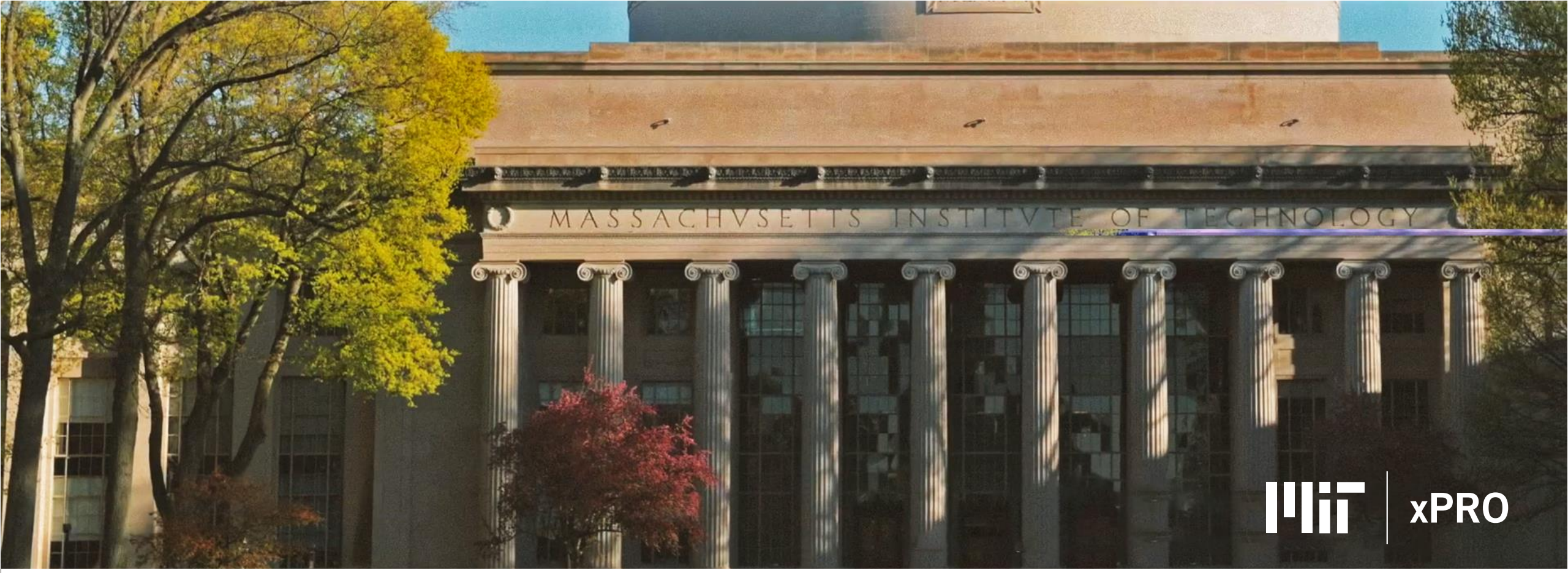
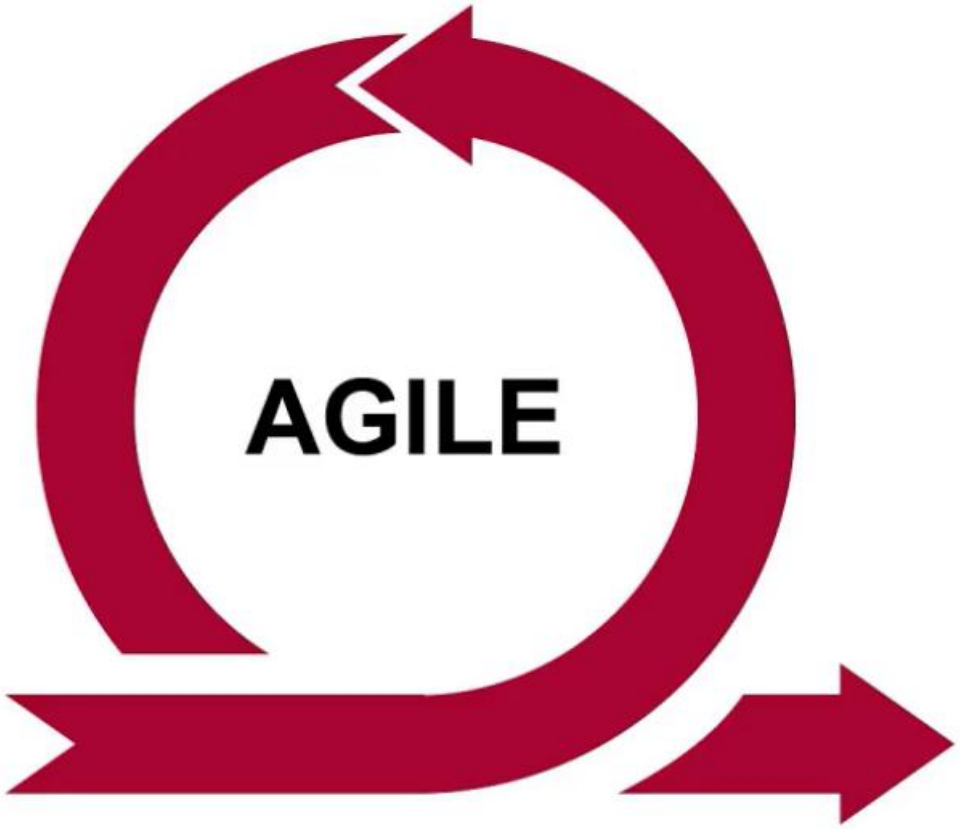


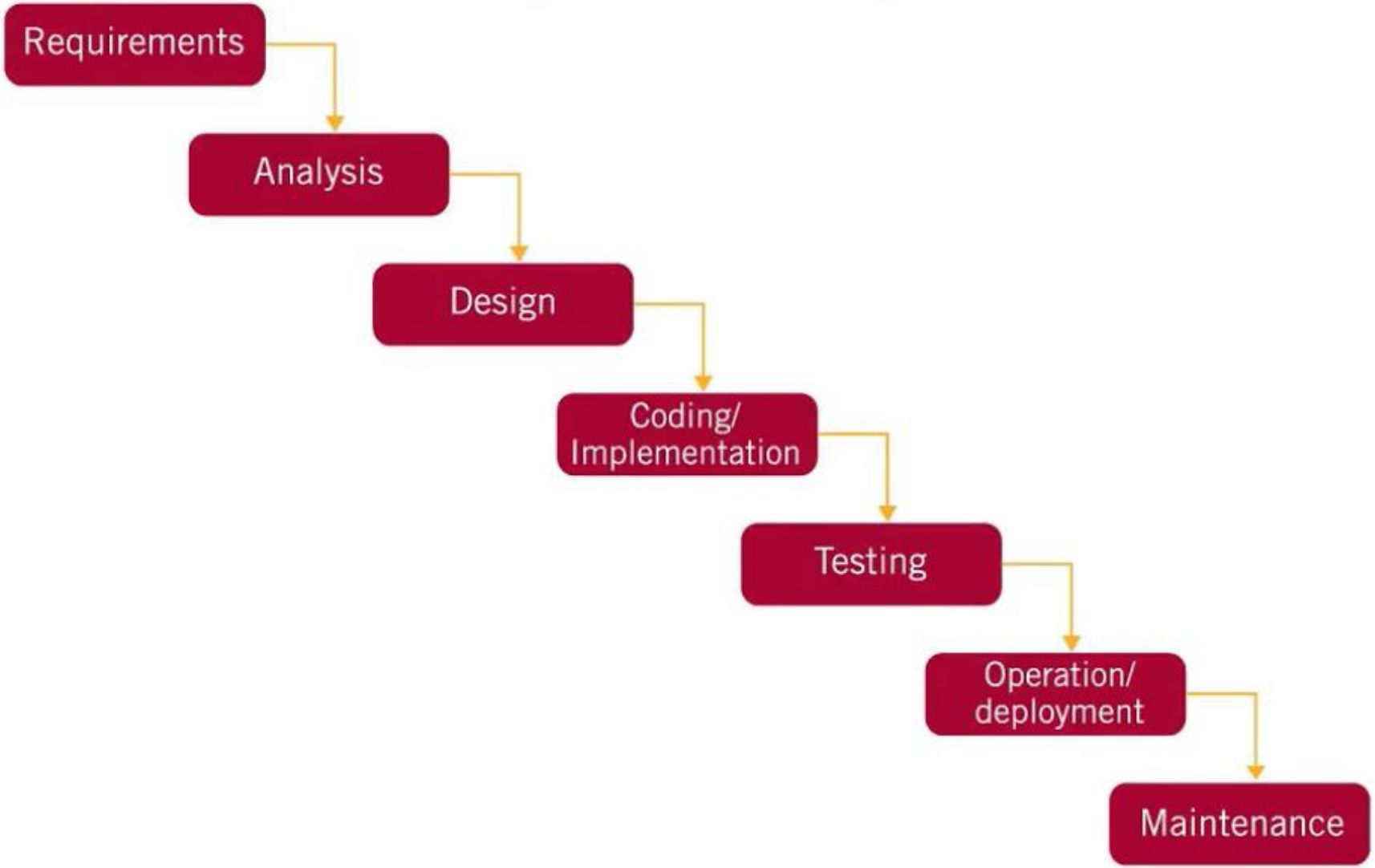
Professional Certificate in Coding: Full Stack Development with MERN: Week 10

Introduction To Cybersecurity And Recursion

Agile Methodology



Waterfall Model



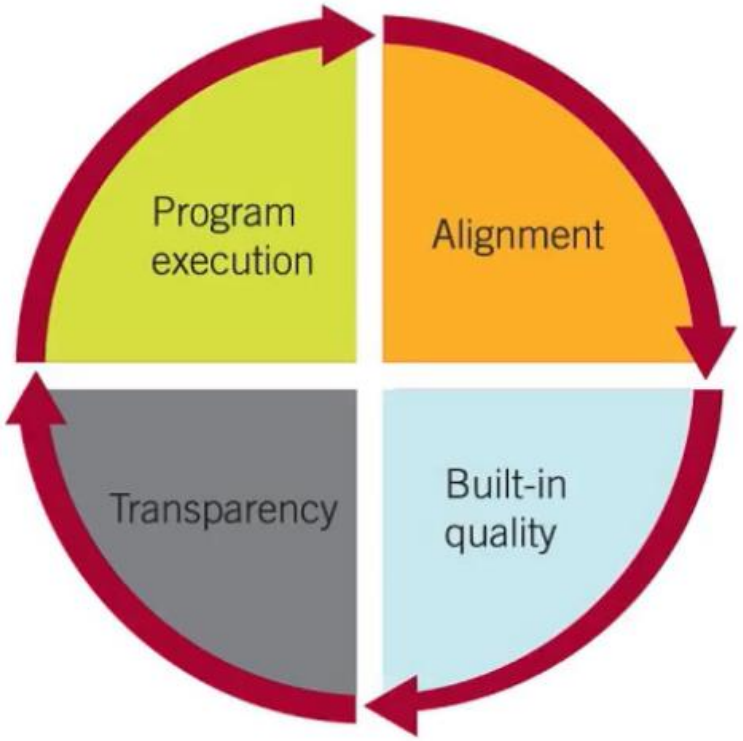
Agile Principles

1. Satisfy the customer through continuous delivery
2. Welcome changing requirements
3. Deliver working software
4. Collaborate with business leadership
5. Built projects around motivated individuals
6. Face to face conversation
7. Measure progress by working software
8. Maintain a constant development pace
9. Focus on technical excellence and good design
10. Keep it simple with clear scope
11. Self organizing teams produce the best software
12. Reflection improves team performance

Agile Values

Agile Values

1. Individual and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan



Language Of Agile

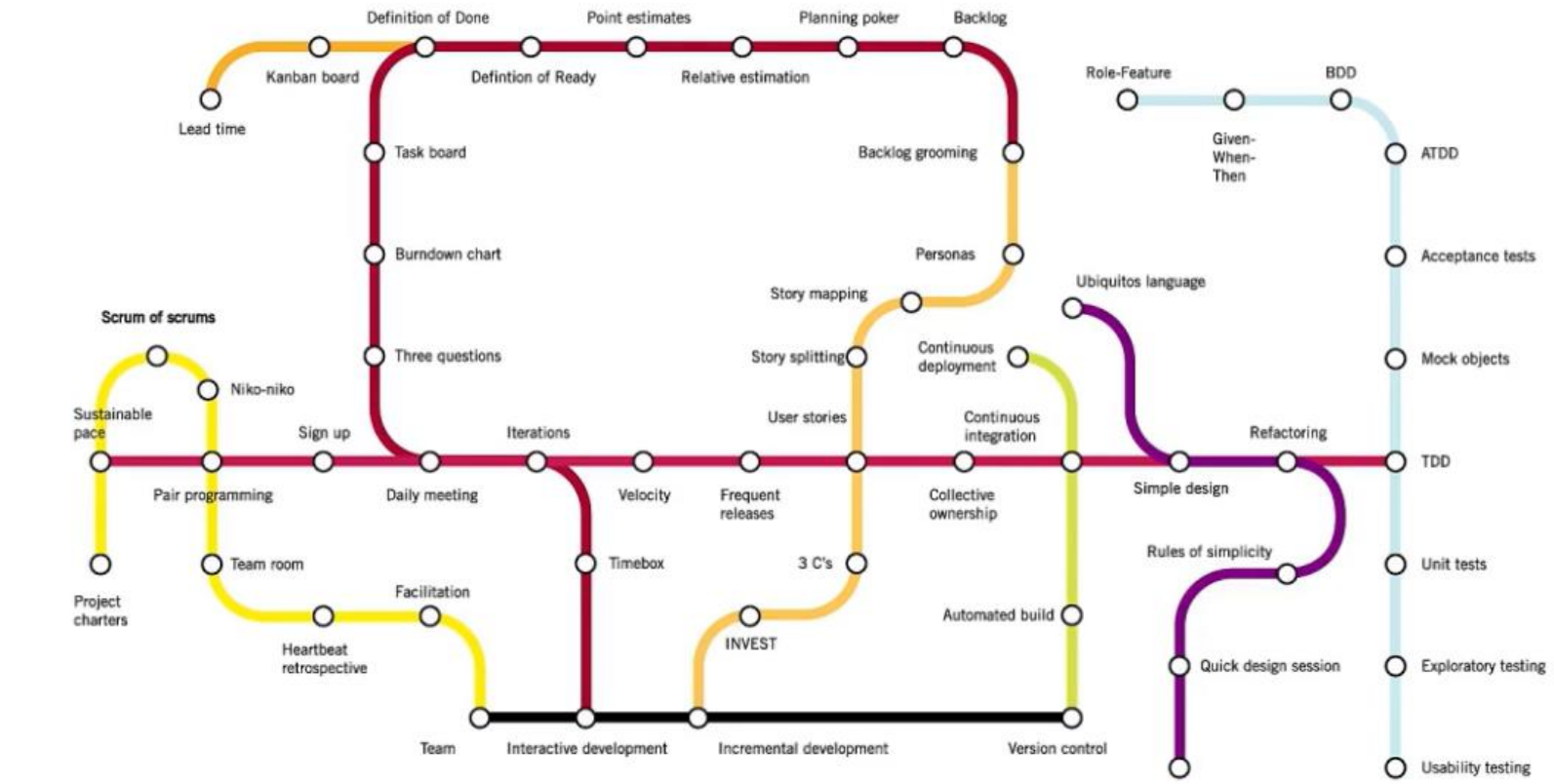
- User Stories
  - Add card, Conversation, Confirmation
- Define some problem in the real world e.g. booking an airplane ticket
  - Upselling, cross-selling (booking hotel, car rental, ...)
  - Updating flight times, seats ...
  - Cancelling, refunds, invoice receipts, credits, air miles, ...

Scrum


Review
Test
Build
Design
Plan
Sprint 1
Launch
Review
Test
Build
Design
Plan
Sprint 2
Launch
Review
Test
Build
Design
Plan
Sprint 3
Launch

Project Tradeoffs

Various Agile “Tribes”



Authentication And Authorization (1/2)

Authentication:
who are you?

Authentication And Authorization (2/2)

Authorization: what are you allowed to access?

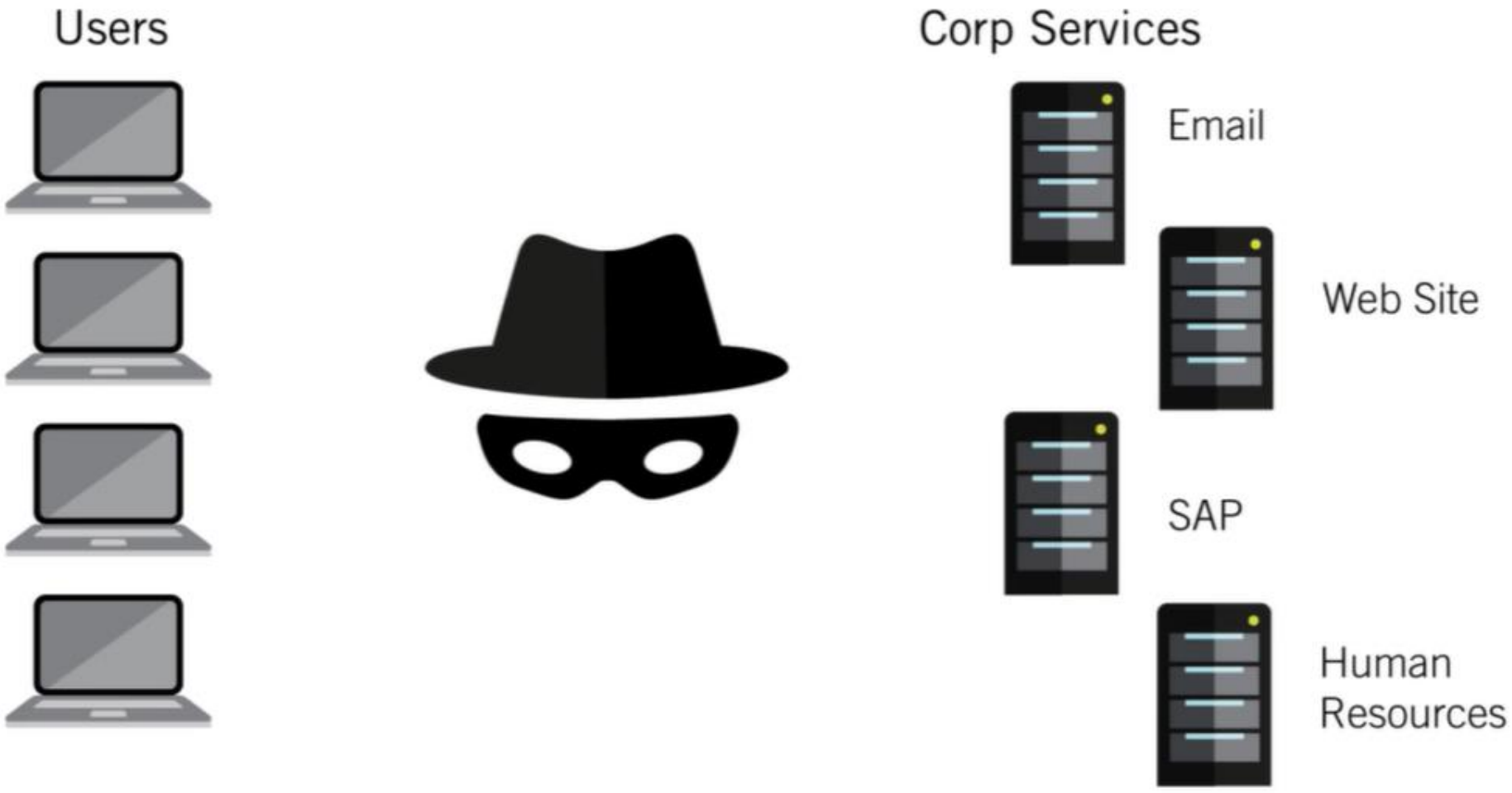
Symmetric Encryption And Notation

Key = 12345

Key {Hello World} → c935af

Key {c935af} → Hello World

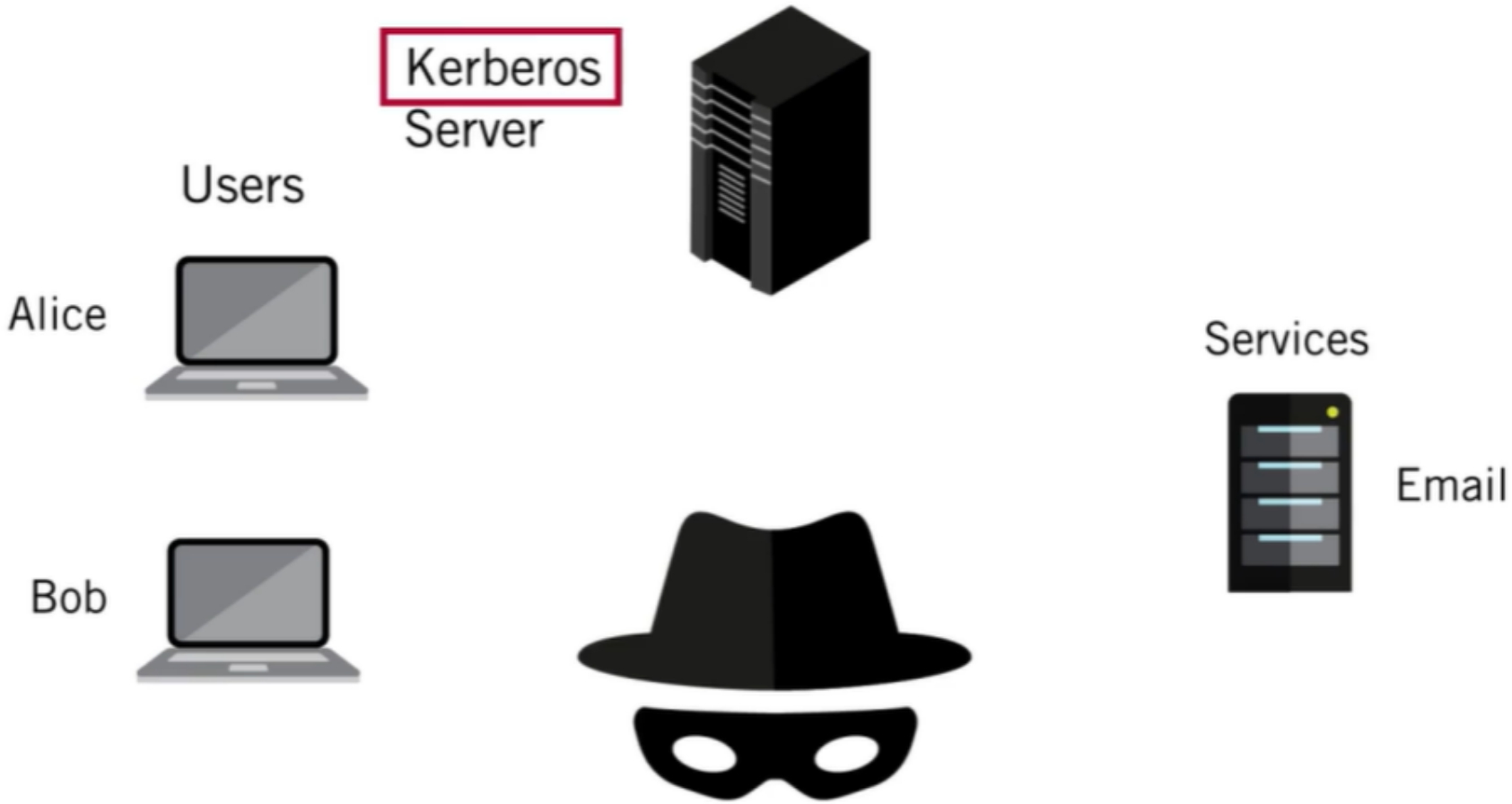
Multiple Users That Needs To Use Number Of Services



