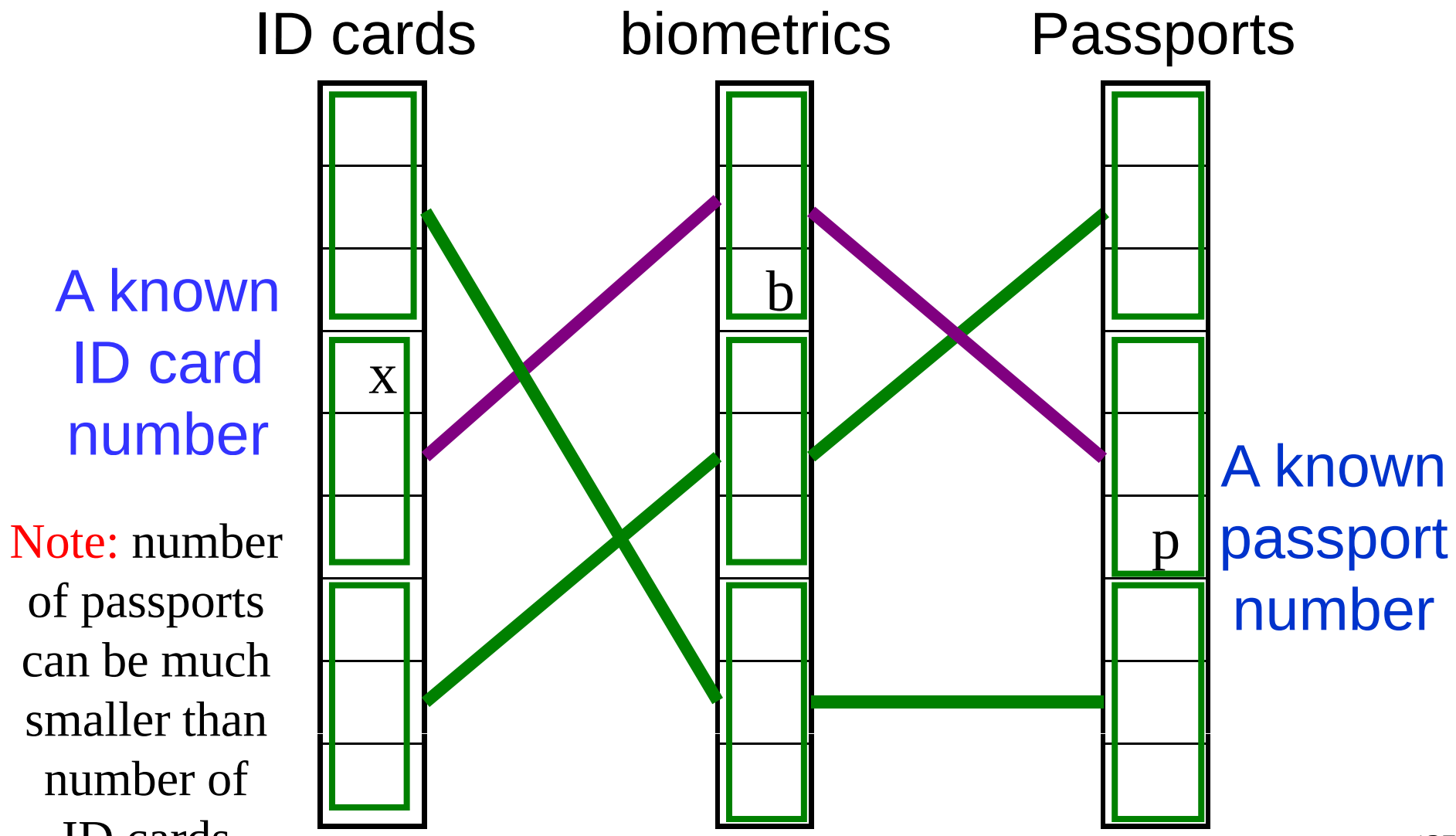
Real life Problems Are More Complicated: The Advantages of Hypergraph Setbases



Real life Problems Are More Complicated: The Dual Problem of Multiple Card Types



Real life Problems Are More Complicated: The Dual Problem of Multiple Card Types



Summary of Setbases:

- Like any other privacy enhancing technique, setbases are a compromise between the conflicting demands for privacy and functionality
- Double issuing can be prevented at almost no additional cost and with very high probability
- Individuals can be identified from their biometrics, but only by a long, expensive and highly visible police investigation, and can't be easily entrapped
- This privacy protection cannot be eliminated by changing the law or expropriating the crypto keys

Conclusion:

Random graphs are wonderful objects to study

Understanding their structure can lead to many cryptographic and cryptanalytic optimizations, as well as to new privacy enhancing techniques

In this talk I gave only a small sample of the published and folklore results at the interface between cryptography and random graph theory


Thank You