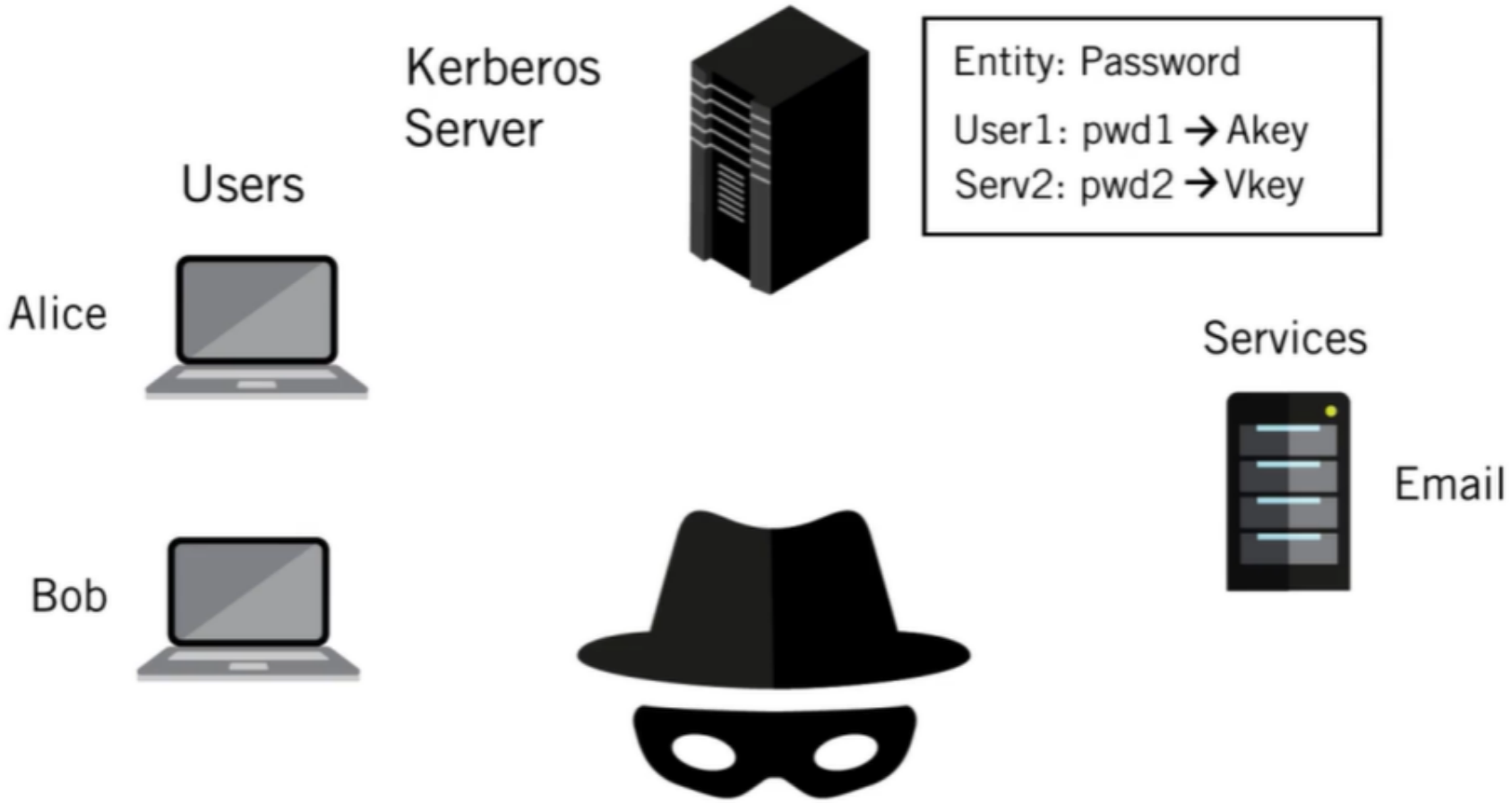
Assumptions

- We do not trust the network to be secure.
- Attackers can see all messages and can copy and replay any message
- We would like single sing on (SSO) so users do not have to enter password multiple times
- Password should never be stored on the client machine
- Password should never be passed in cleartext on network

Concept Of An Authentication Service That Users And Services Trust (1/2)



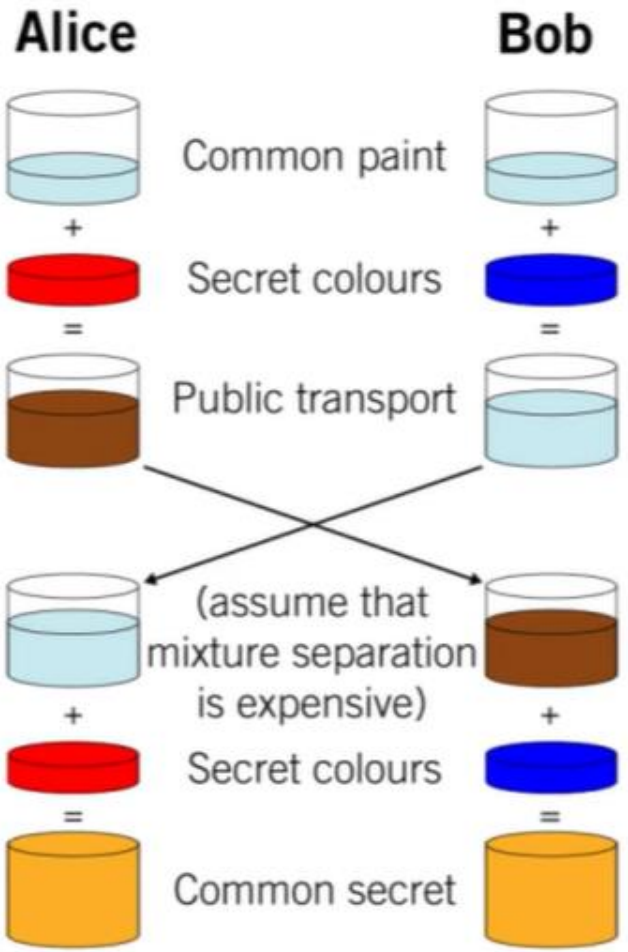
Concept Of An Authentication Service That Users And Services Trust (2/2)



Shared Secret Idea

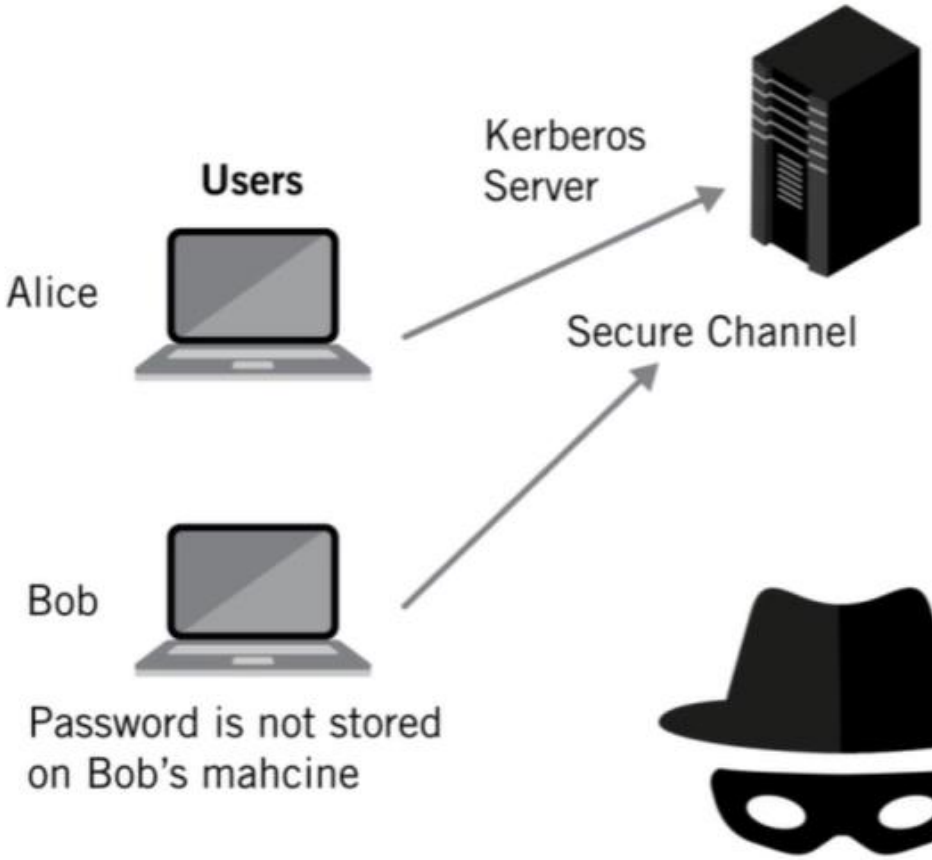
- User and Kerberos share a secret
- They use the secret to encrypt messages
- Kerberos knows a message is from Alice if Alice's secret decrypts the message
- Alice knows it's a legitimate Kerberos message if her secret decrypts the message
- Encrypted messages can be "copied" by attacker but cannot be decrypted

Diffie-Hellman Key Exchange



Secure Channel To Set And Store The Password

A Secure Channel is used to initially set the password and store in Kerberos database eg Diffie-Hellman



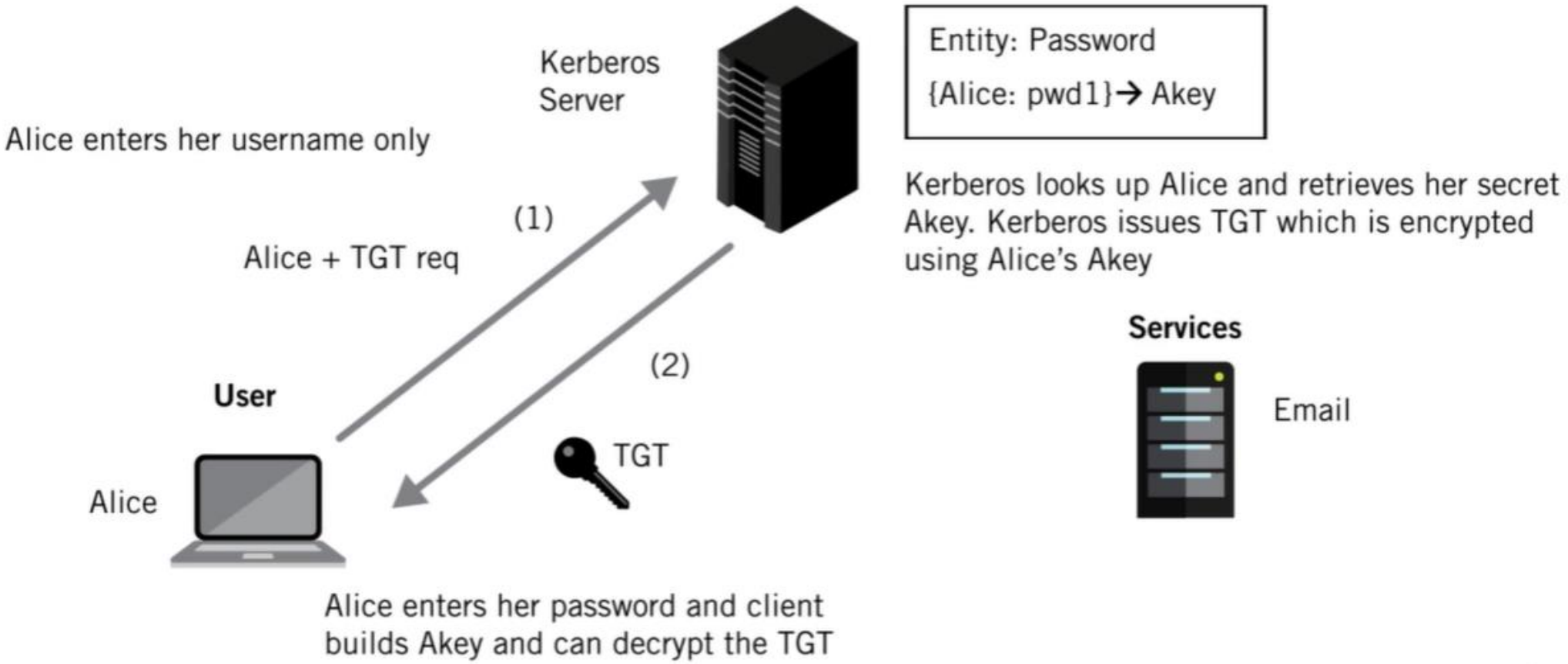
Entity: Password
User1: pwd1 → Akey
Serv2: pwd2 → Vkey

Username + password are combined to form a secret key, Akey etc. These keys are used to encrypt messages
Kerberos uses a Master Key to encrypt this database

Services

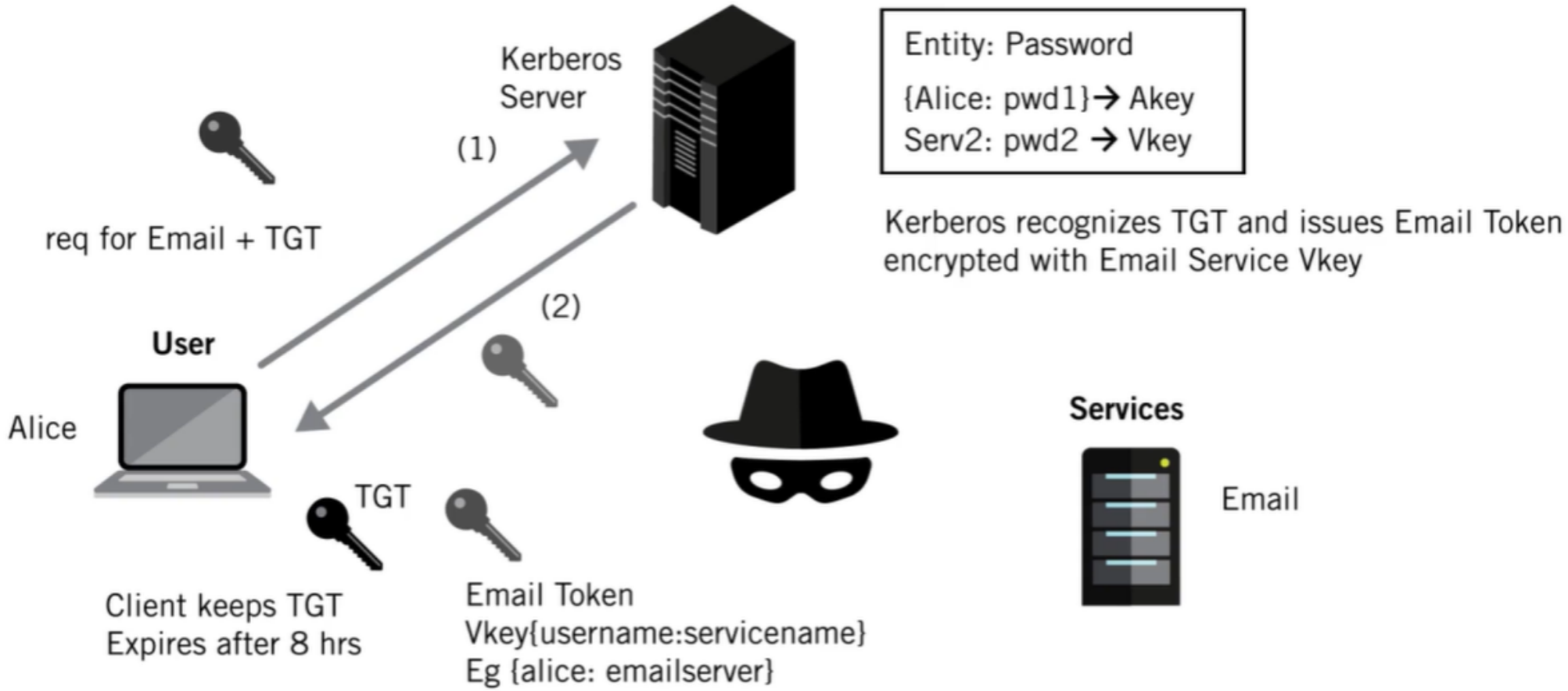
Email

Alice Gets A Ticket-Granting-Ticket (TGT)

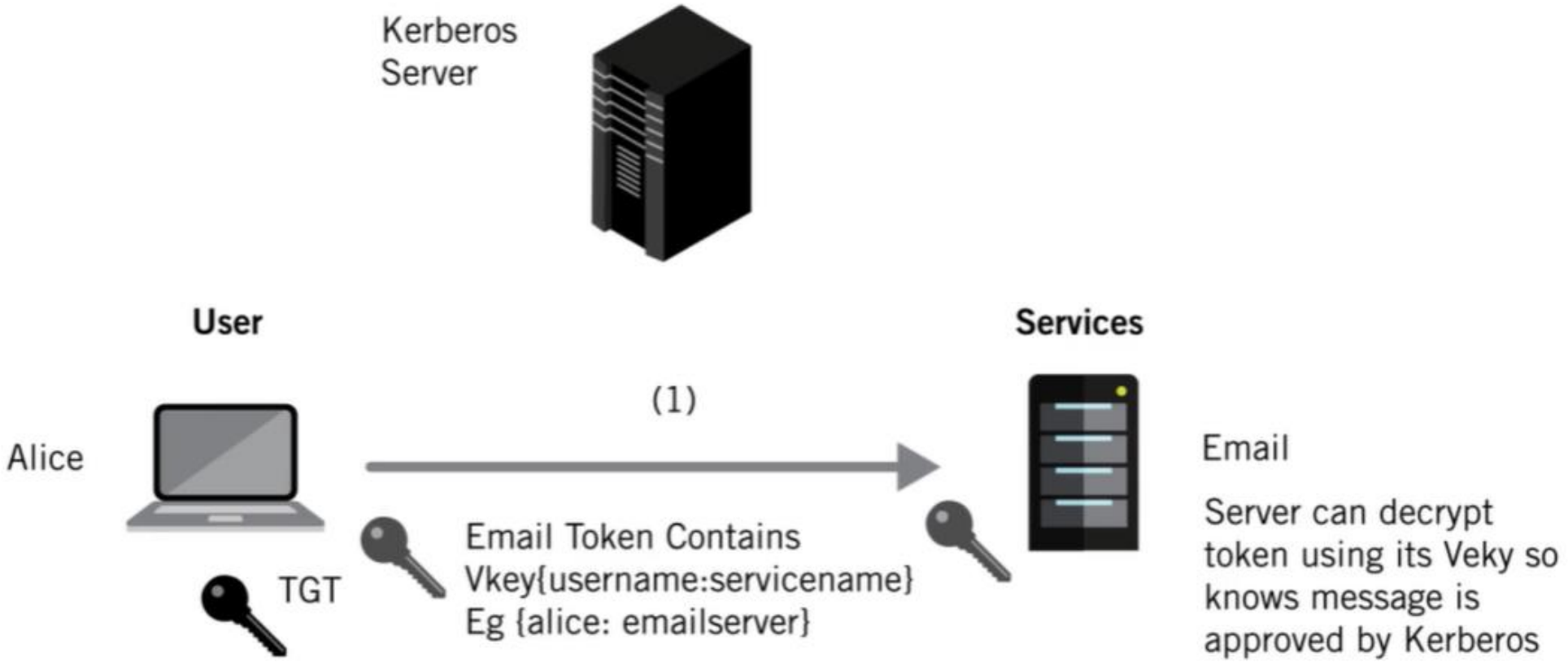


Alice Requests Access To Email

TGT is now used rather than Password. Alice requests access to Email

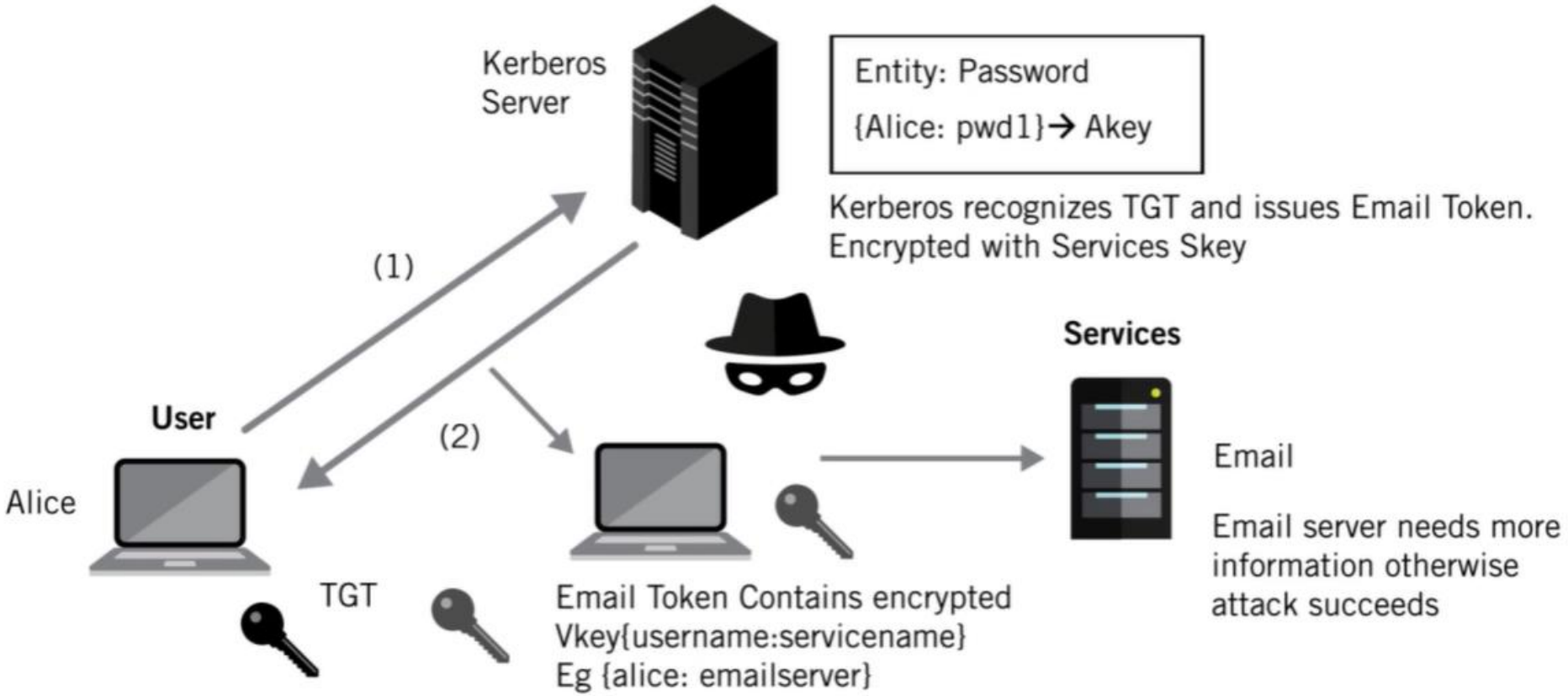


Alice Sends Email Token To Server



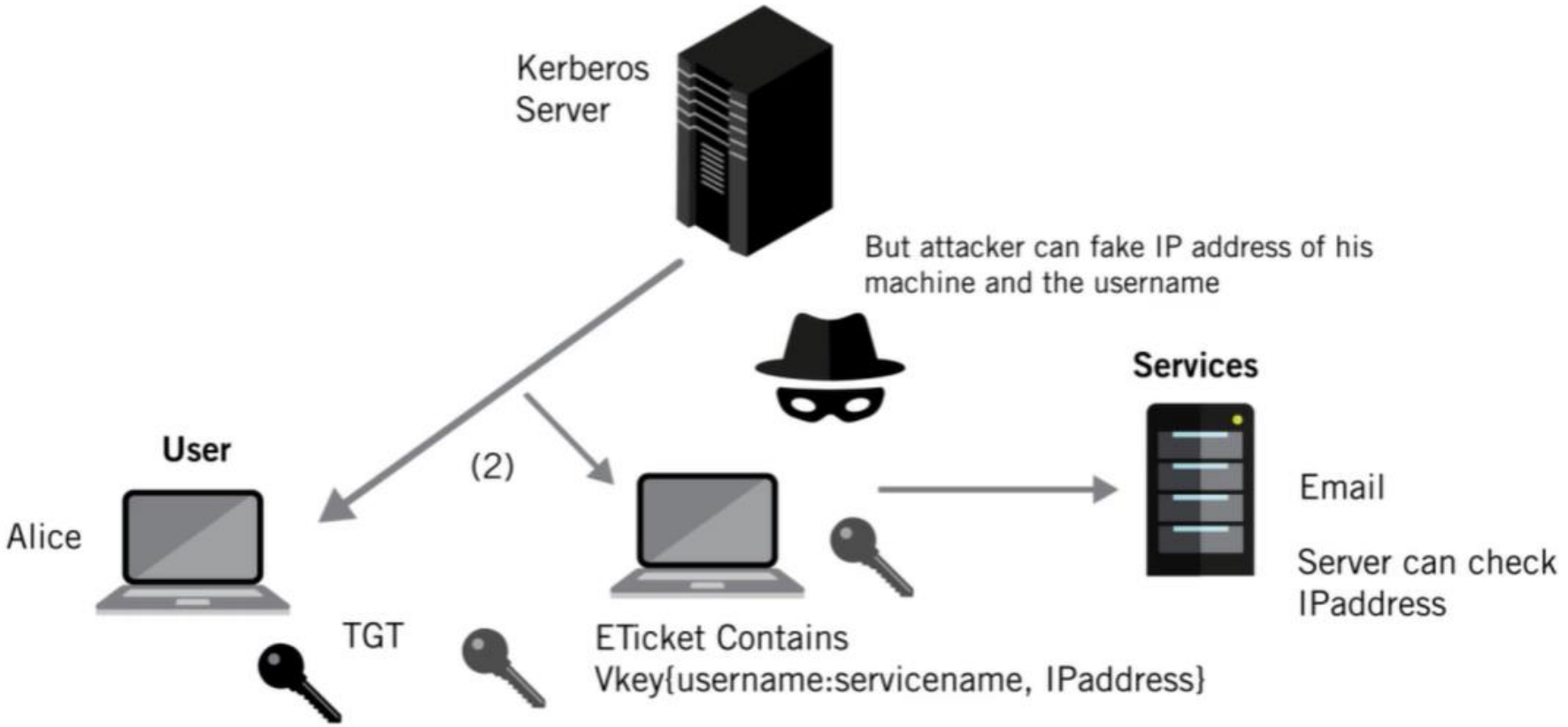
What If The Attacker Steals The Encrypted Message?

Suppose Attacker copies Email Token then sends it on to Email Server pretending to be Alice

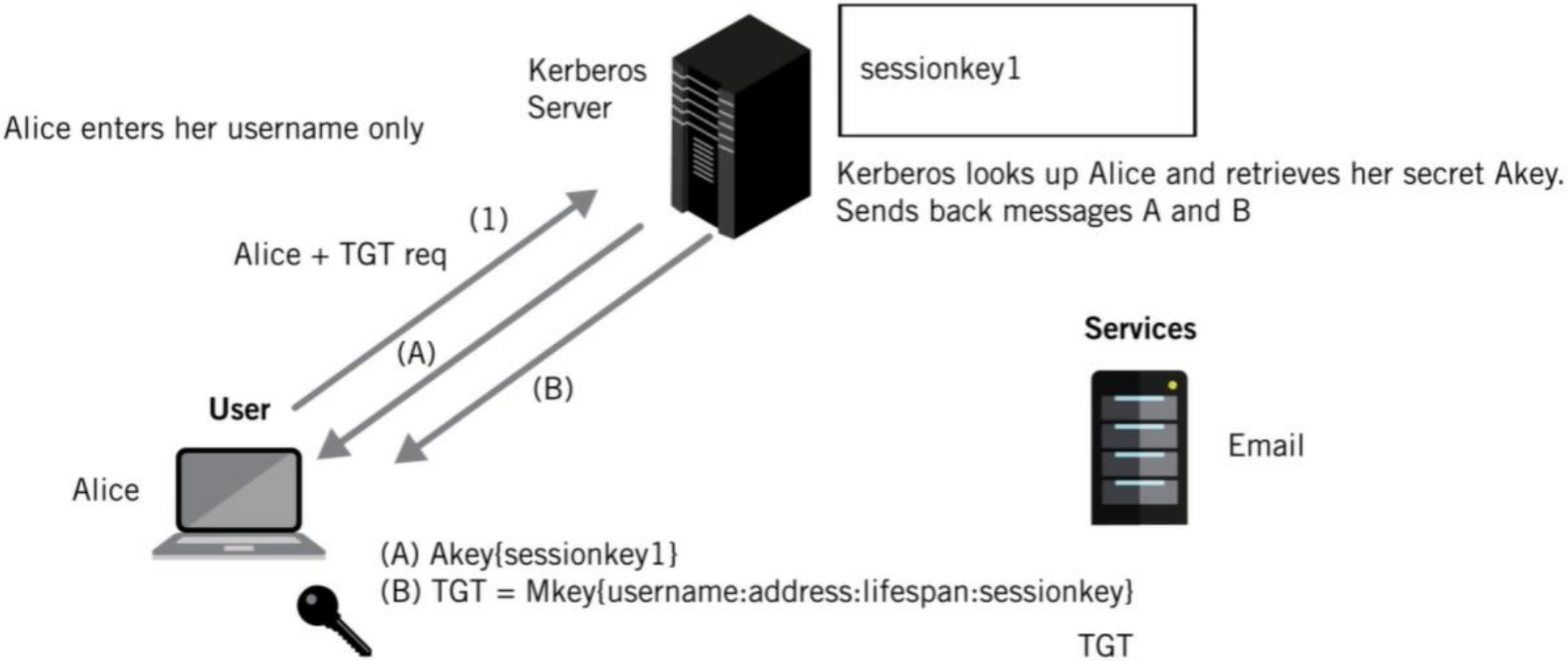


What If The Attacker Fakes The IP Address As Well?

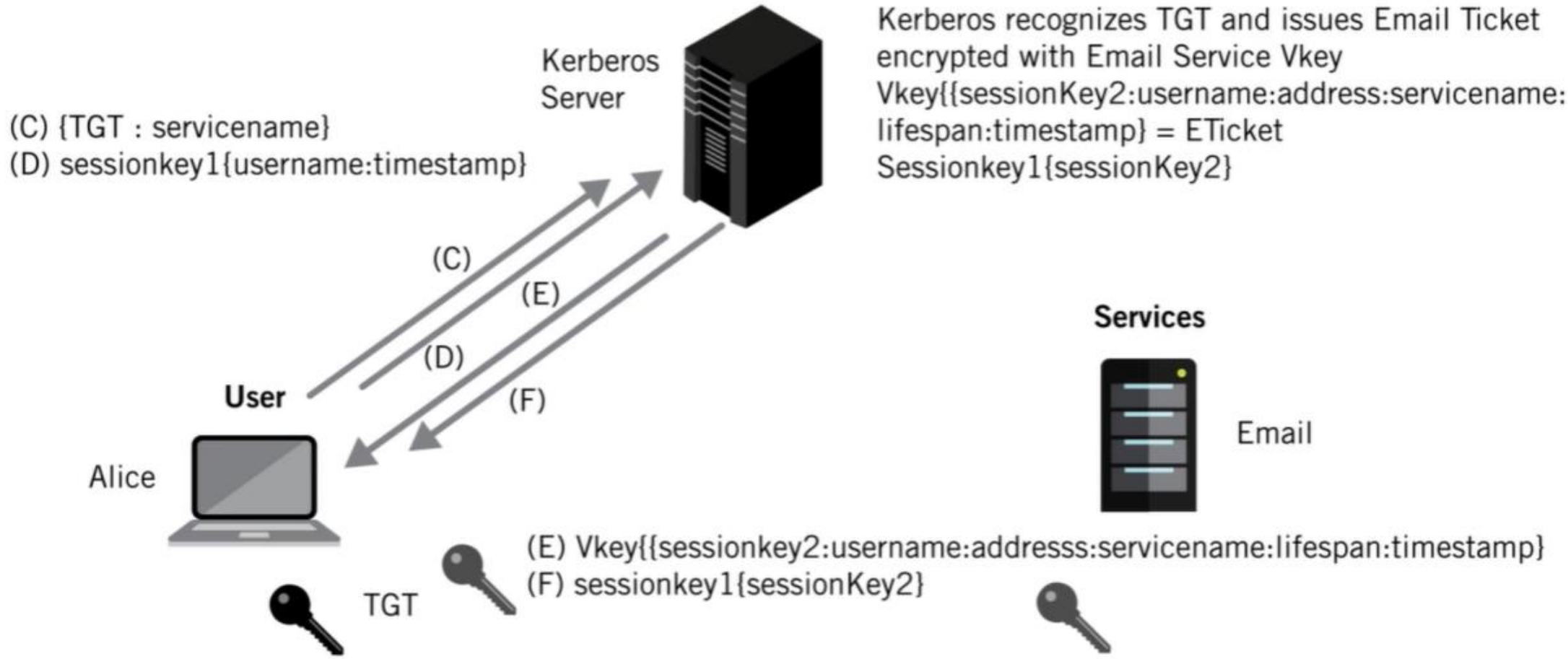
Suppose Attacker copies ETicket then sends it on to Email Server pretending to be Alice



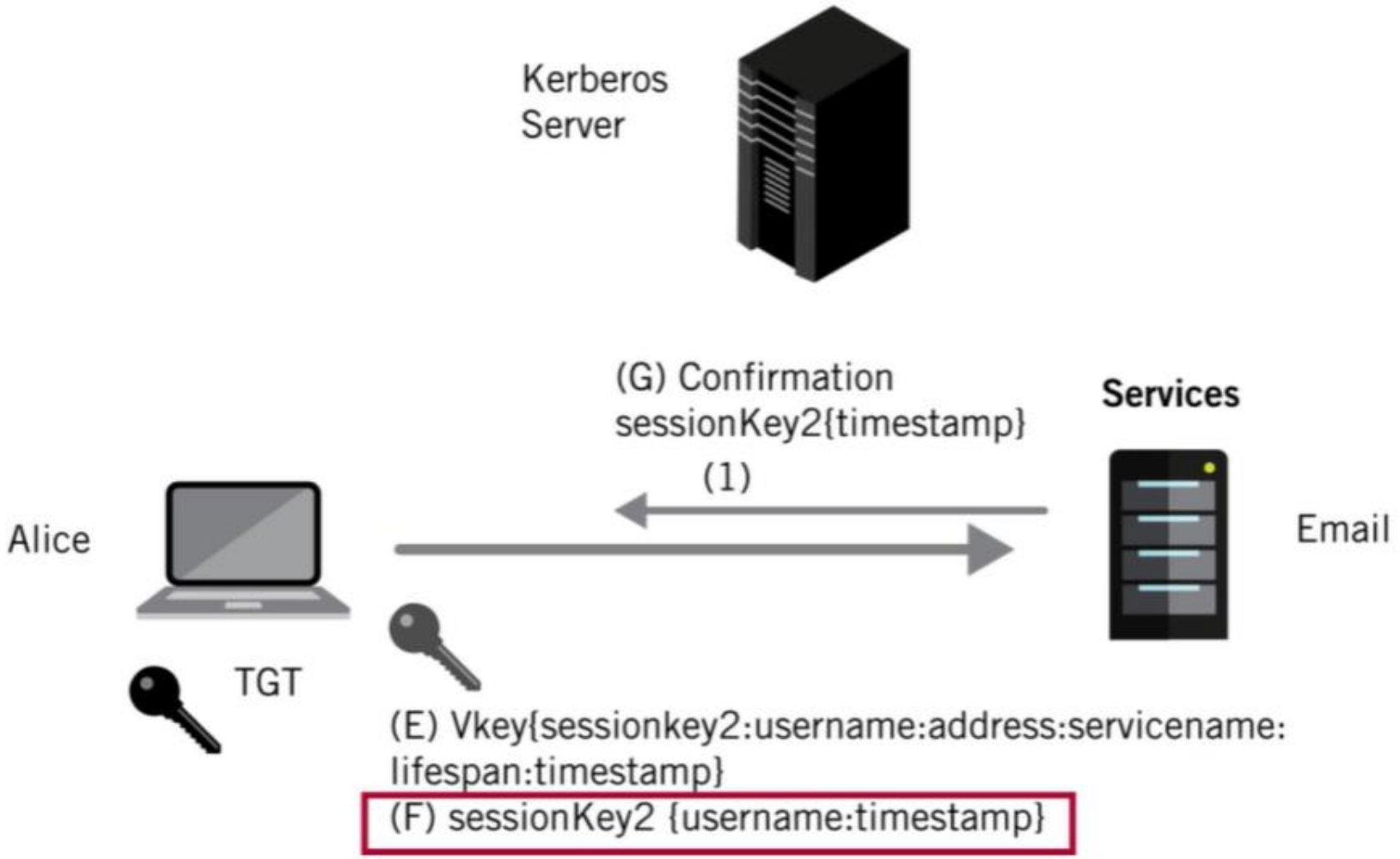
Alice Requests A Ticket-Granting-Ticket (TGT)



Alice Requests Email Server Access (1/2)



Alice Requests Email Server Access (2/2)



Authenticator

Sent from to Alice to kerberos

{username: TGT request} clear text

Sent from Kerberos Ticket Server to Alice at Login

Akey { sessionKey1 : TGT } Alice enters her password and client generates Akey

Sent from Alice to Kerberos requesting Email Ticket

sessioniKey1 {user: address : servicename}

Sent from to Kerberos to Alice

sessionKey1 {sessionKey2: Vkey{ETicket} }

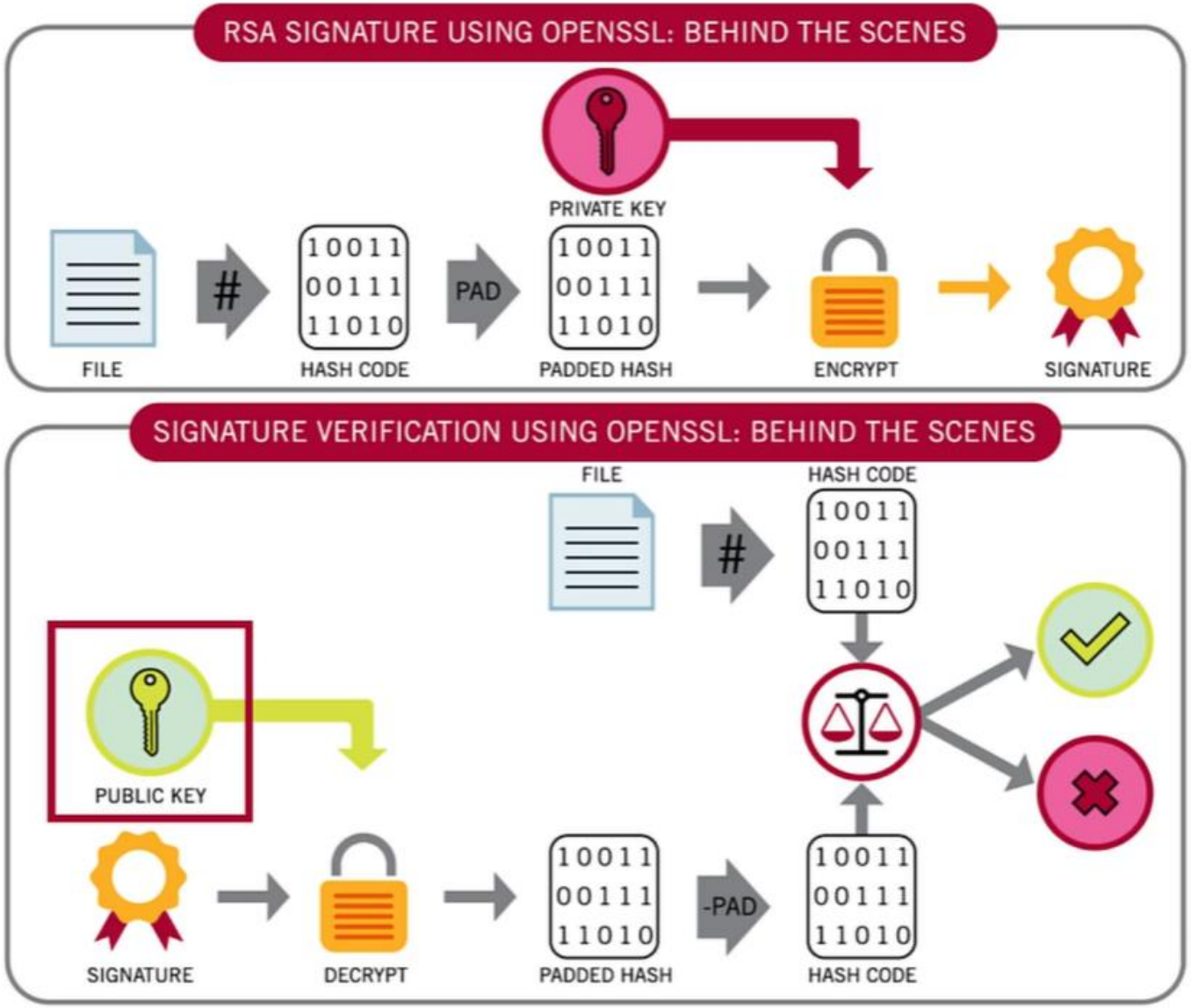
ETicket = {sessionkey2:username:address:servicename:lifespan:timestamp}

Sent from Alice to Email Server

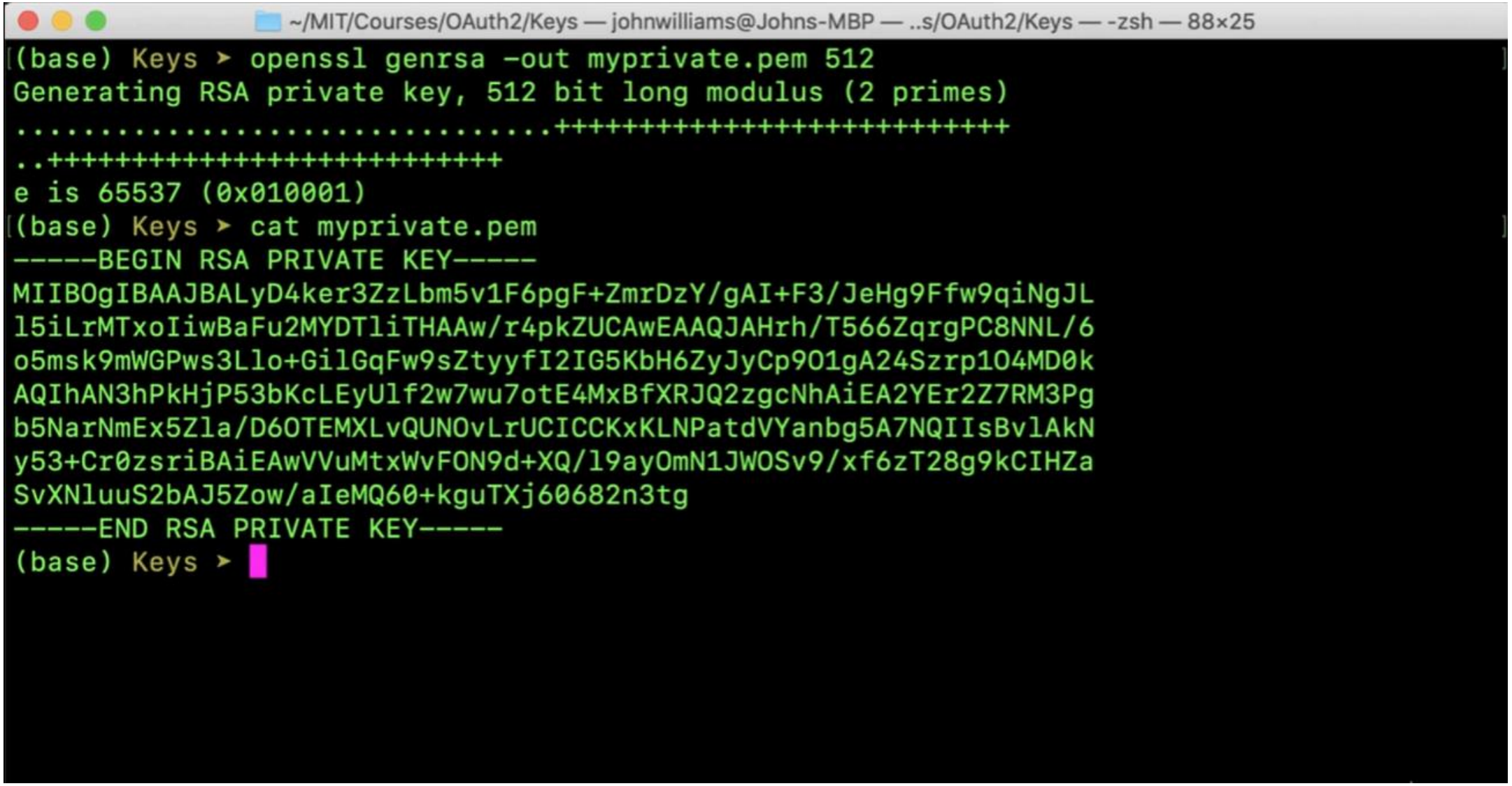
sessionKey2{ username:address } + Vkey{ETicket}

PKI Keys

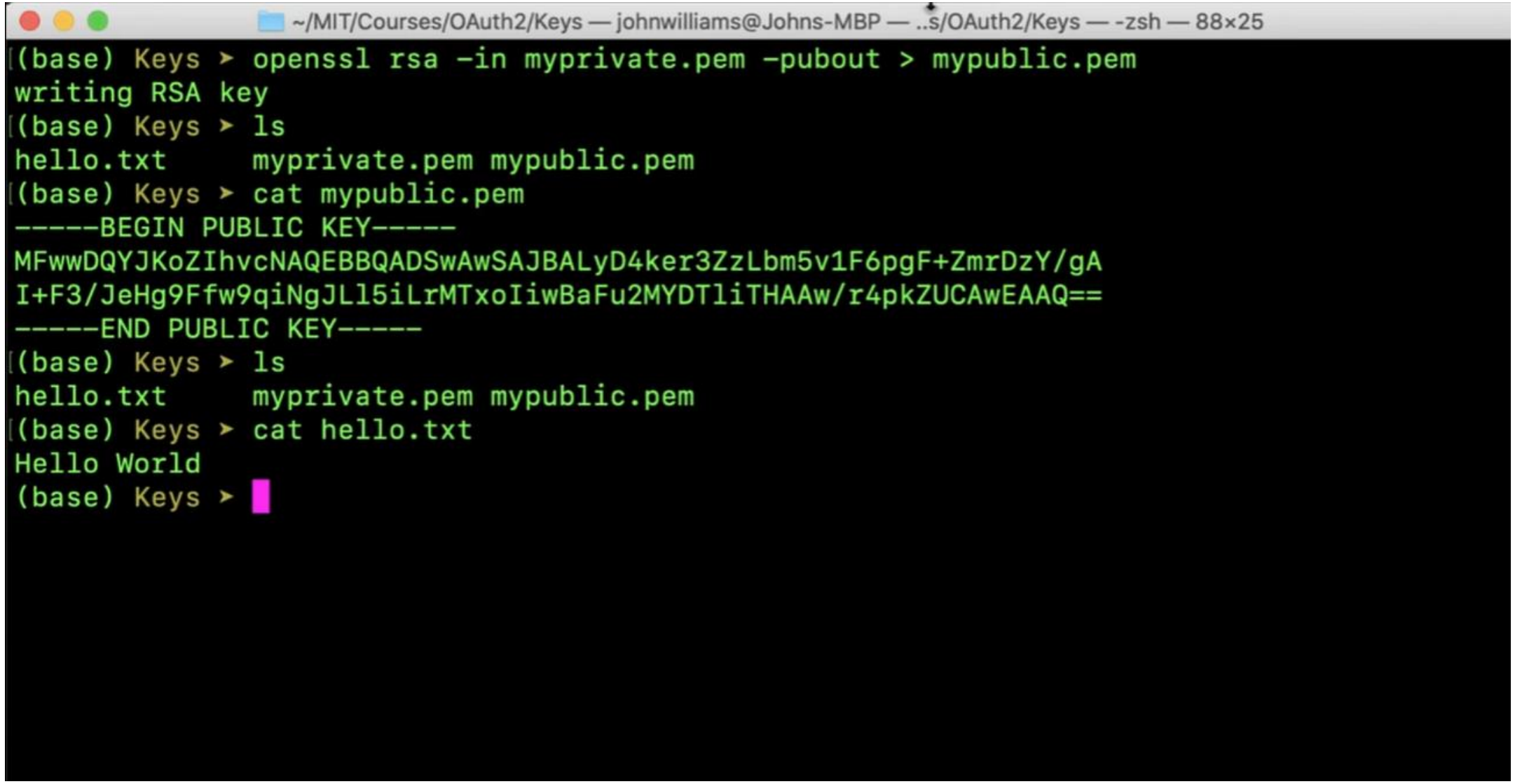
Generate A Key (1/3)



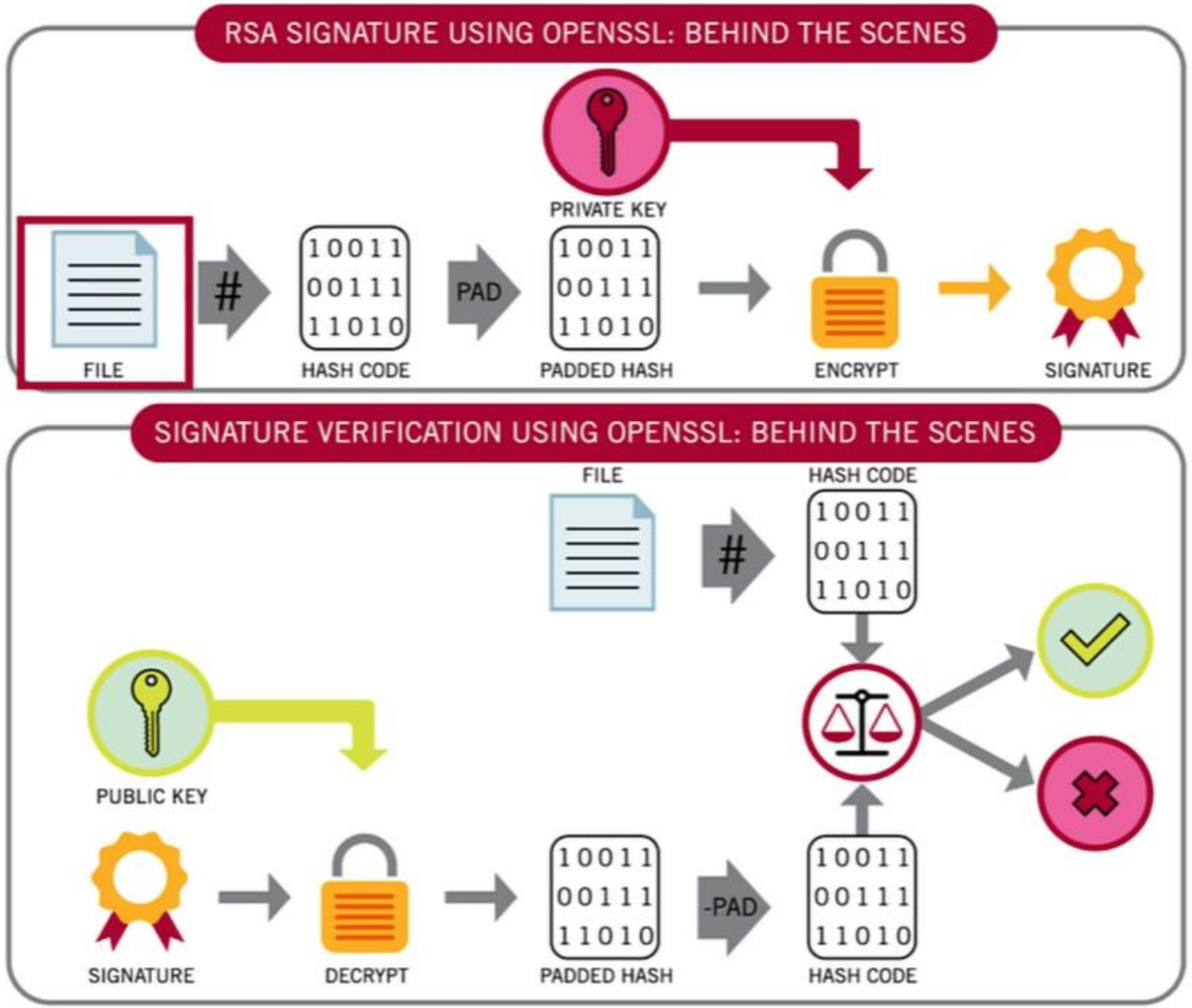
Generate A Key (2/3)



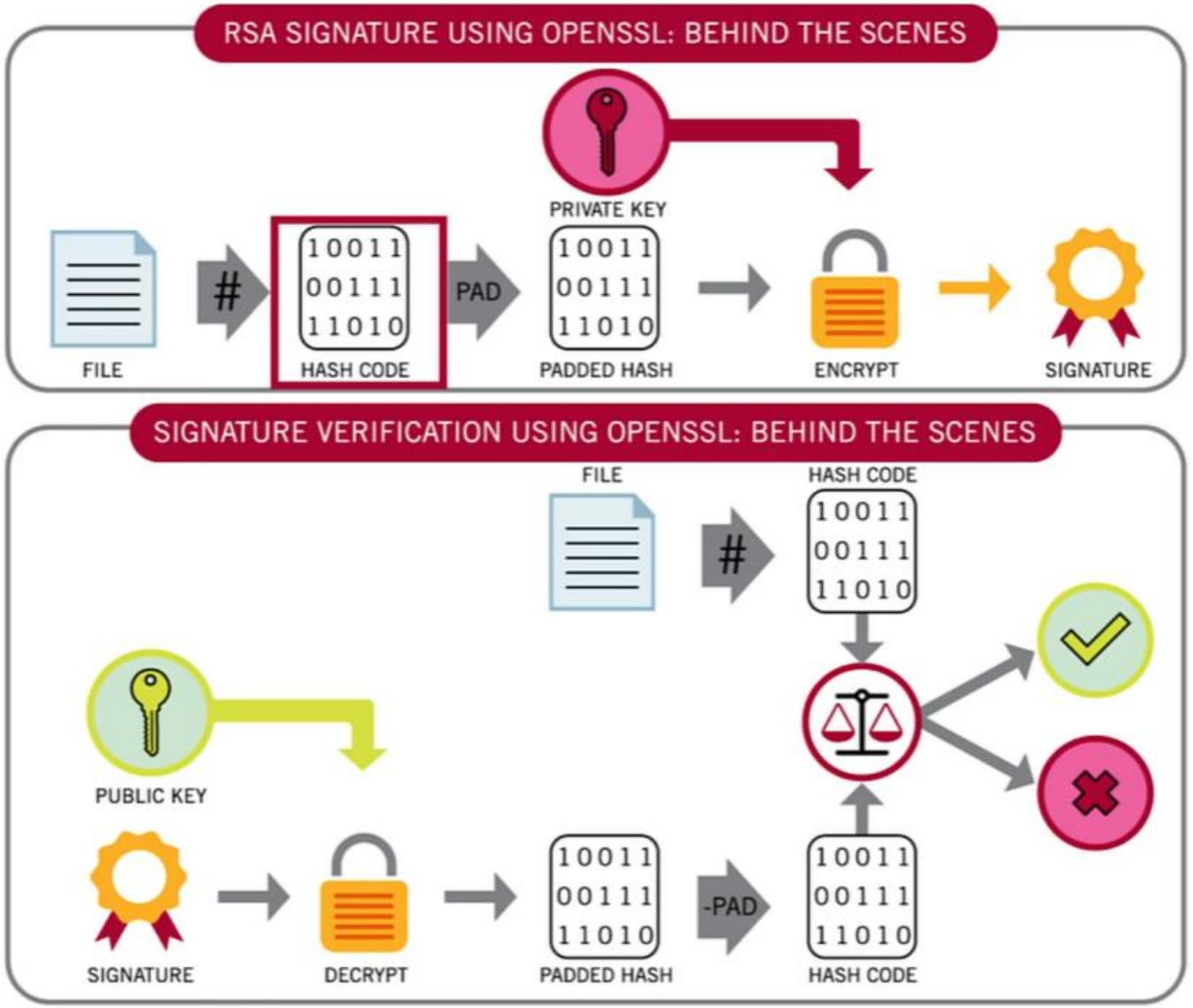
Generate A Key (3/3)



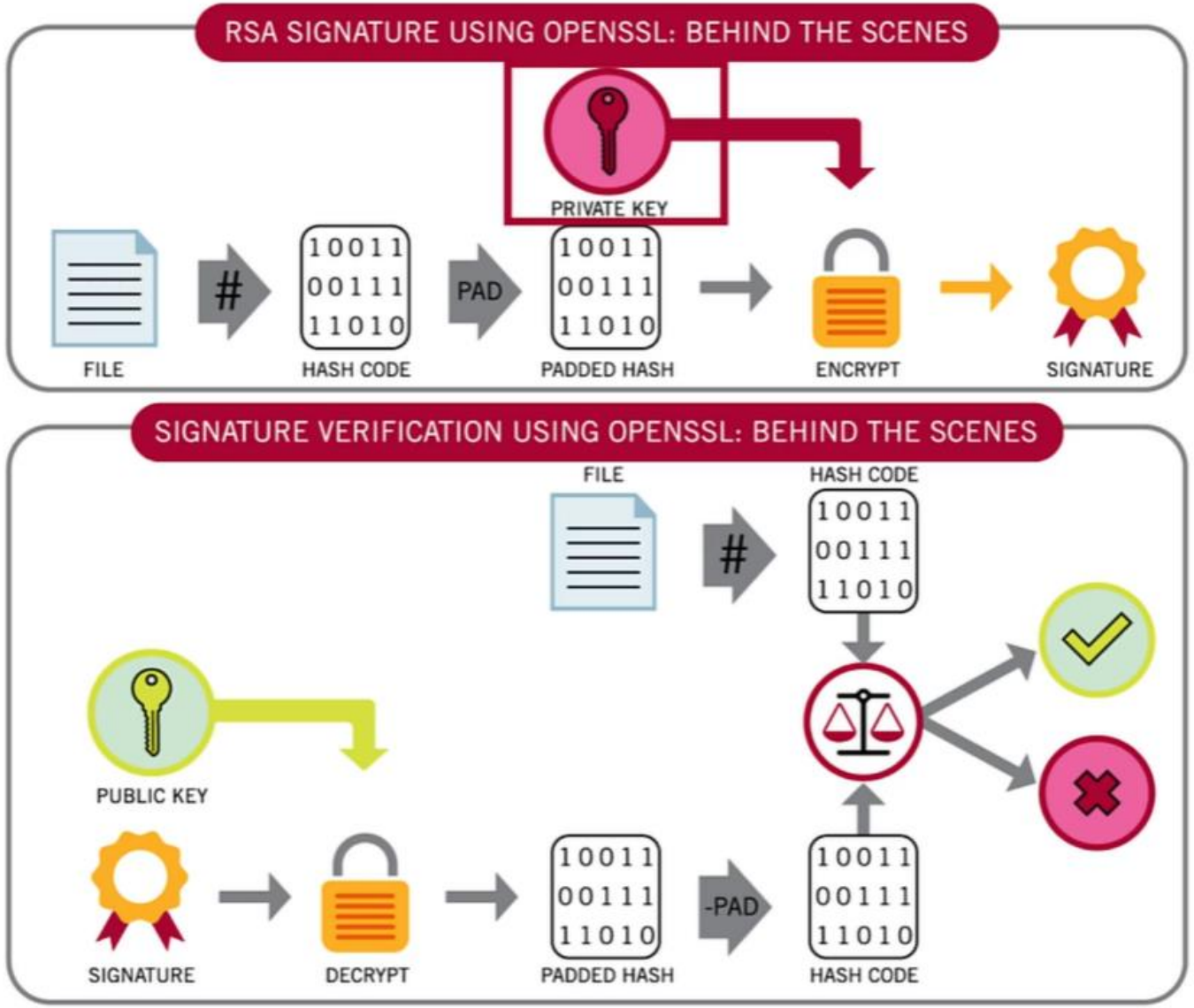
RSA Signature Using OPENSSL (1/5)



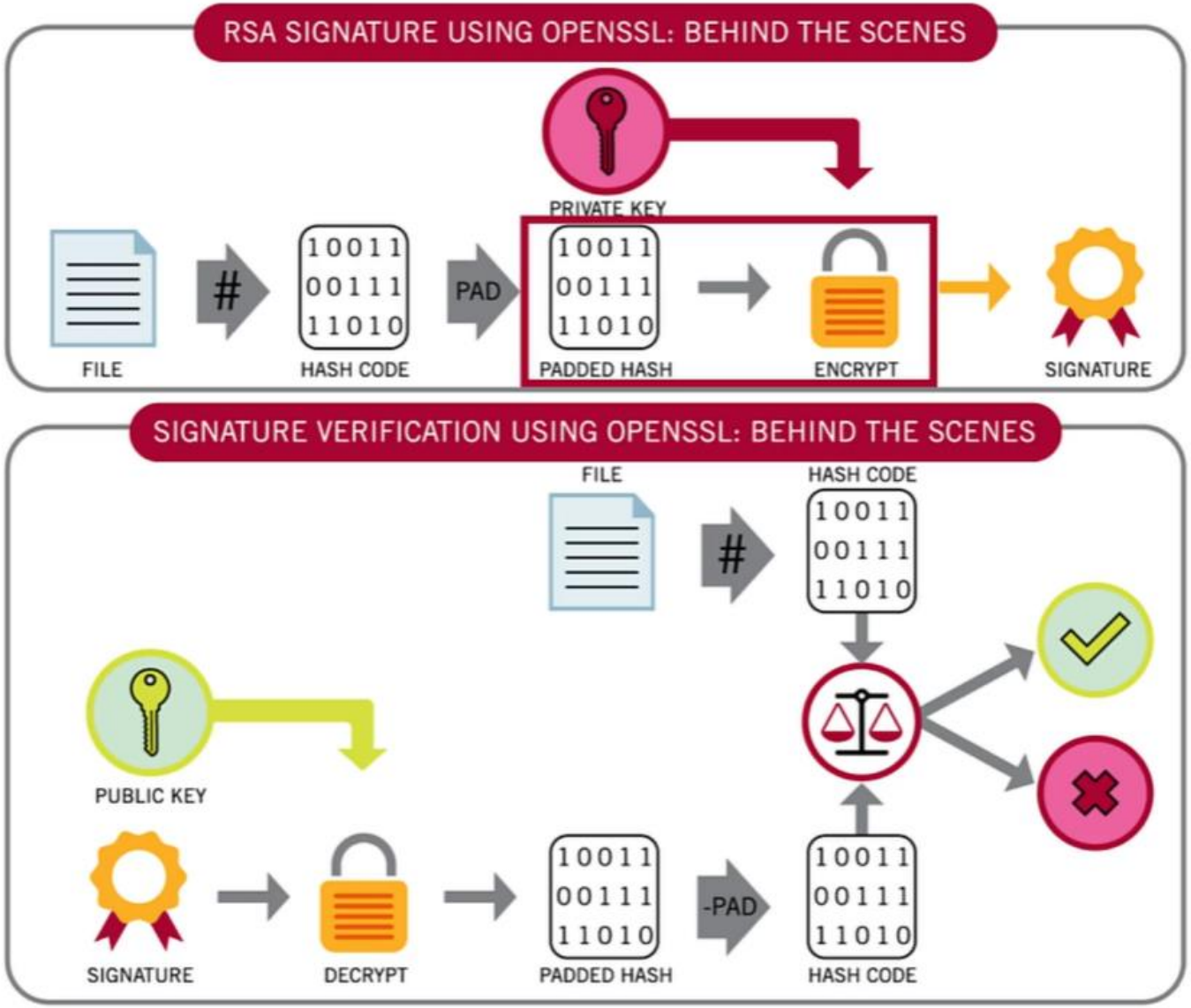
RSA Signature Using OPENSSL (2/5)



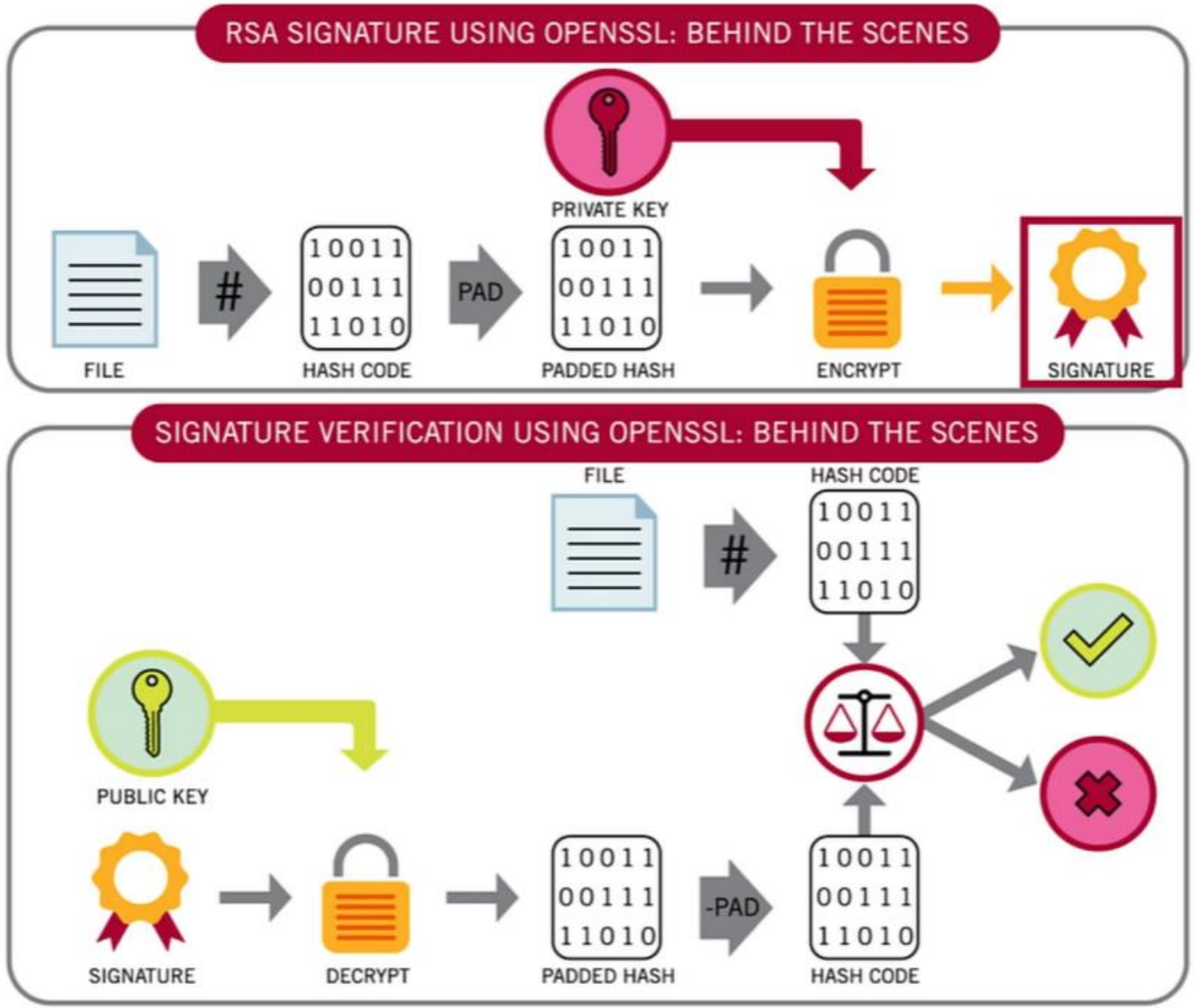
RSA Signature Using OPENSSL (3/5)



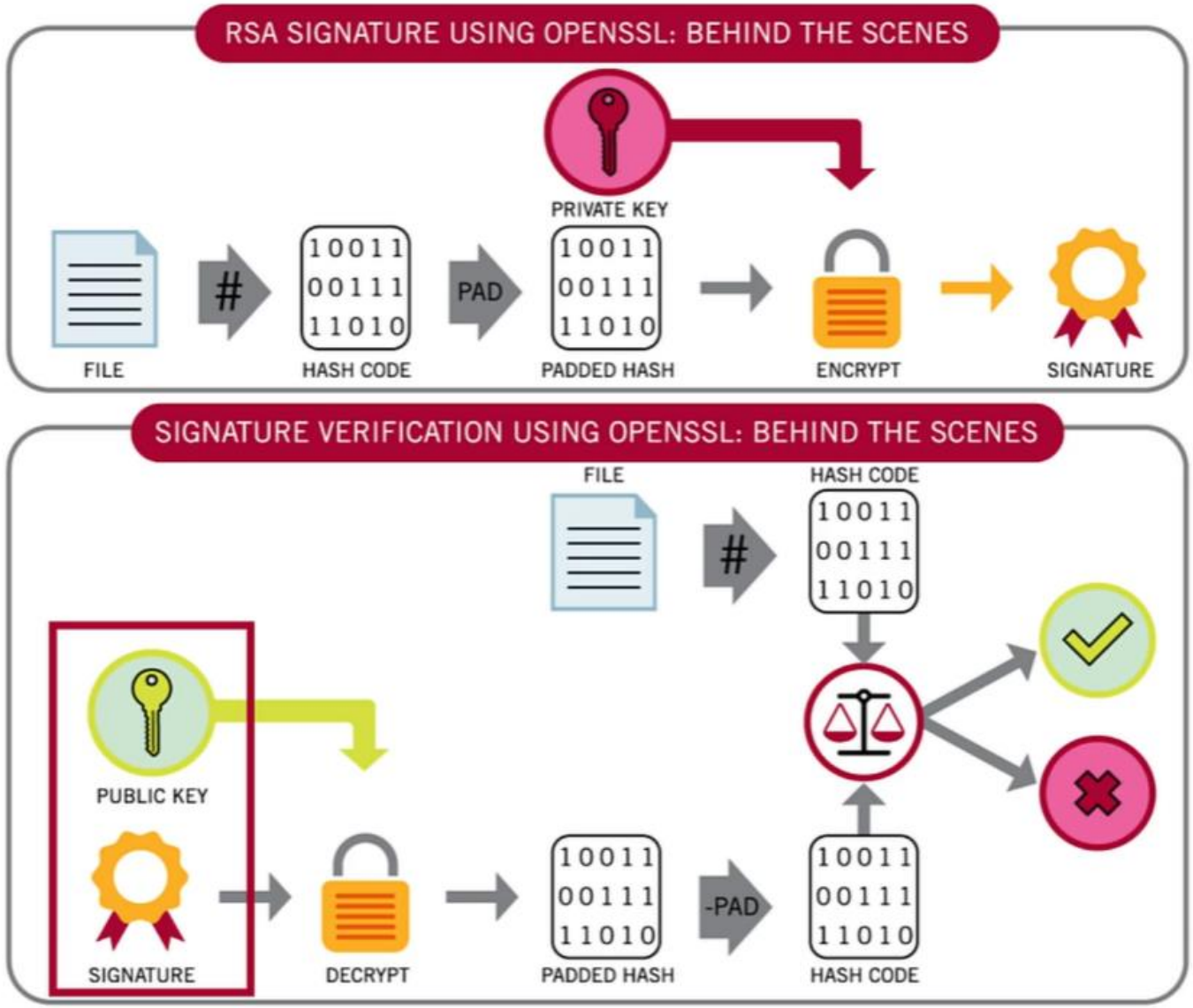
RSA Signature Using OPENSSL (4/5)



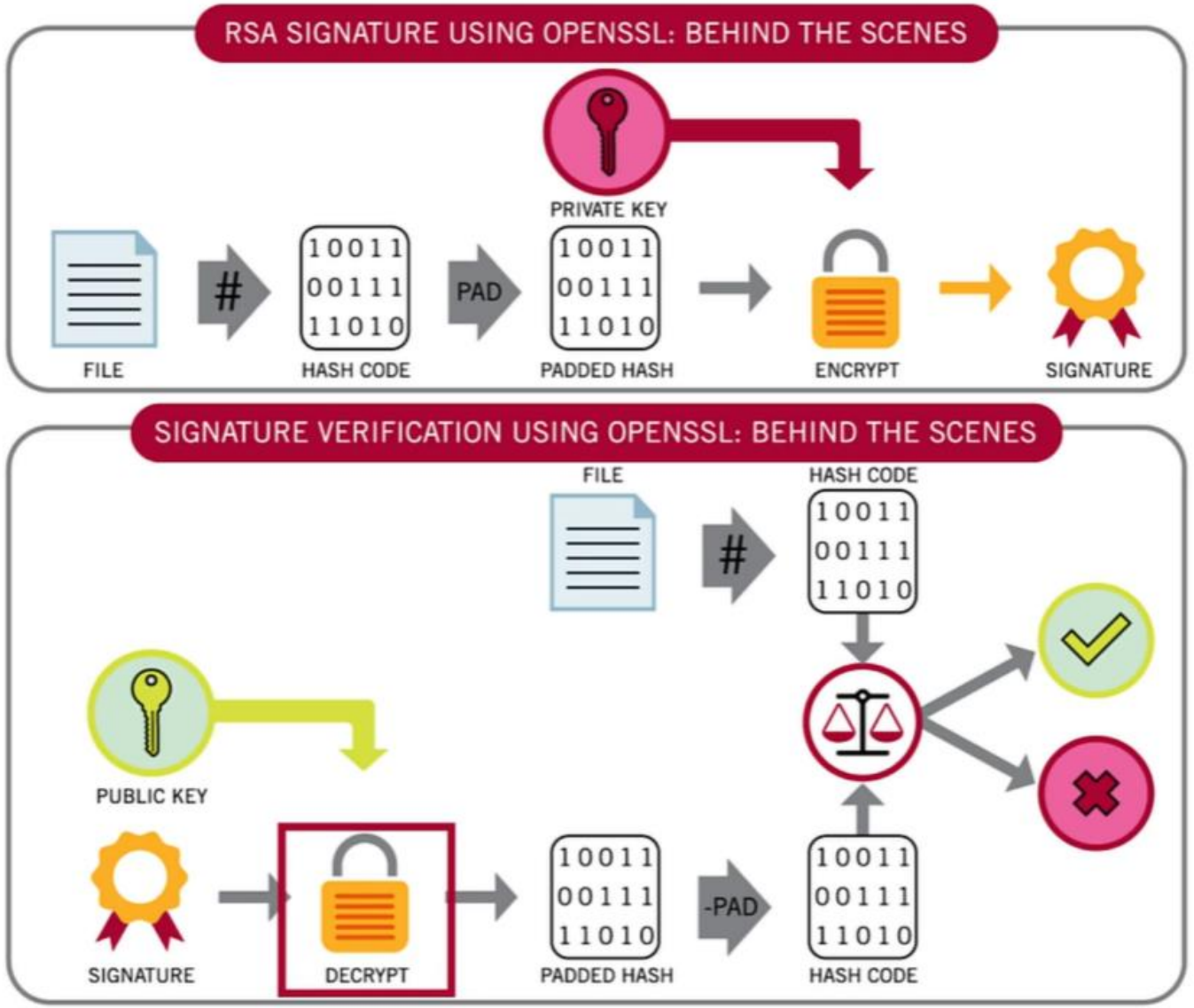
RSA Signature Using OPENSSL (5/5)



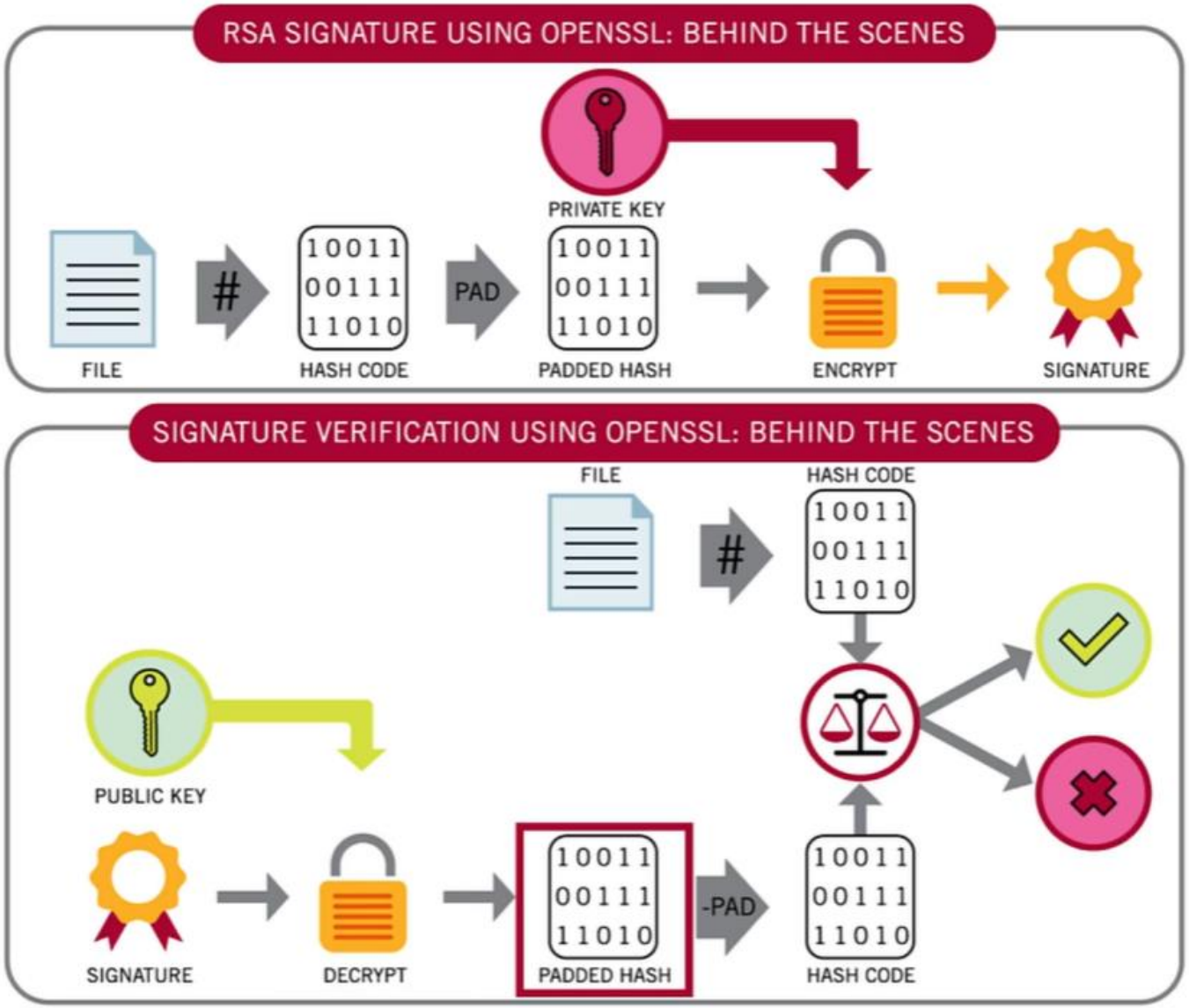
Decrypting The File (1/13)



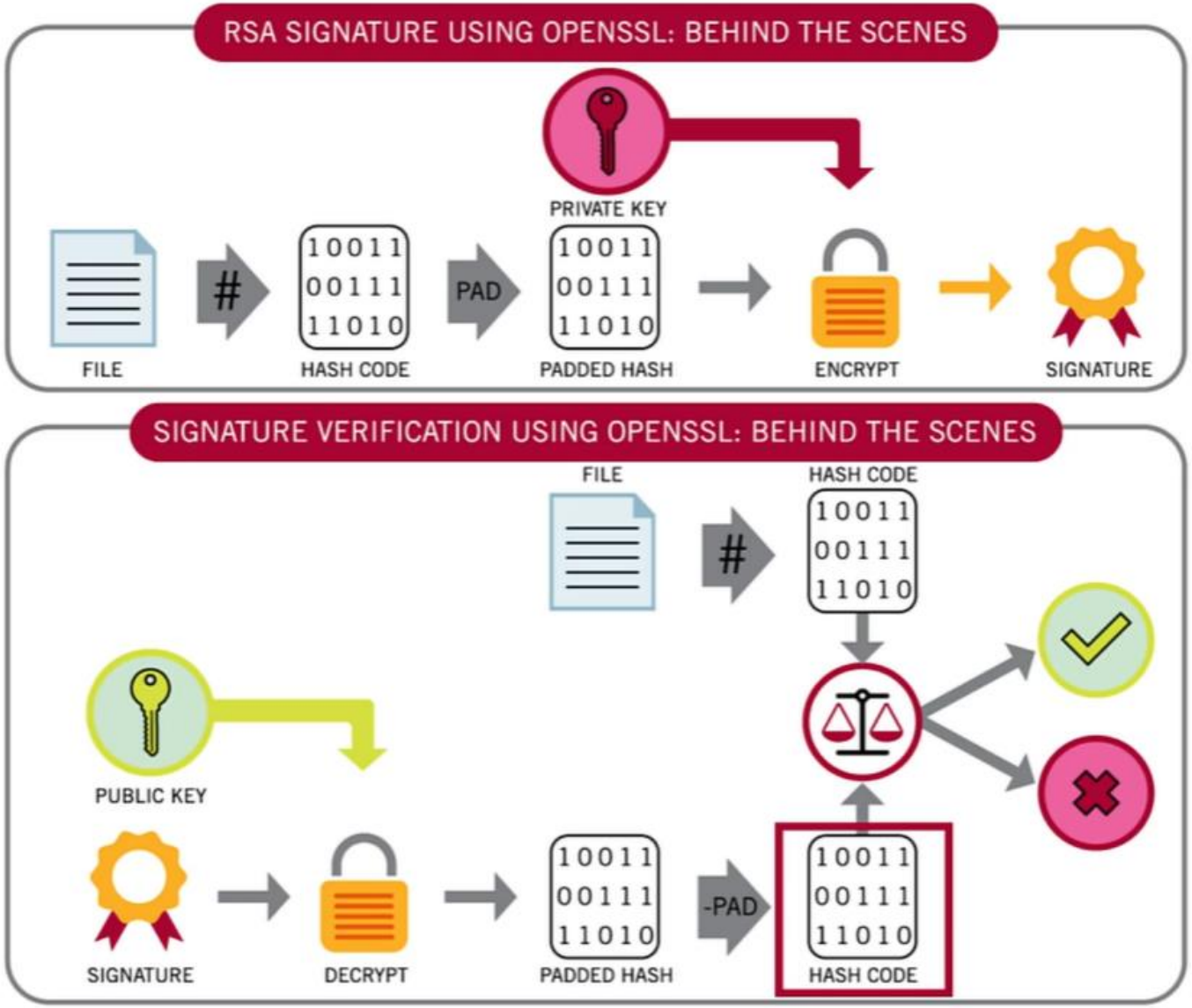
Decrypting The File (2/13)



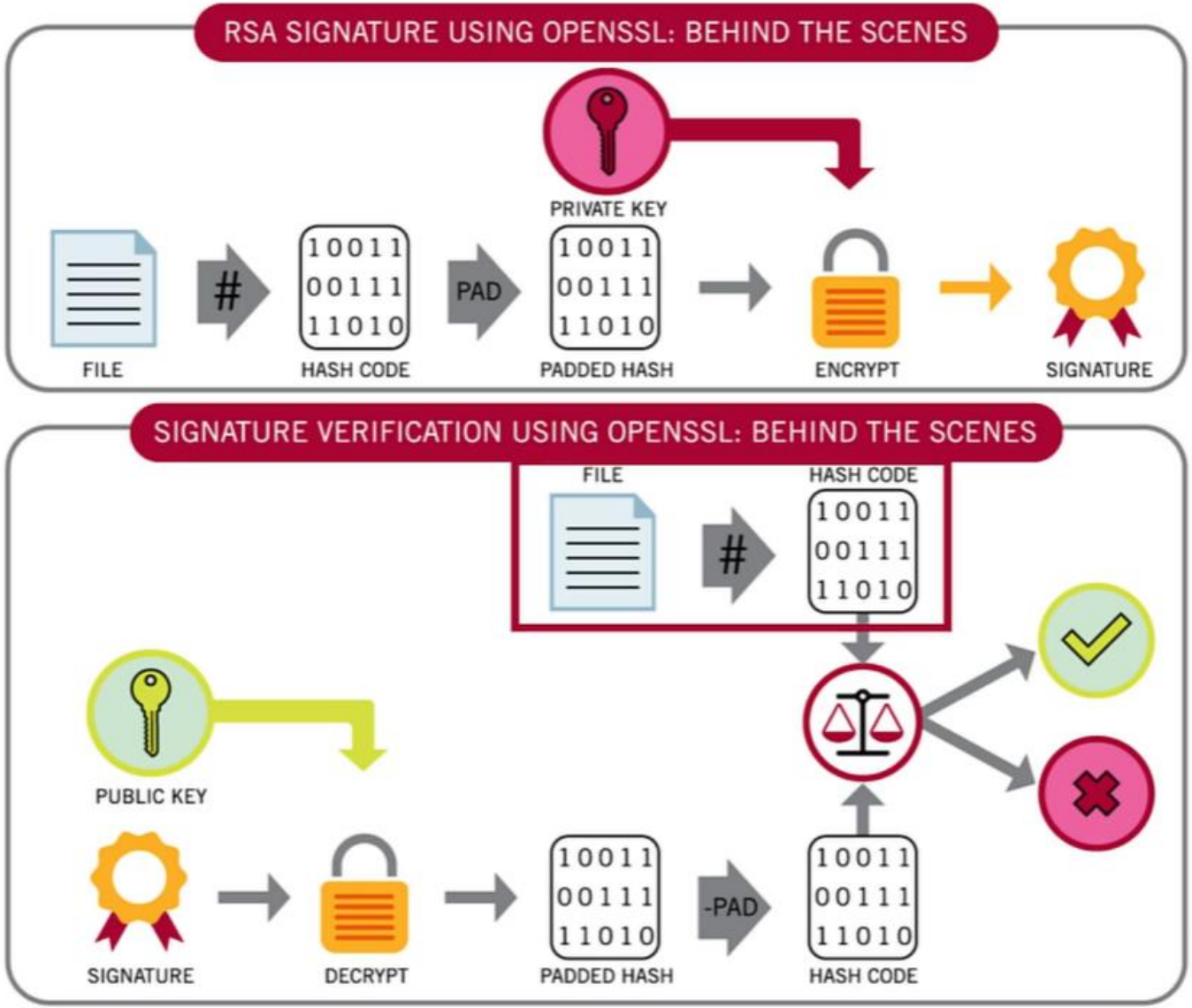
Decrypting The File (3/13)



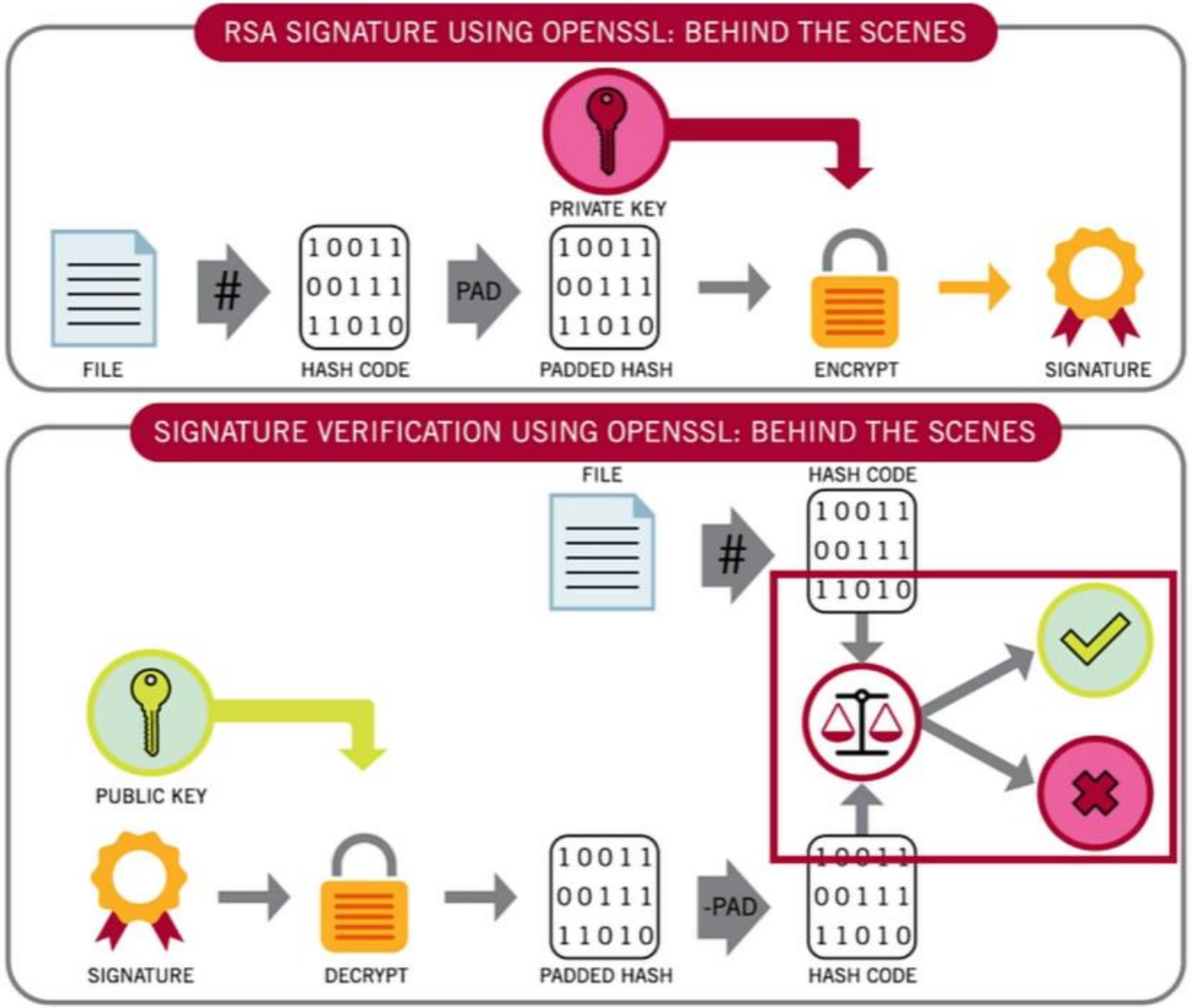
Decrypting The File (4/13)



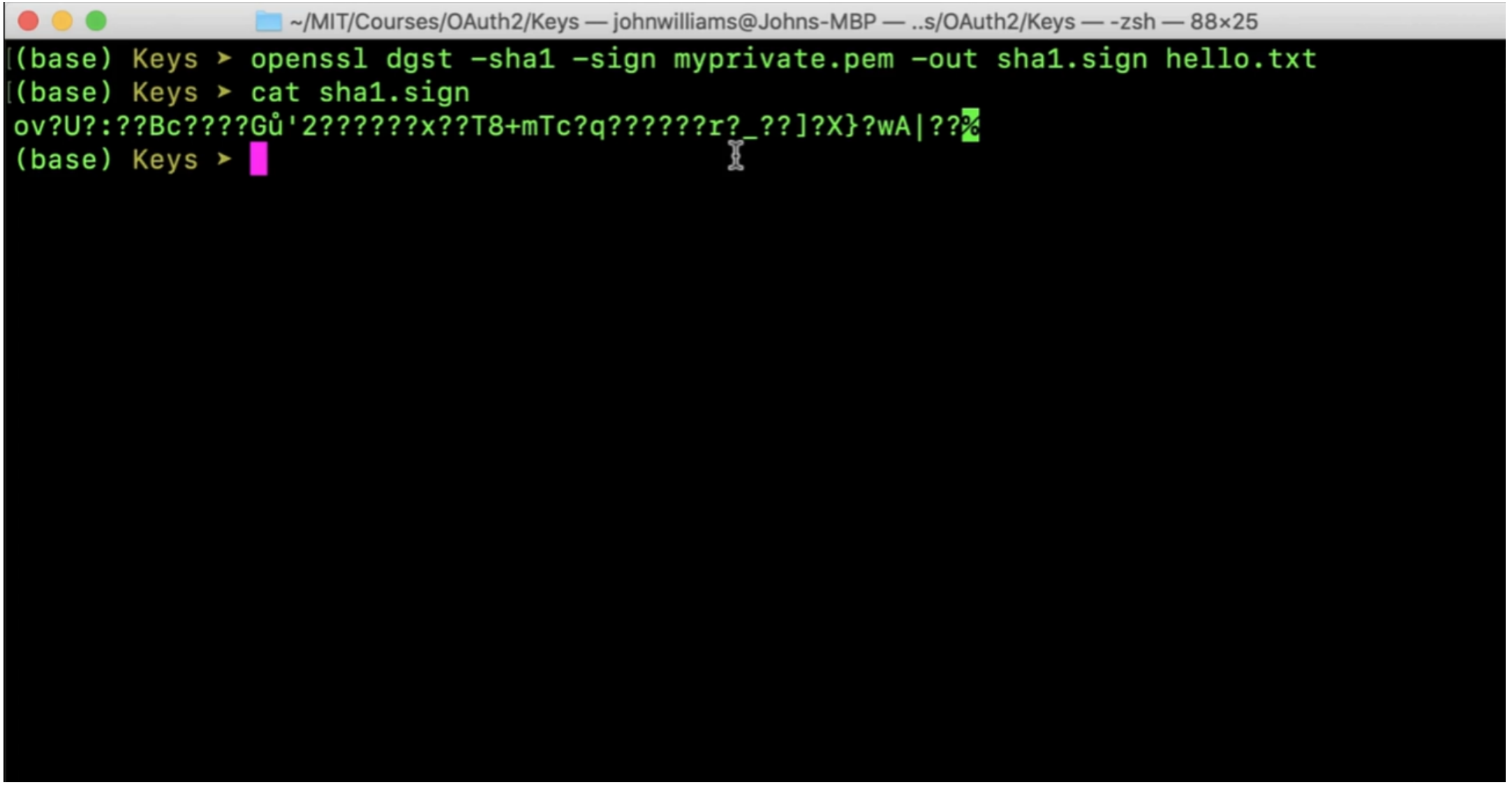
Decrypting The File (5/13)



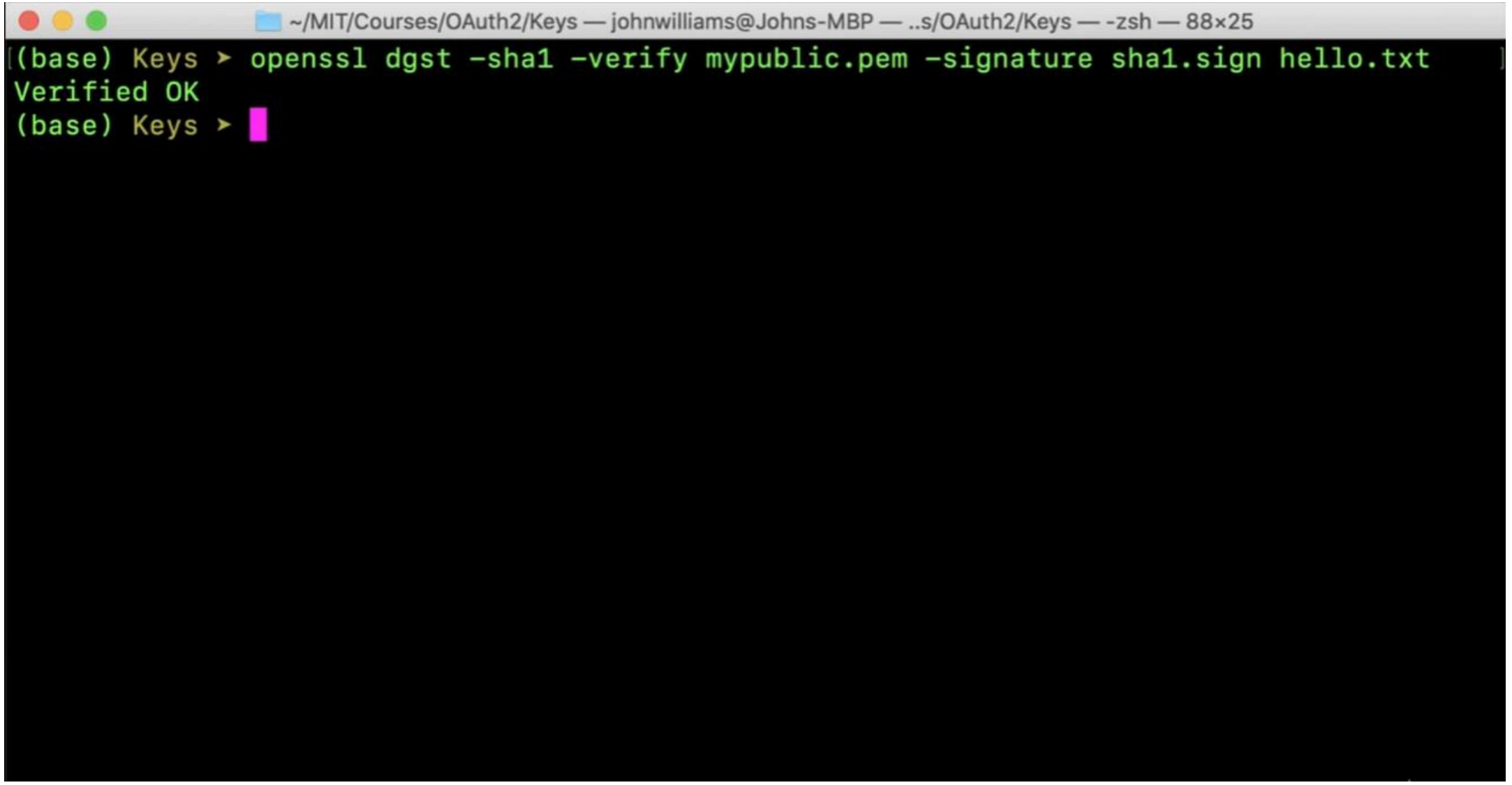
Decrypting The File (6/13)



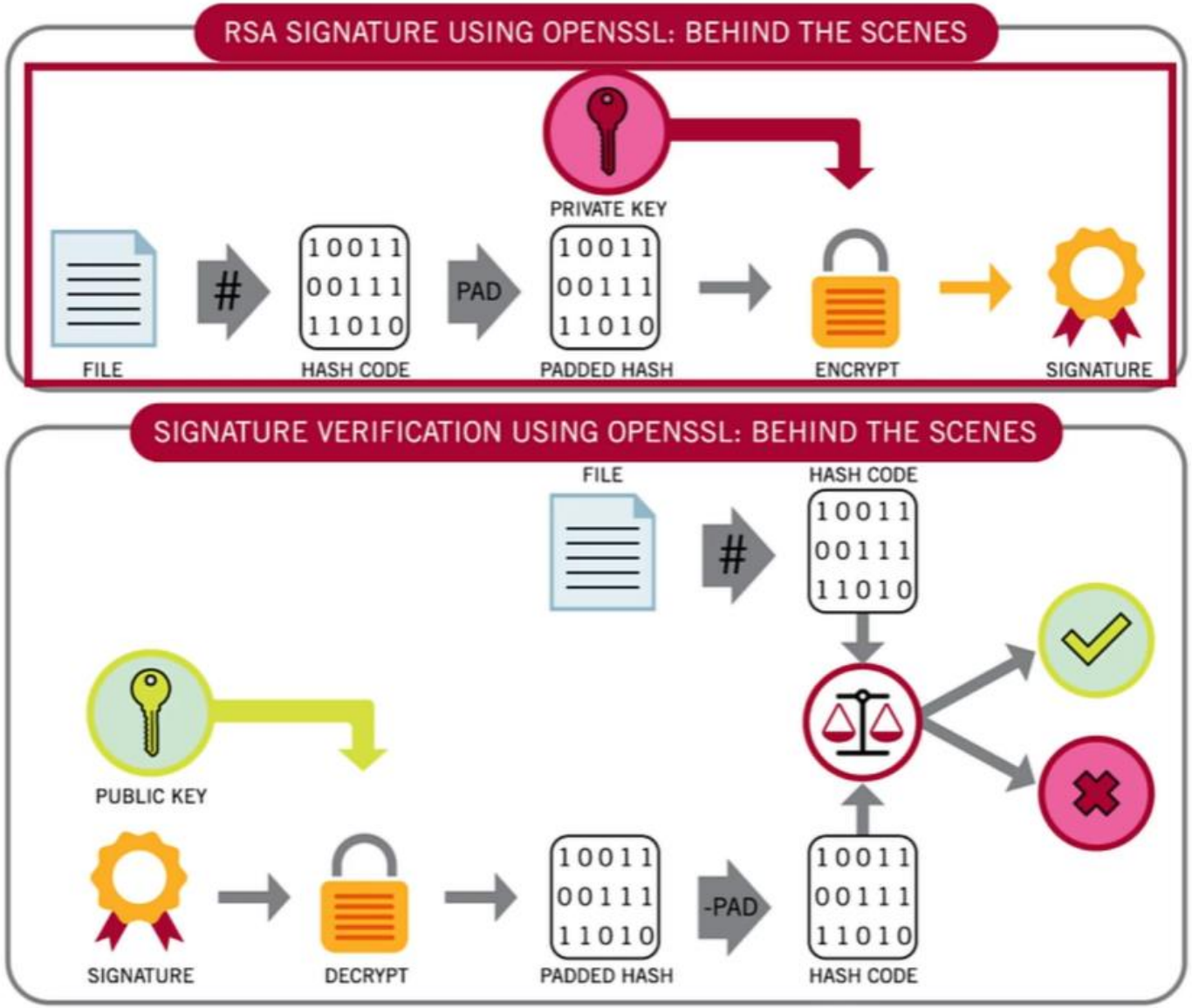
Decrypting The File (7/13)



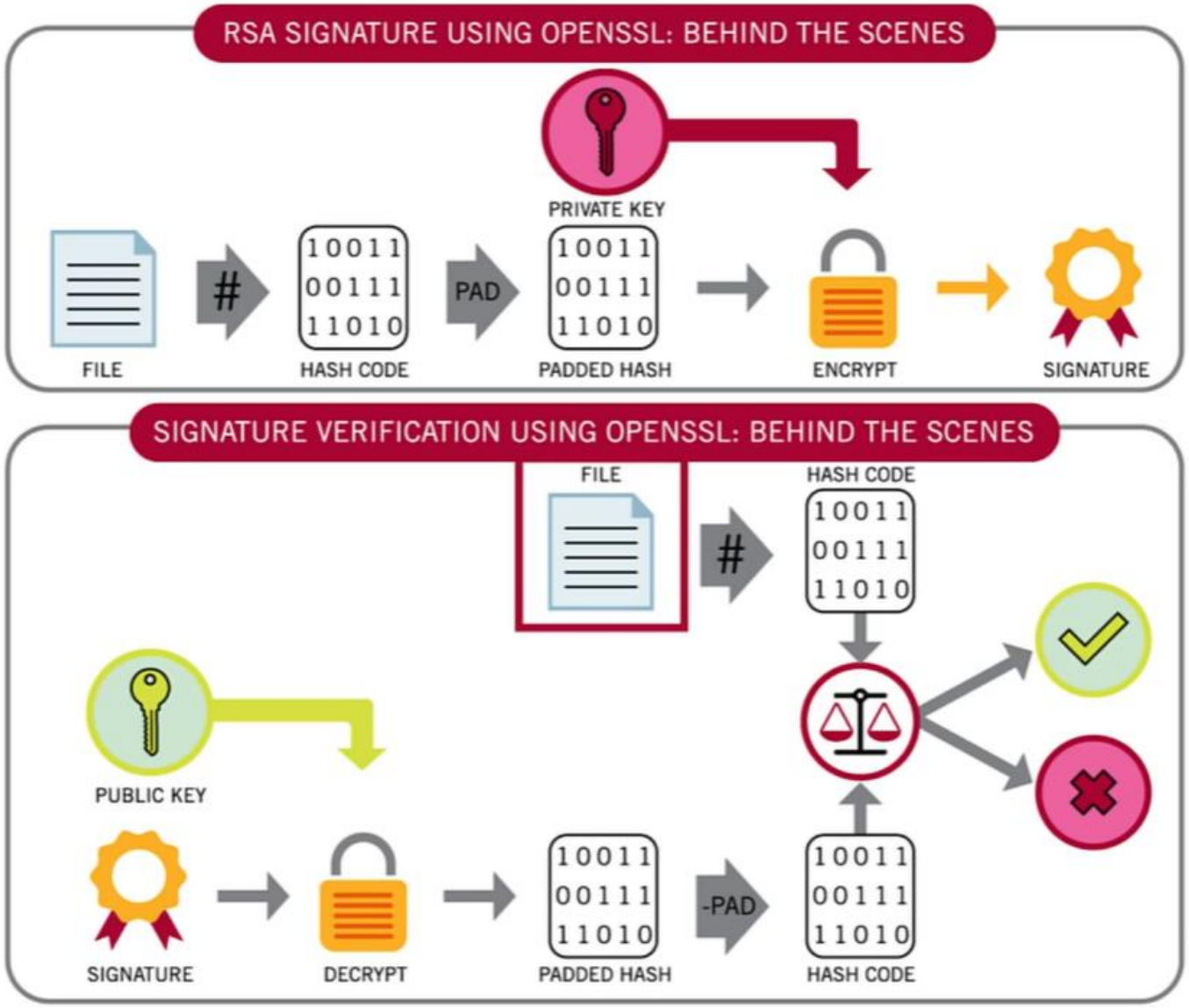
Decrypting The File (8/13)



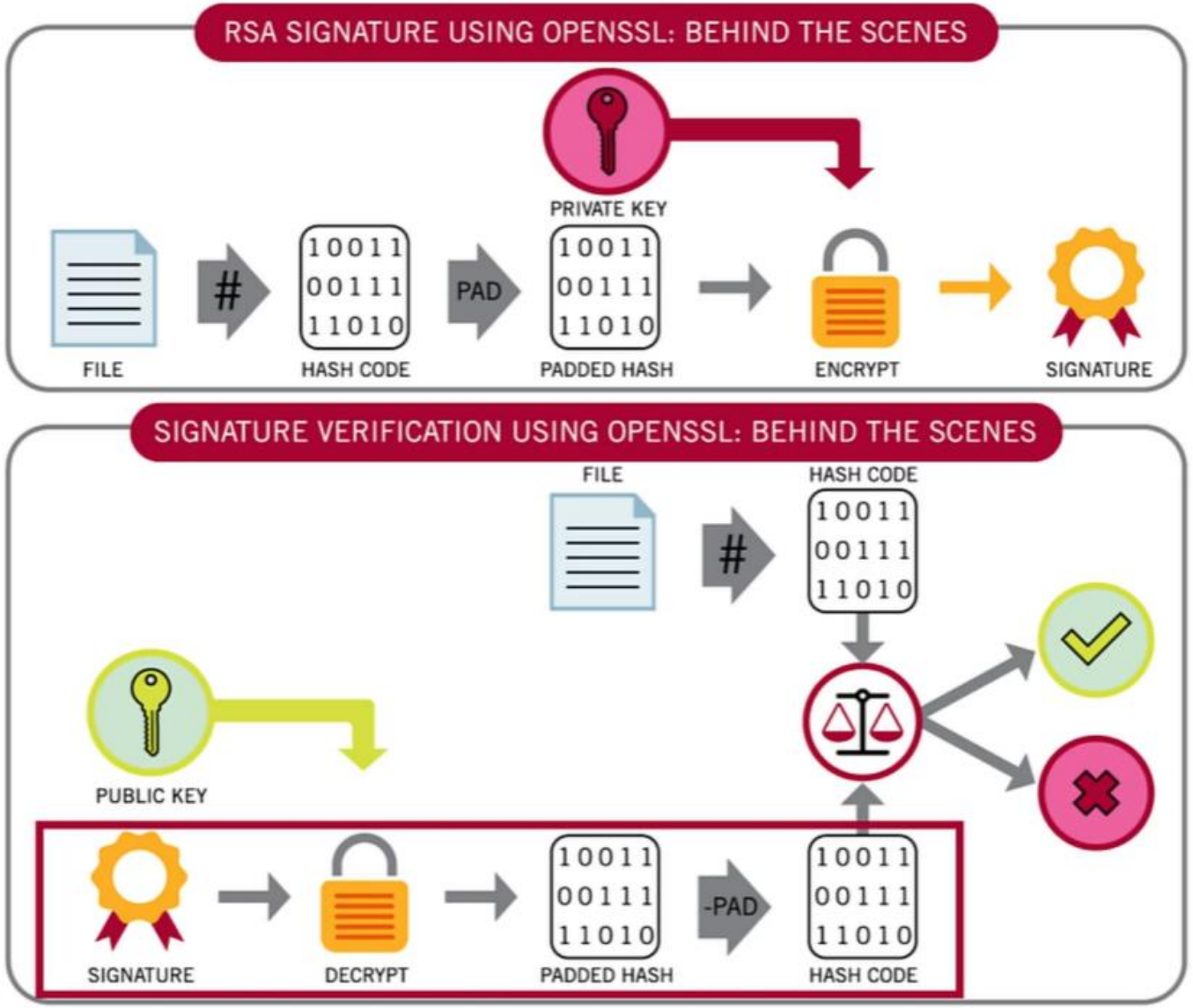
Decrypting The File (9/13)



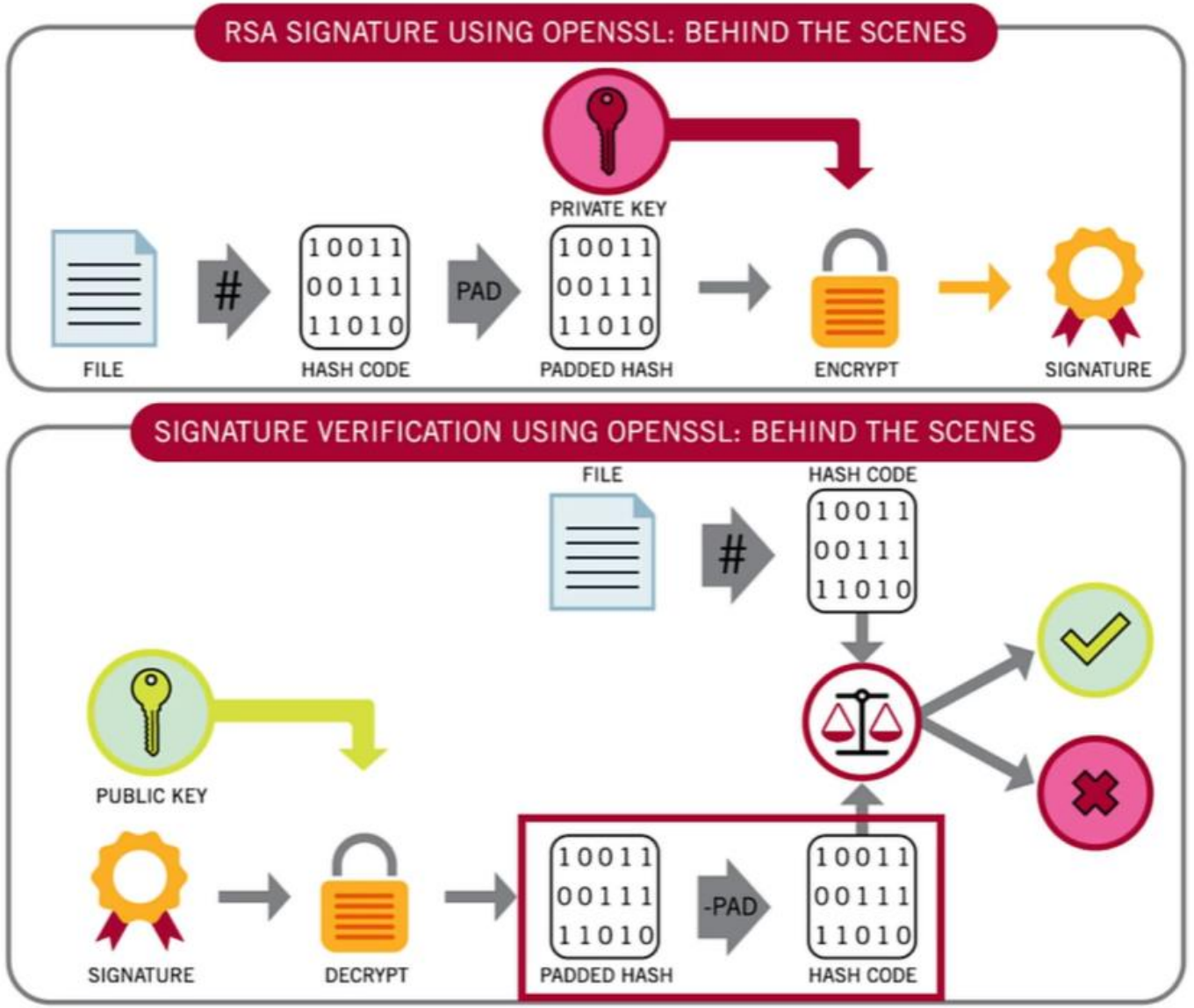
Decrypting The File (10/13)



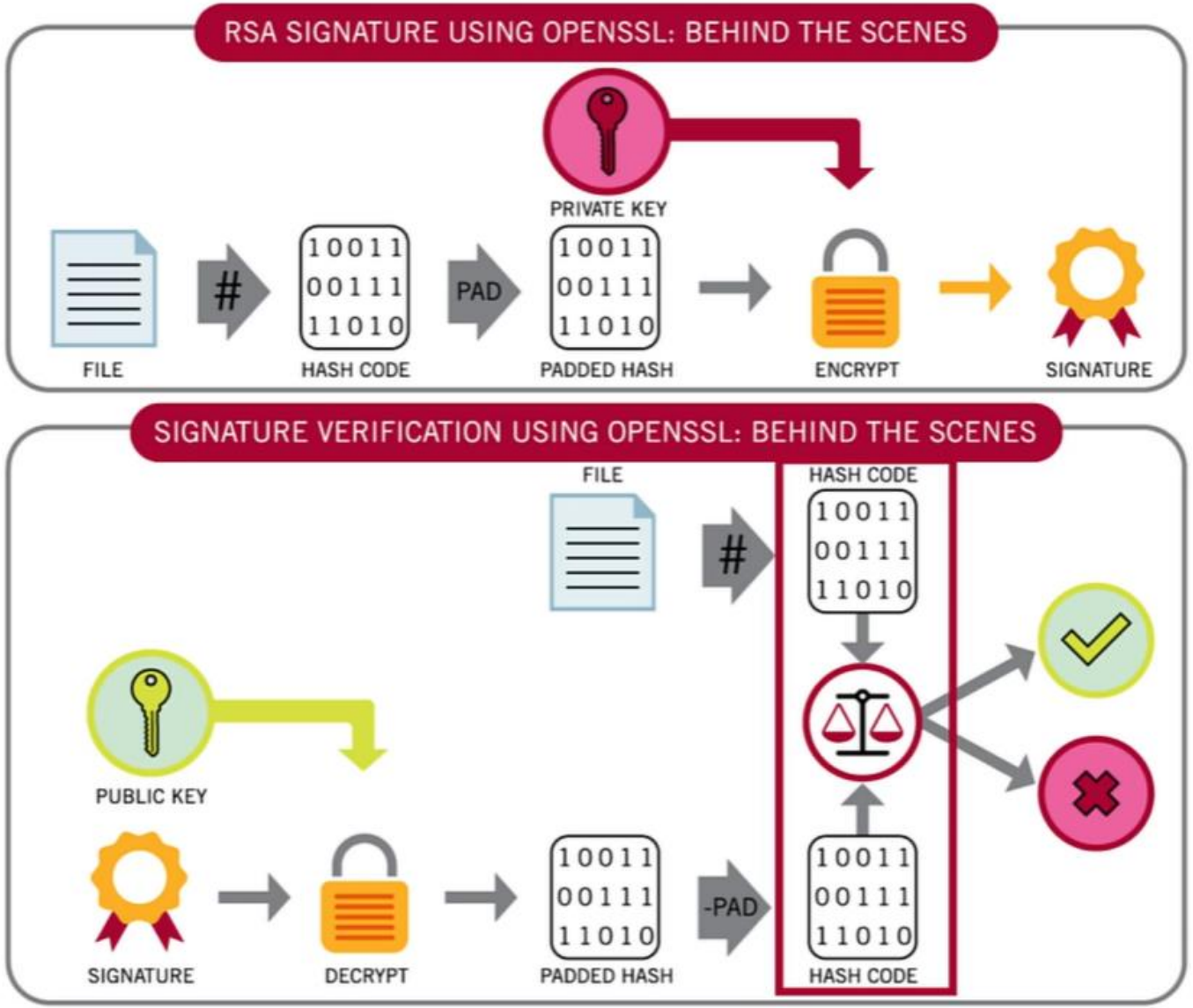
Decrypting The File (11/13)



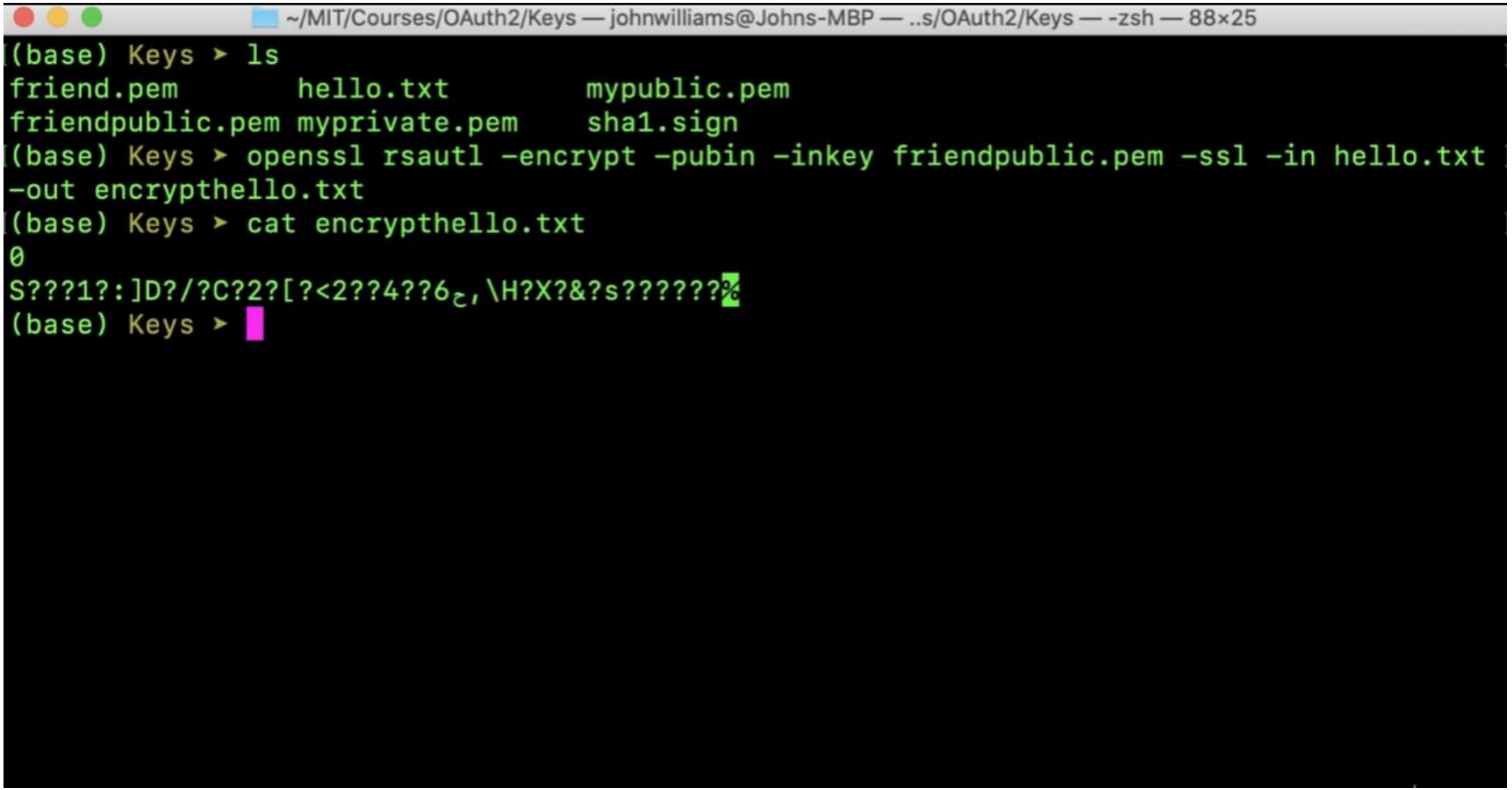
Decrypting The File (12/13)



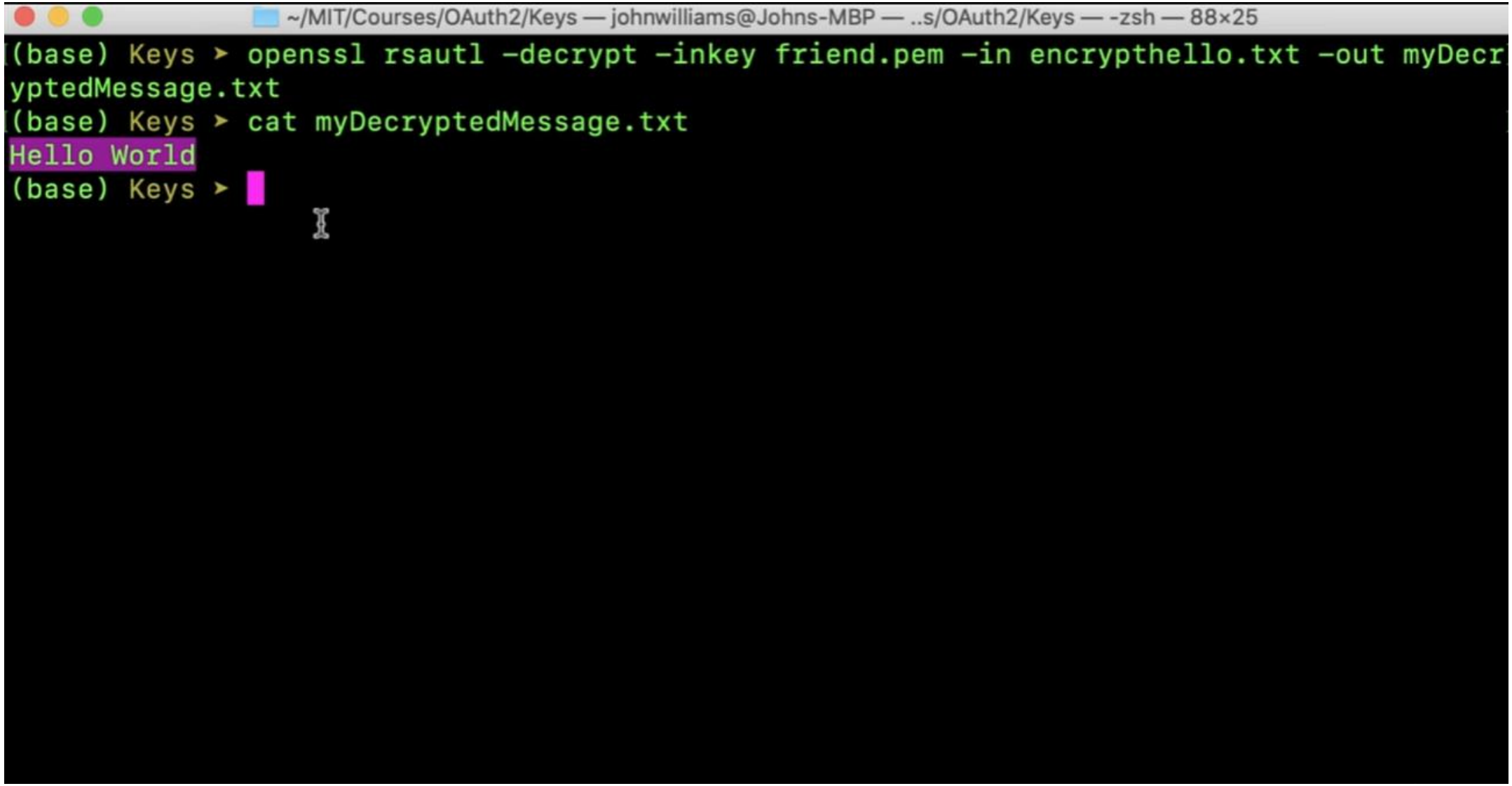
Decrypting The File (13/13)



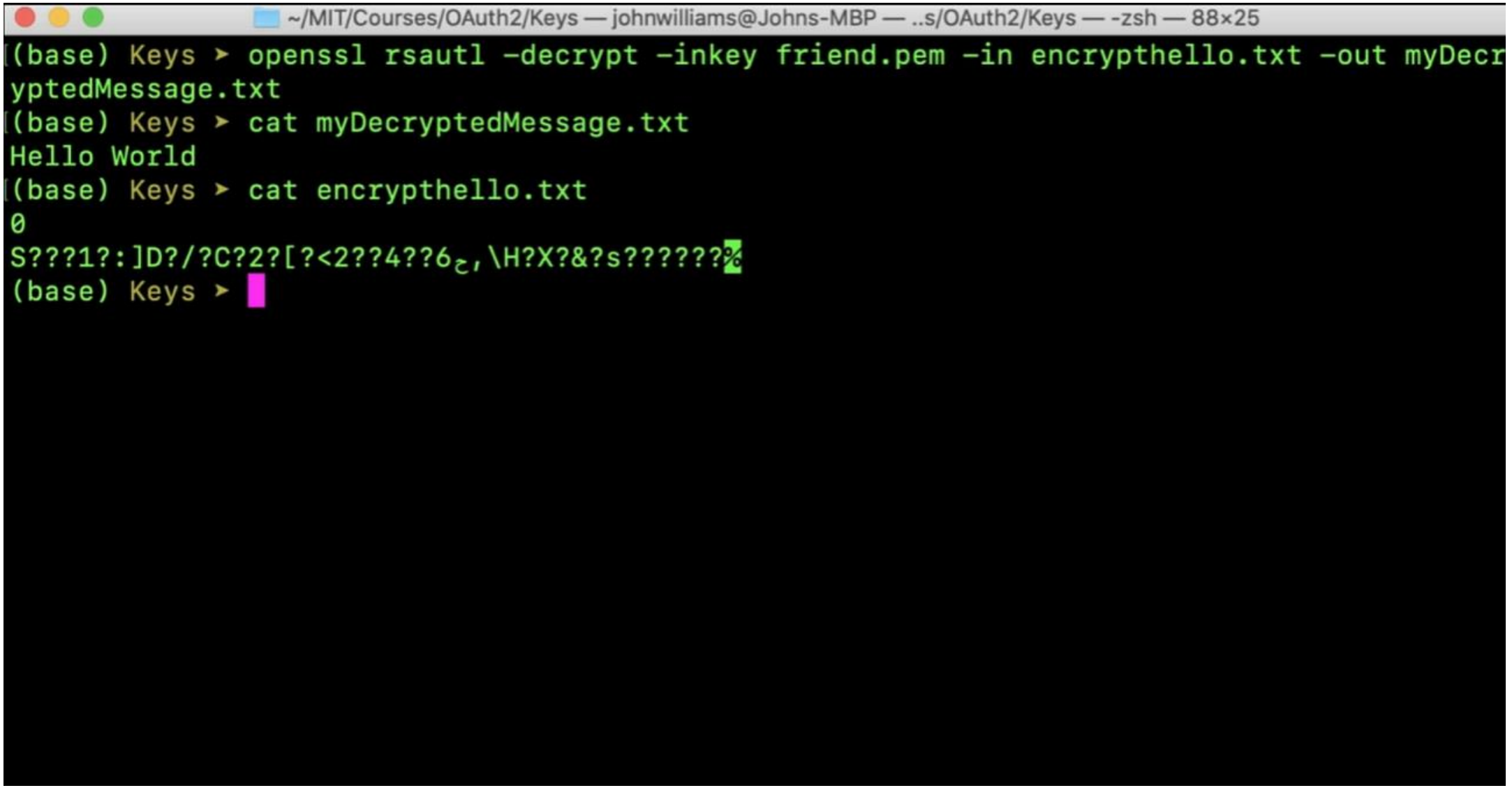
Encrypting The Message (1/2)



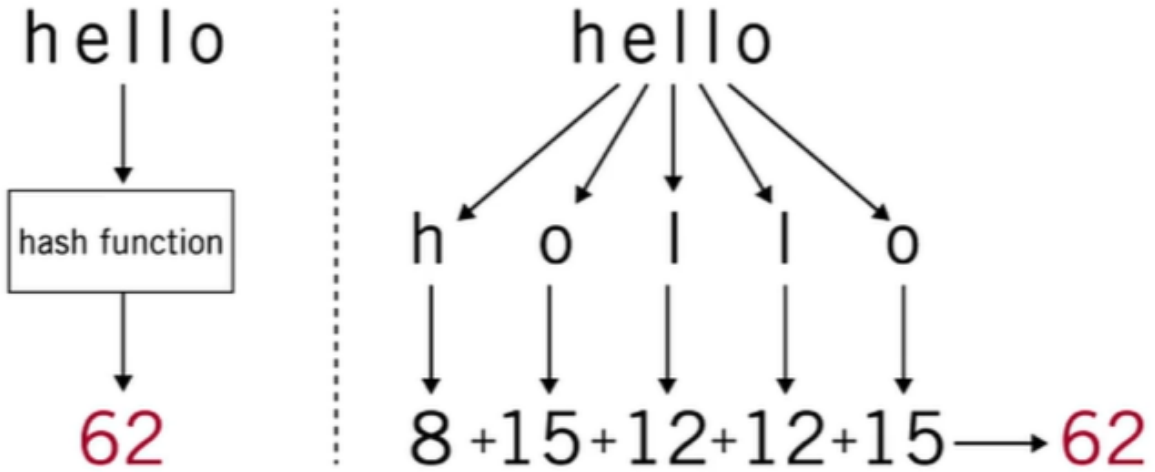
Decrypting The Message (2/2)



Signing The Encrypted Message Instead Of Plain Text



What Is A Hash?



- Hash is like a "finger print" of a document
- If anything in the document changes, the hash changes
- Can't reconstruct the document from the hash
- One way process: Document → Hash

Using A Hash As A Key i.e. (Key, Value)

Hash Function Examples

Let $h(k) = k \% 15$. Then,

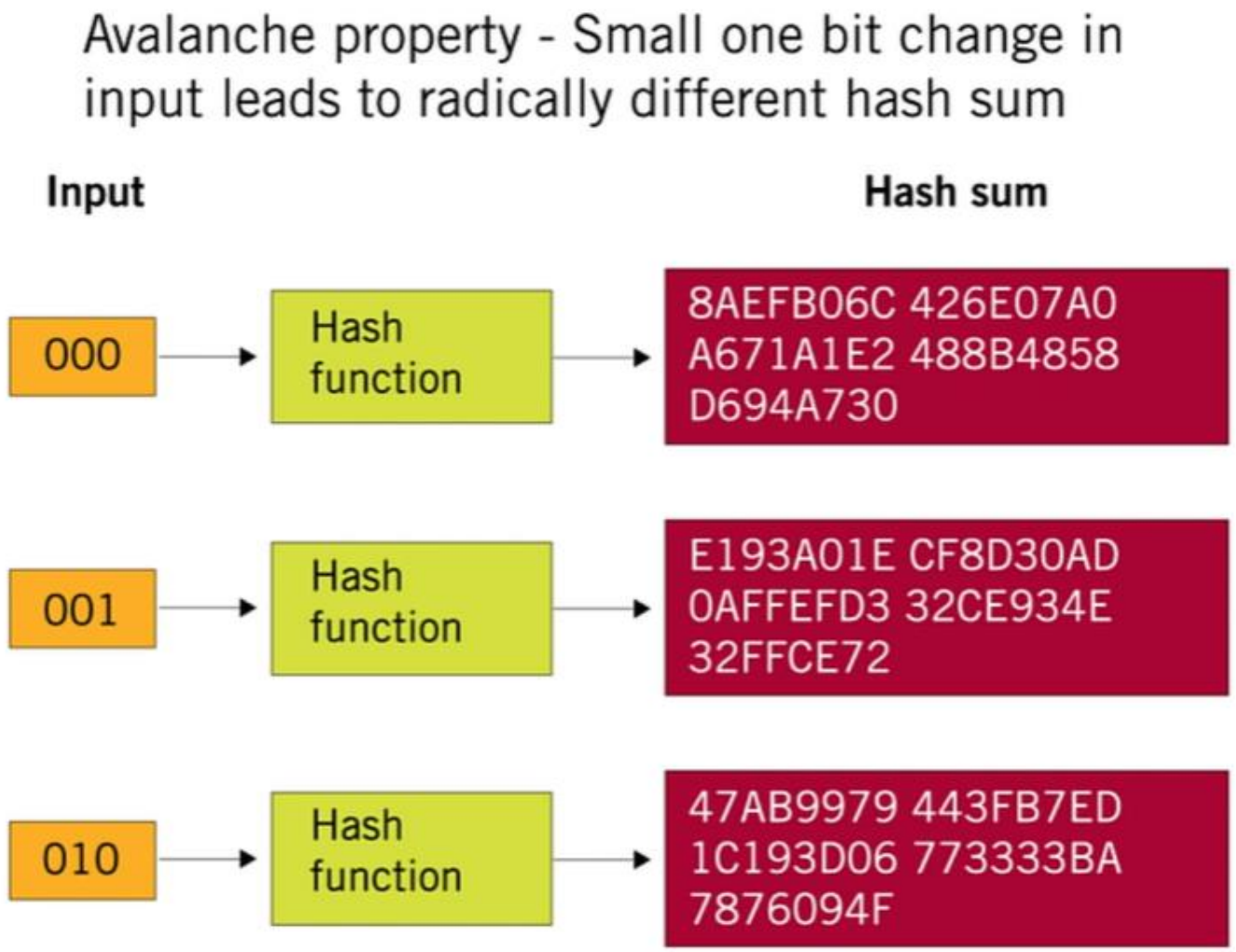
if k =	25	129	35	2501	47	36
$h(k)$ =	10	9	5	11	2	6

Storing the keys in the array is straightforward:

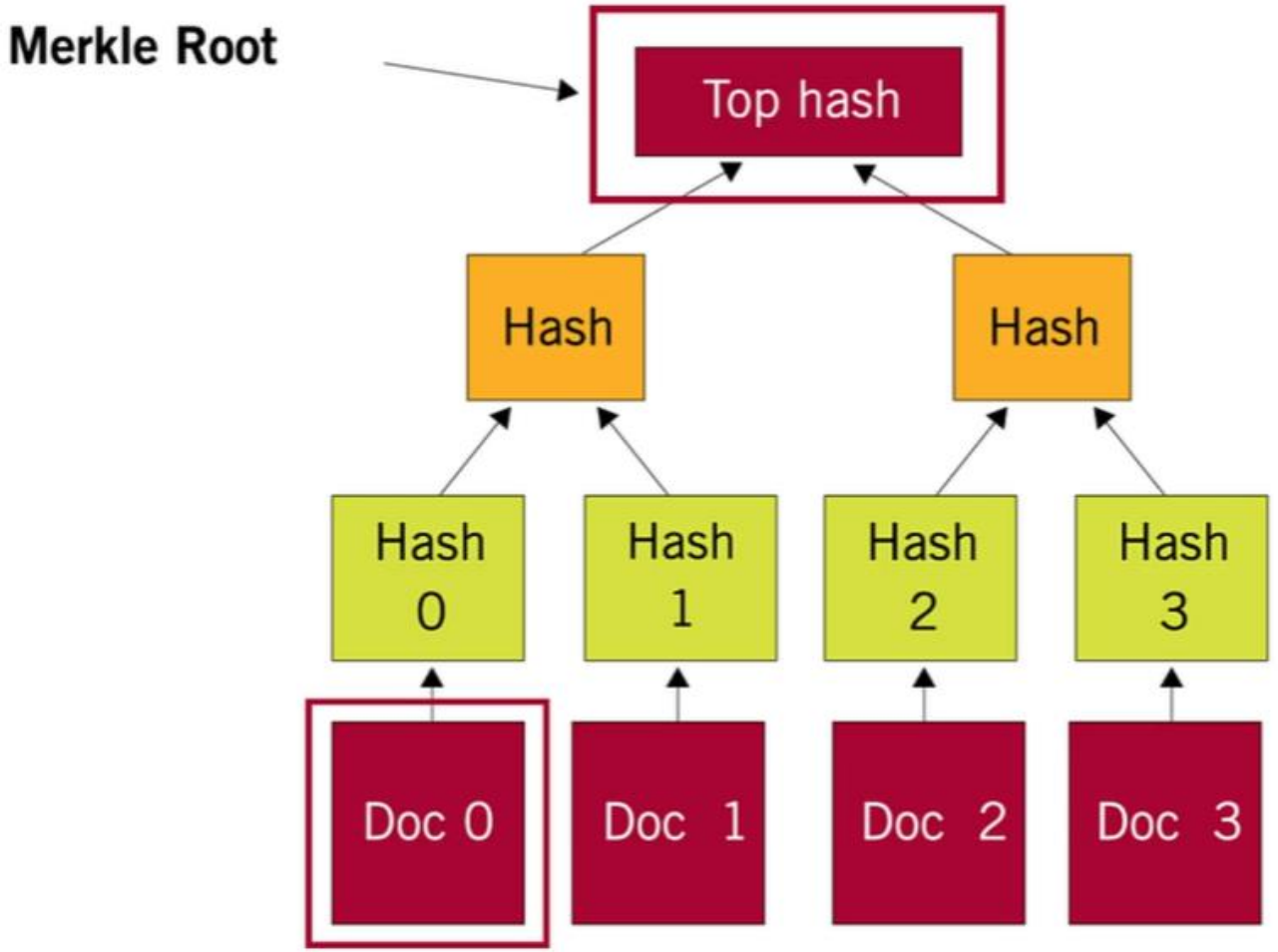
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
_	_	47	_	_	35	36	_	_	129	25	2501	_	_	_

Thus, *delete* and *find* can be done in O(1), and also *insert*

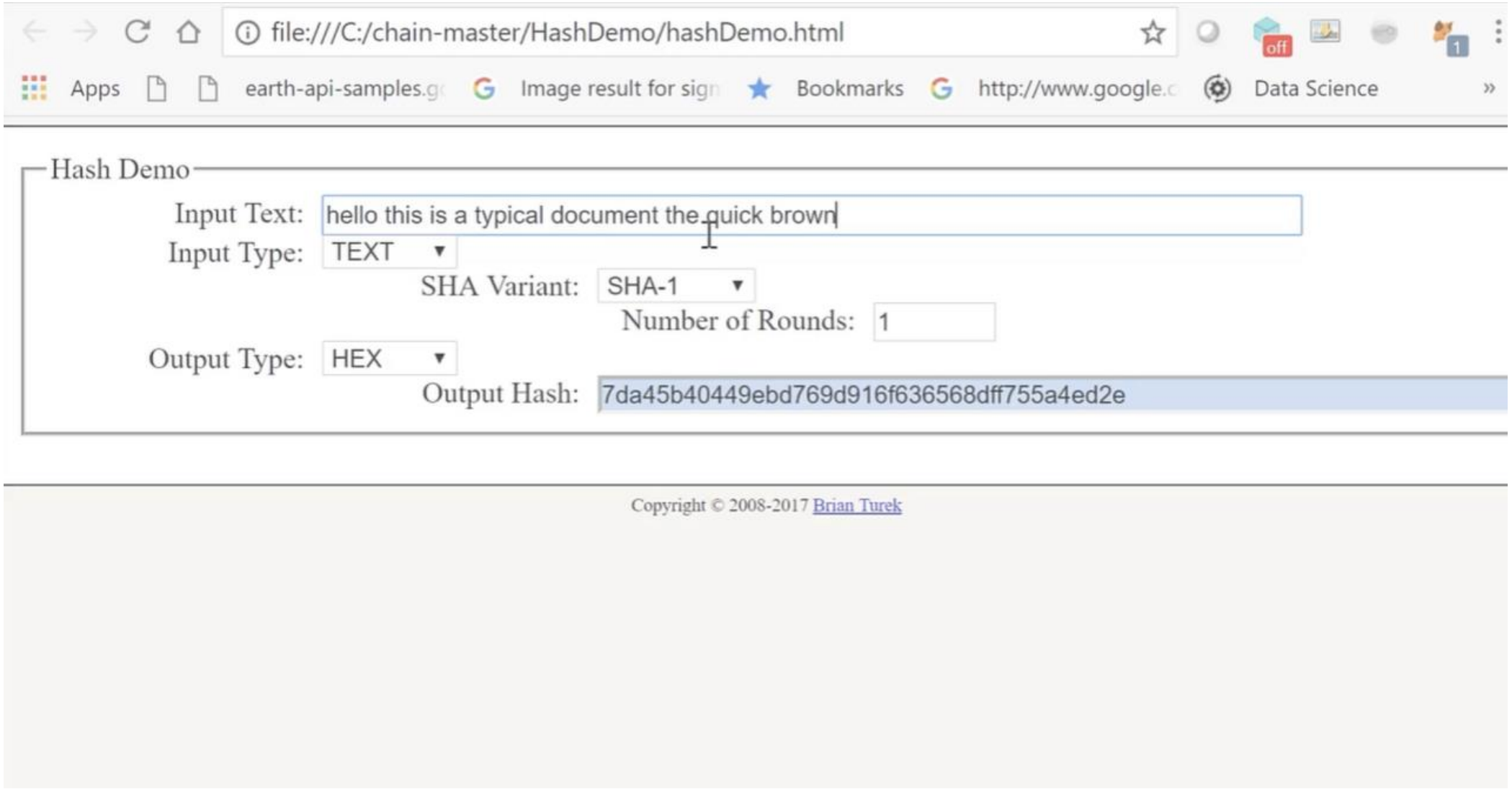
Avalanche Property



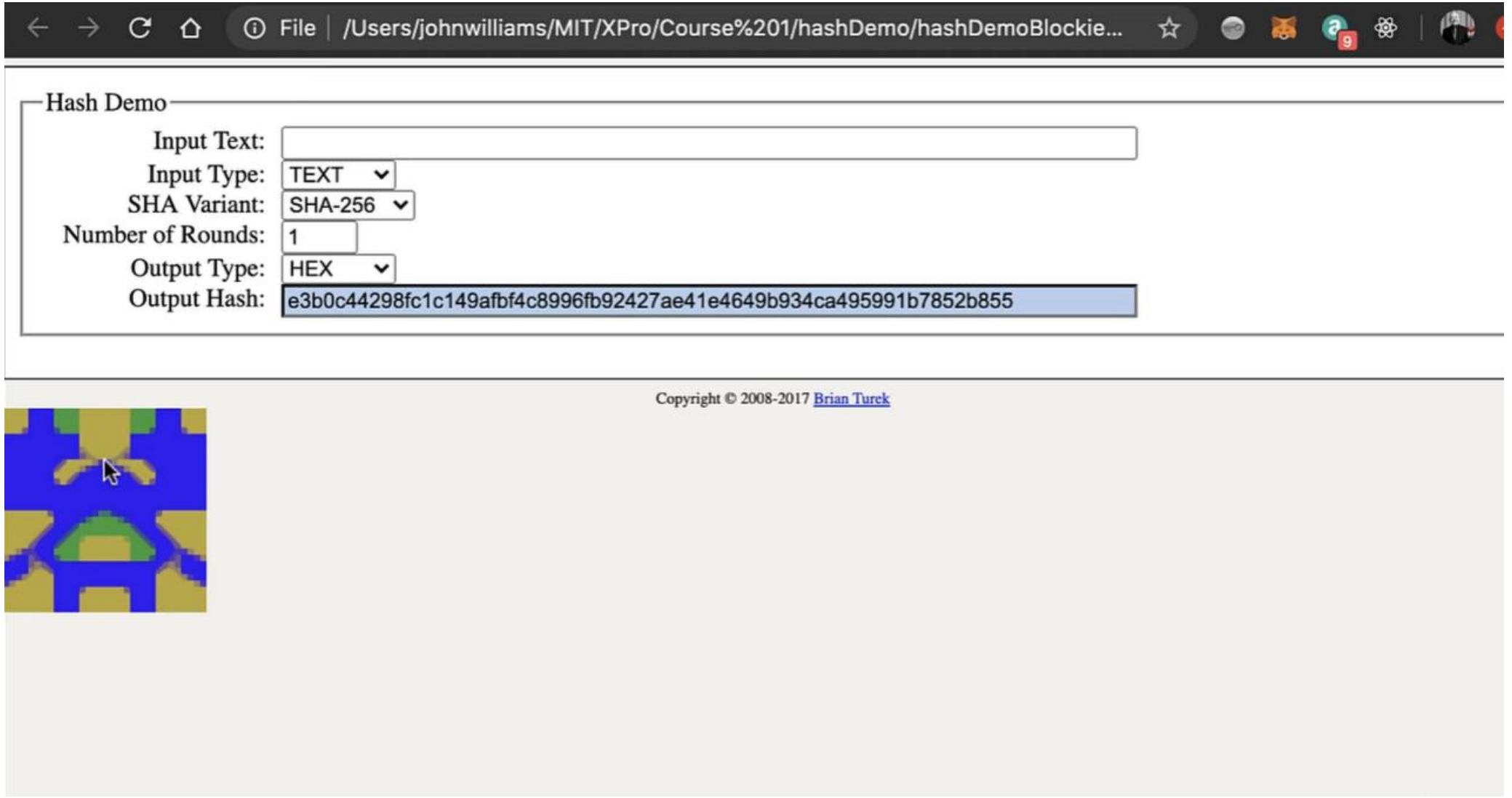
Merkle Tree


Merkle Root
Top hash
Hash
Hash
Hash 0
Hash 1
Hash 2
Hash 3
Doc 0
Doc 1
Doc 2
Doc 3

Hash Function Demo

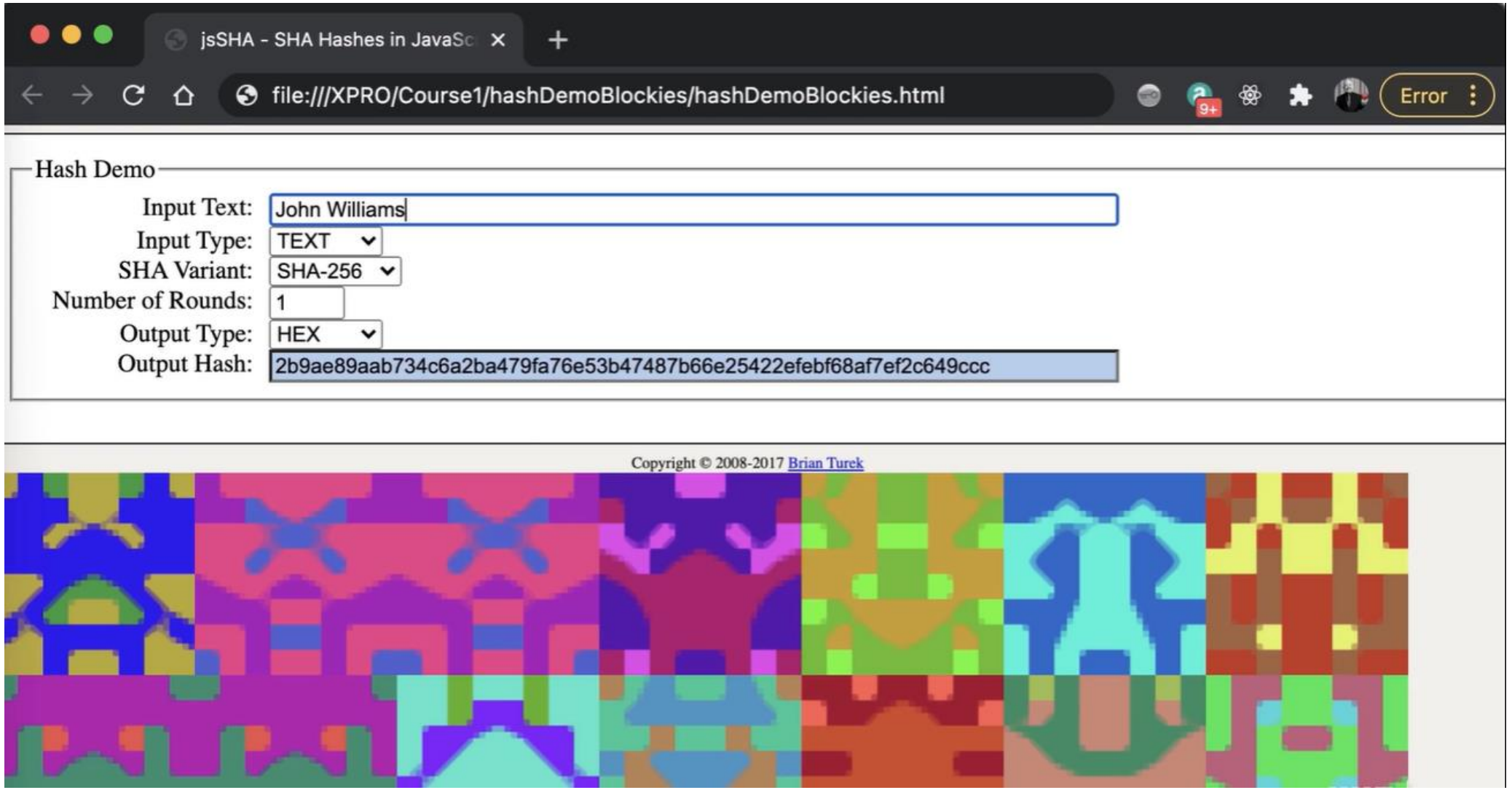


Hash Function: Blockie (1/2)

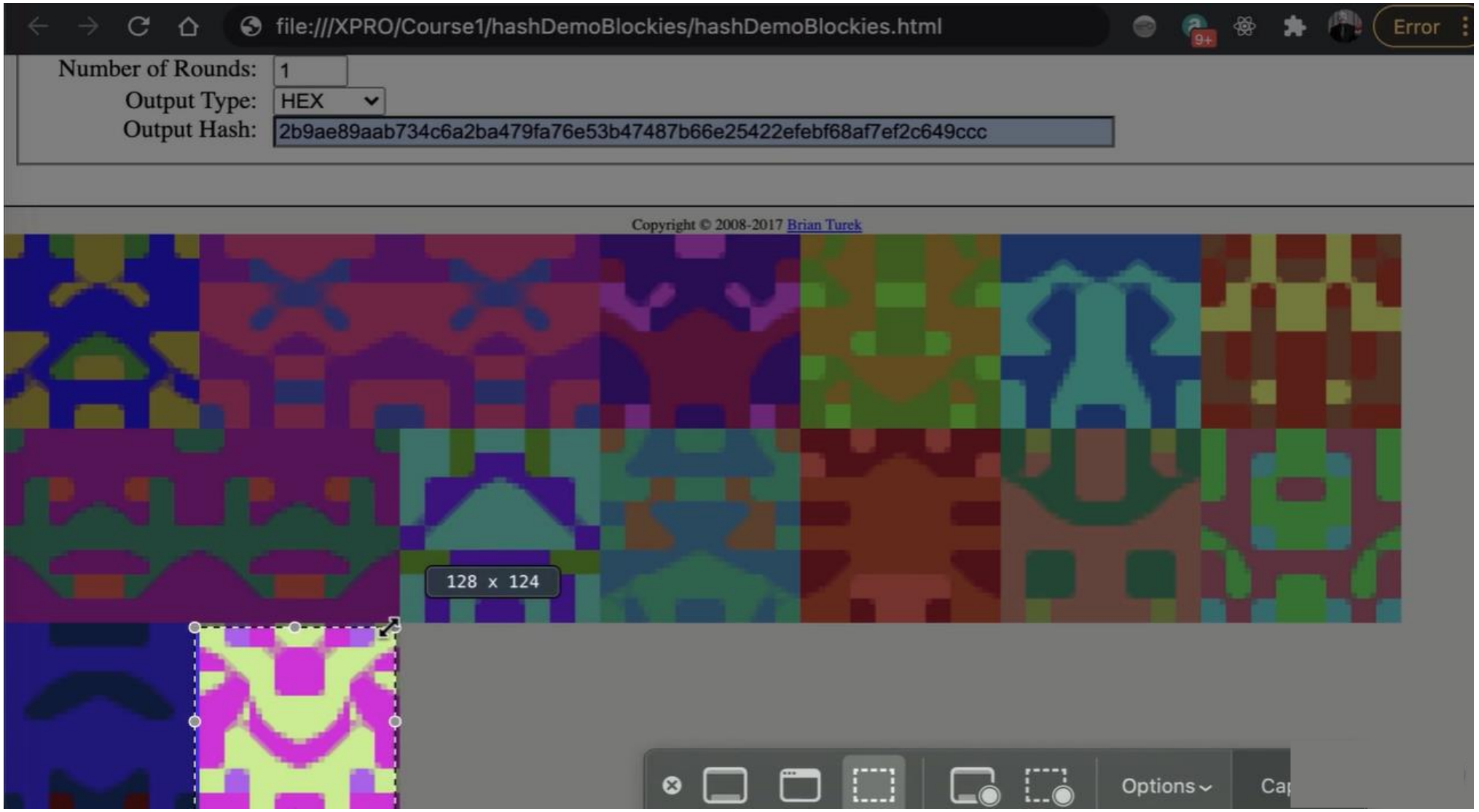


Hash Function: Blockie (2/2)

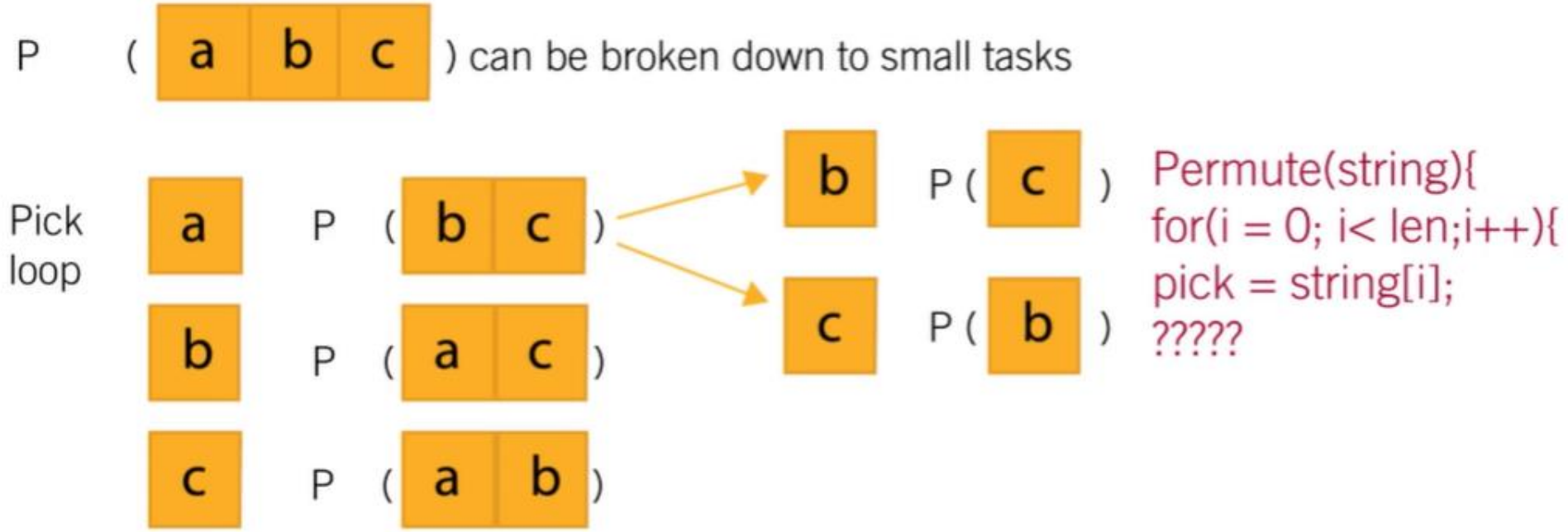
Creating Blockie (1/2)



Creating Blockie (2/2)

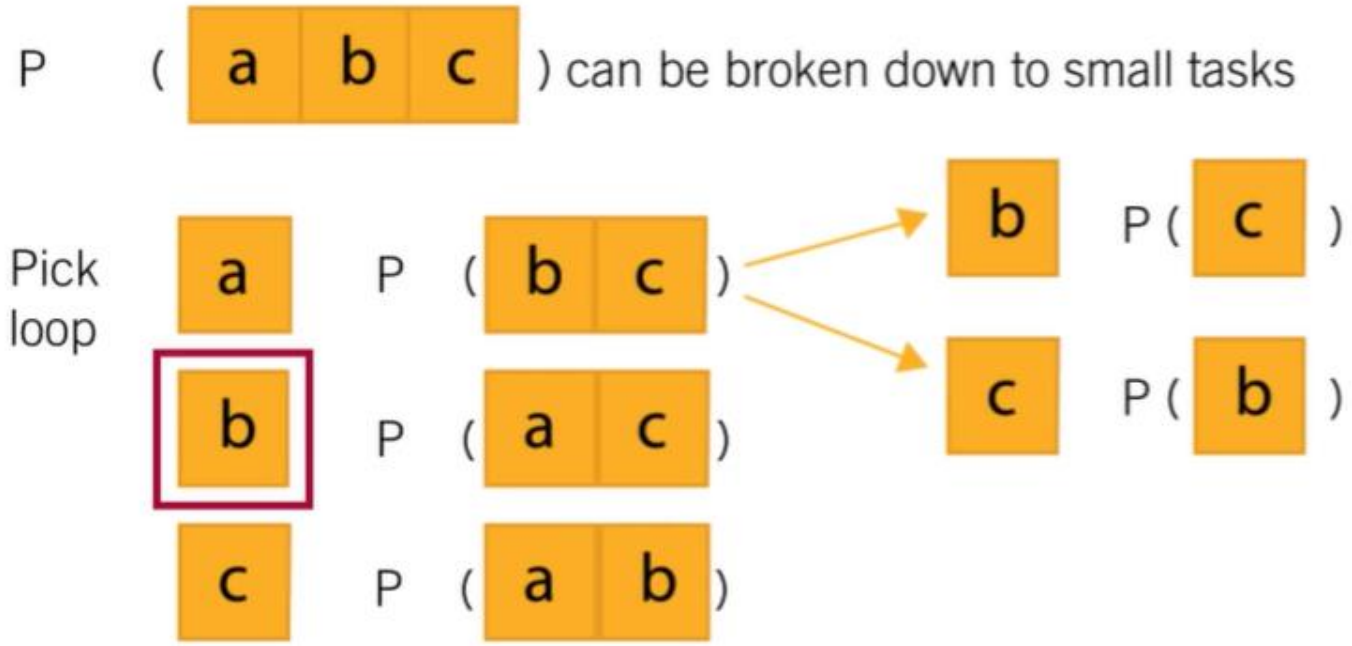


Permute Letters And Print Them All Out (1/12)



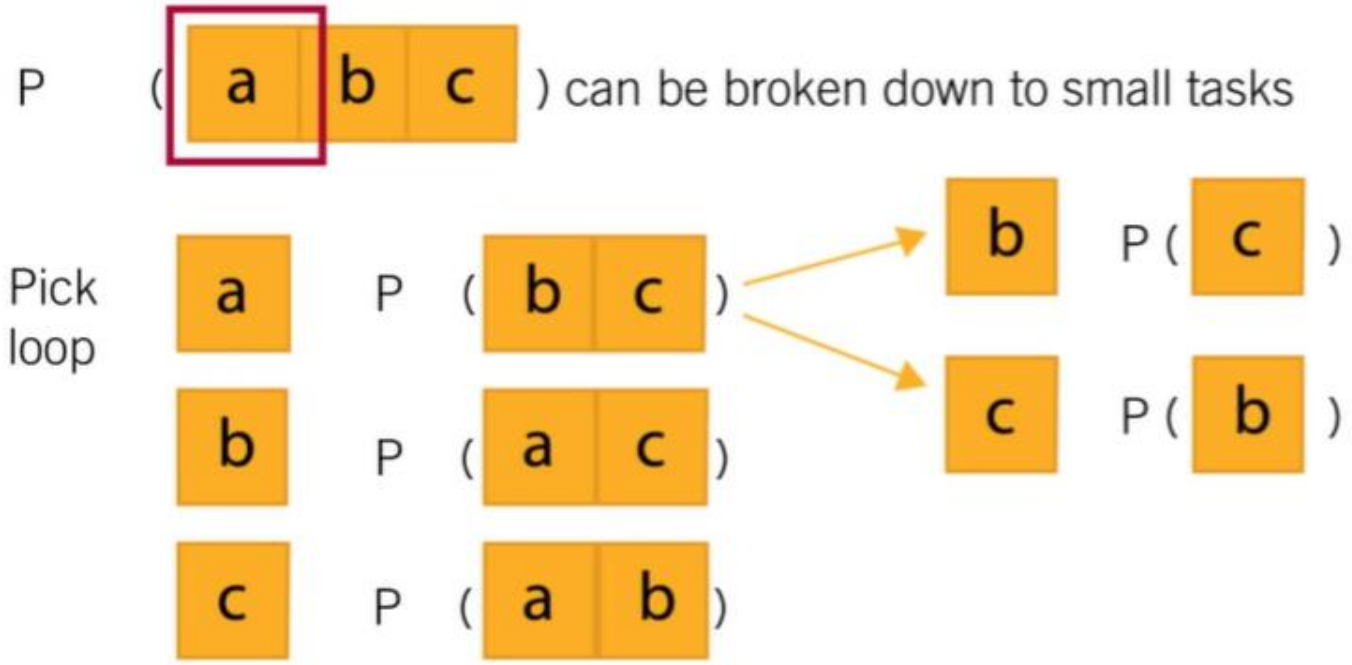
Permute Letters And Print Them All Out (2/12)

Permute Letters And Print Them All Out (3/12)



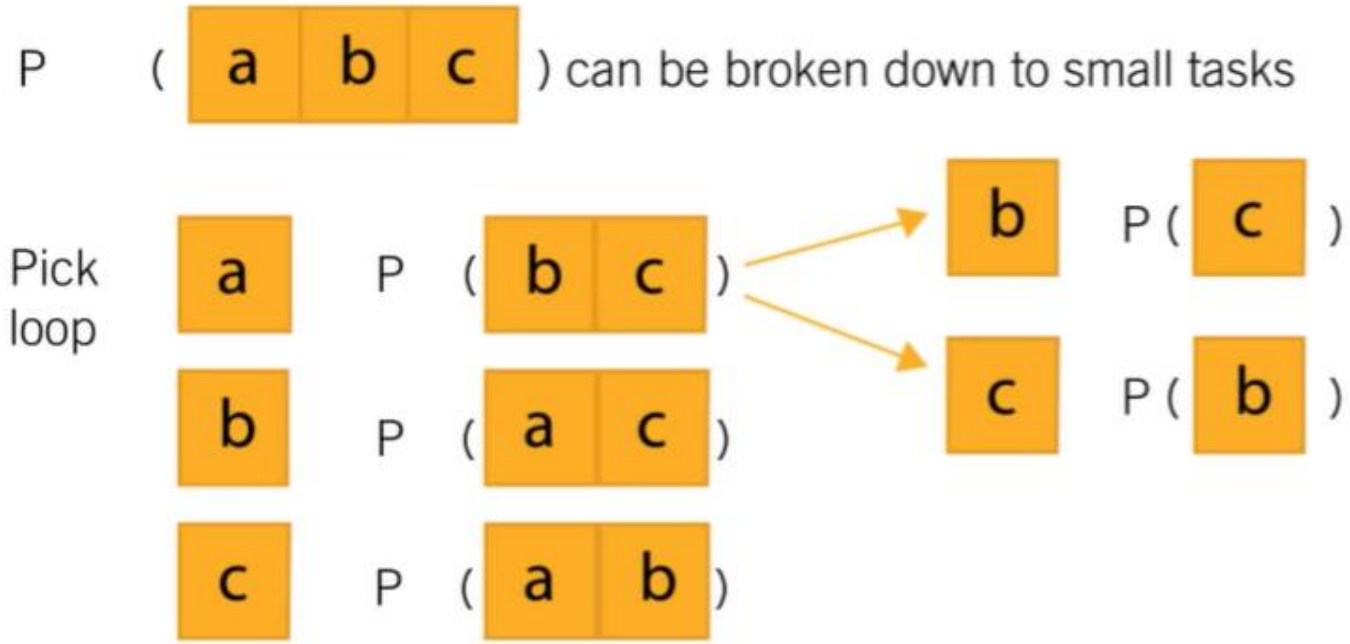

```
Permute(string){
for(i = 0; i< len;i++){
pick = string[i];
remainder =
string.slice(0,i)+string(i+1,len)
For (var perm of permute(remainder)){
permutations. push(pick + perm);}
return permutations;}
```

Permute Letters And Print Them All Out (4/12)



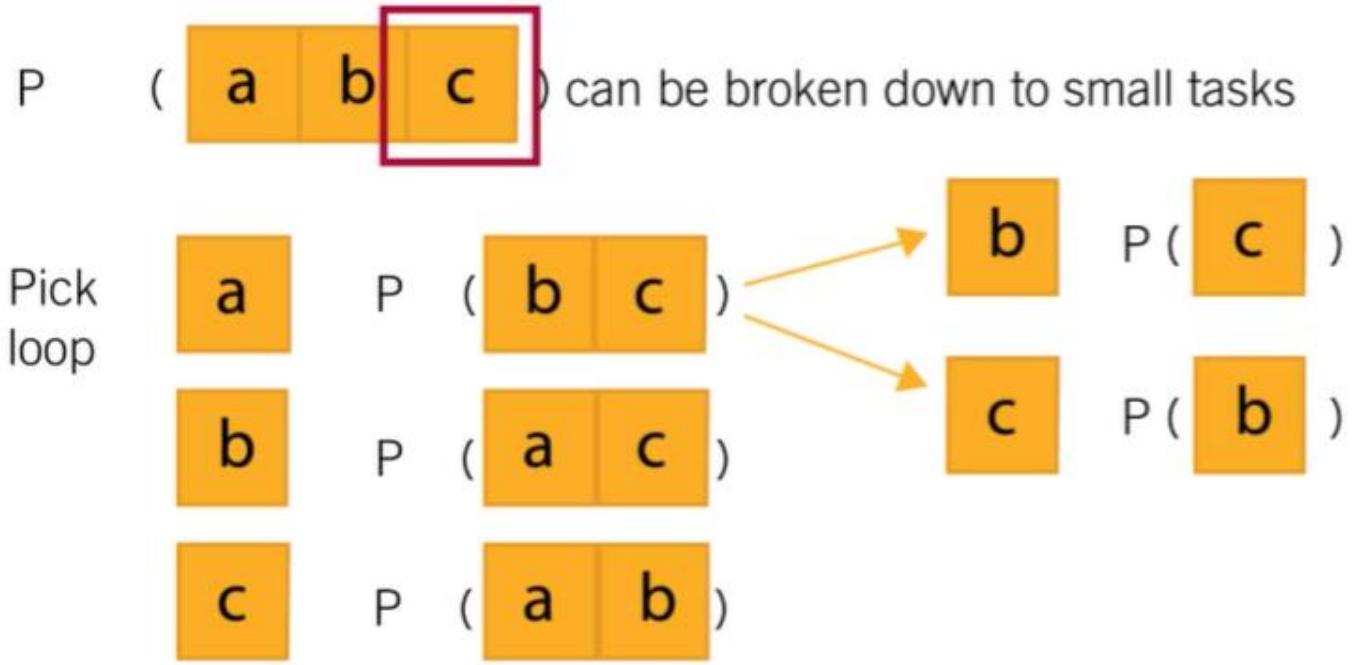

```
Permute(string){
for(i = 0; i< len;i++){
pick = string[i];
remainder =
string.slice(0,i)+string(i+1,len)
For (var perm of permute(remainder)){
permutations. push(pick + perm);}
return permutations;}
```

Permute Letters And Print Them All Out (5/12)



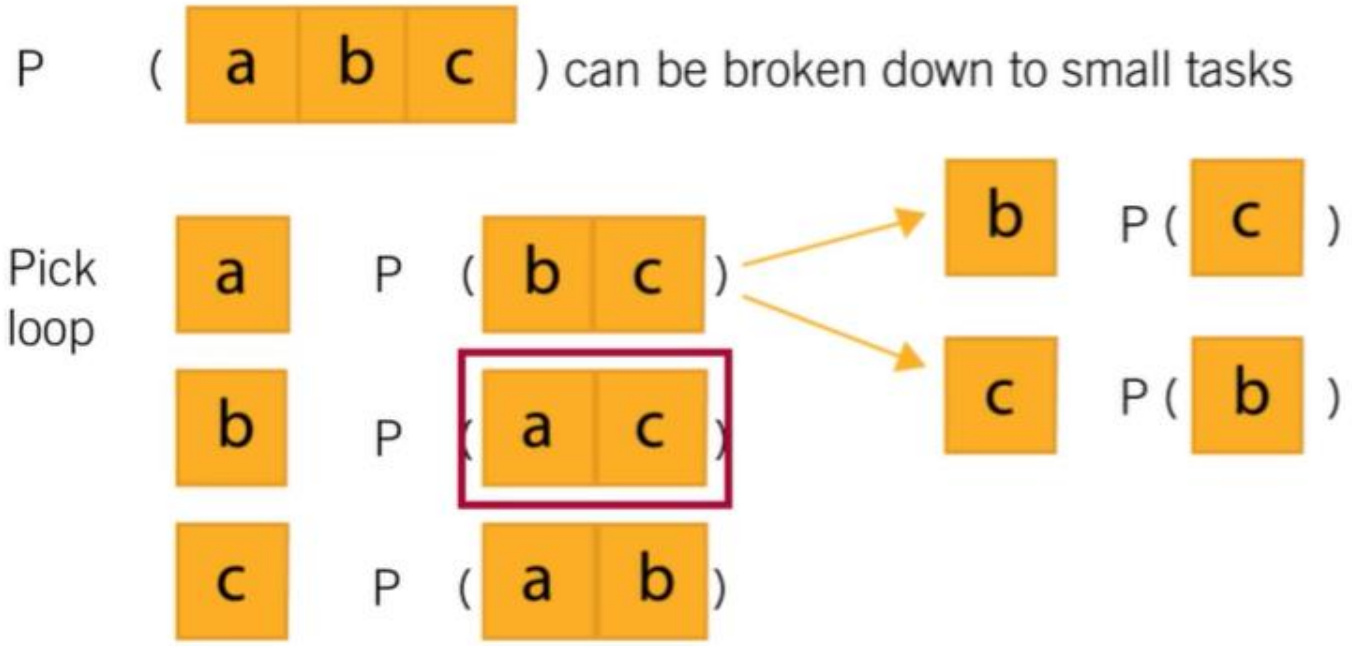

```
Permute(string){
for(i = 0; i< len;i++){
pick = string[i];
remainder =
string.slice(0,i)+string(i+1,len)
For (var perm of permute(remainder)){
permutations. push(pick + perm);}
return permutations;}
```

Permute Letters And Print Them All Out (6/12)



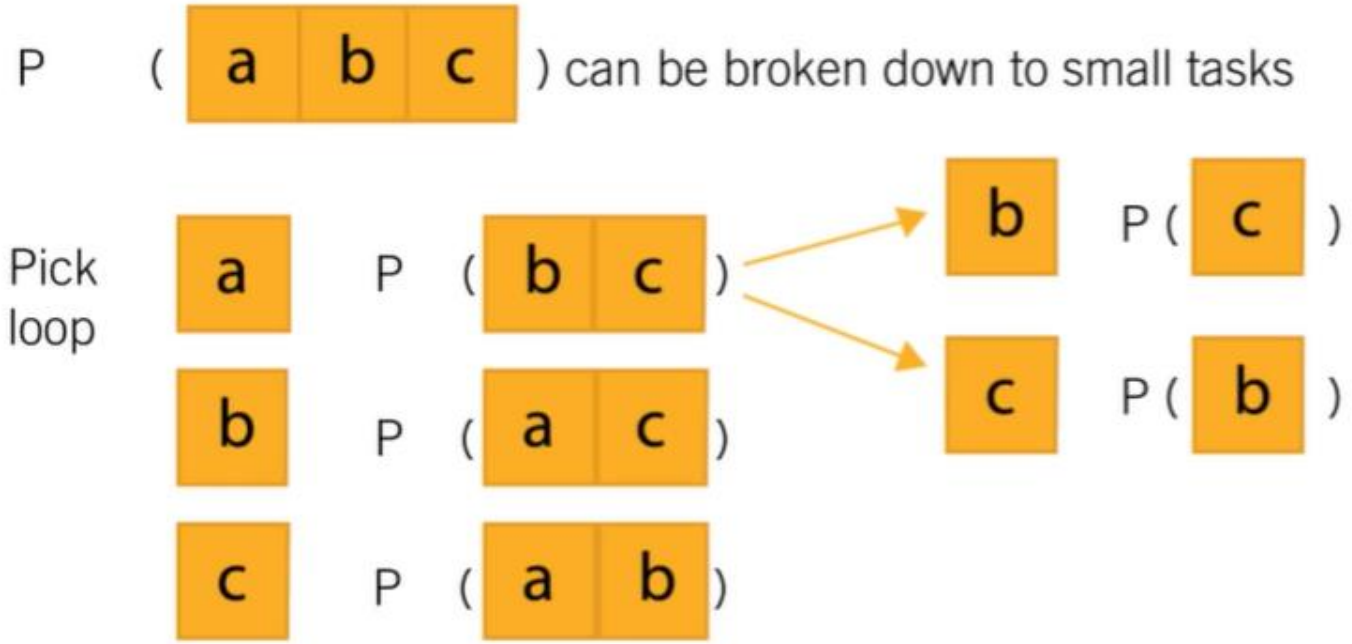

```
Permute(string){
for(i = 0; i< len;i++){
pick = string[i];
remainder =
string.slice(0,i)+string(i+1,len)
For (var perm of permute(remainder)){
permutations. push(pick + perm);}
return permutations;}
```

Permute Letters And Print Them All Out (7/12)



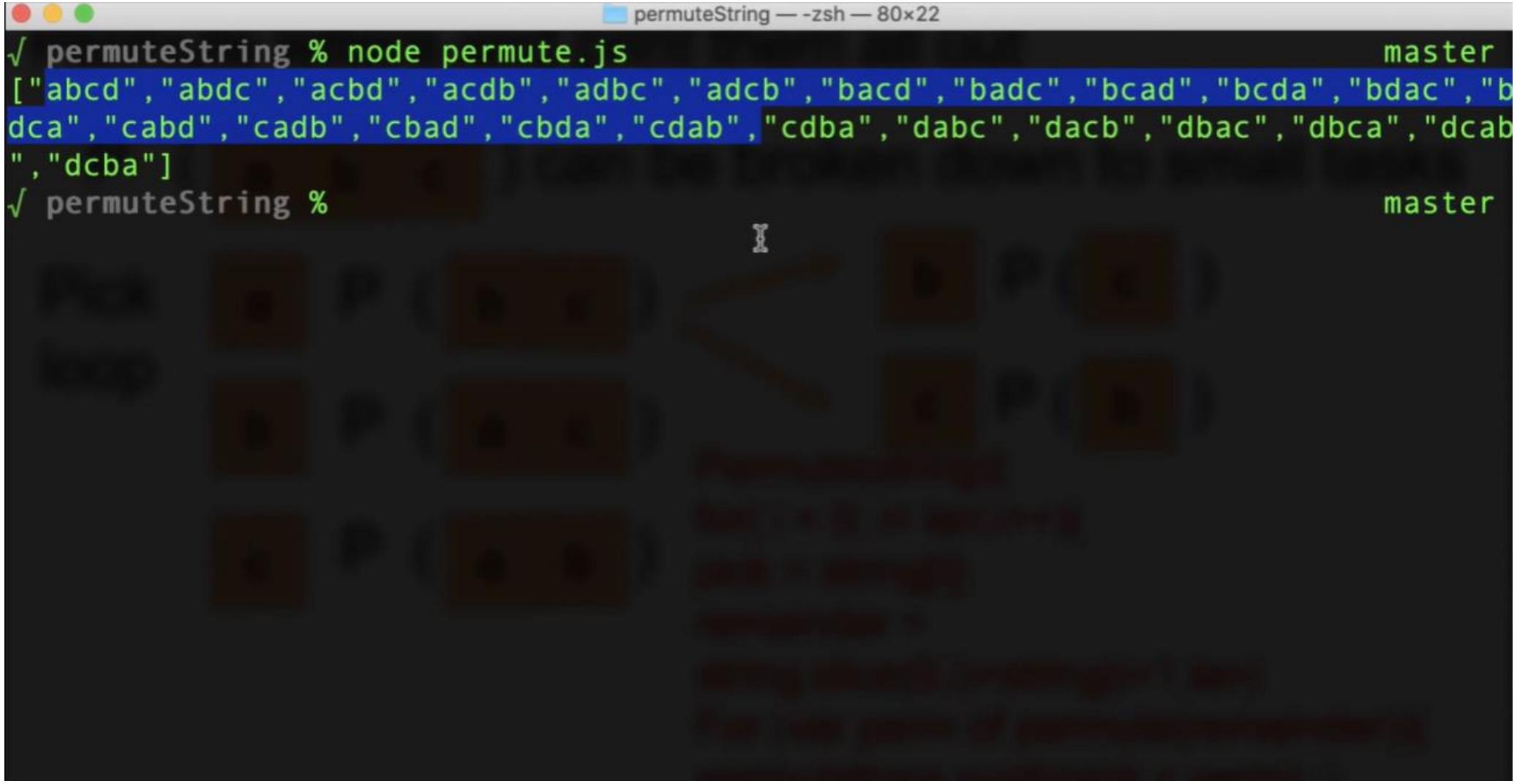

```
Permute(string){
for(i = 0; i< len;i++){
pick = string[i];
remainder =
string.slice(0,i)+string(i+1,len)
For (var perm of permute(remainder)){
permutations. push(pick + perm);}
return permutations;}
```

Permute Letters And Print Them All Out (8/12)

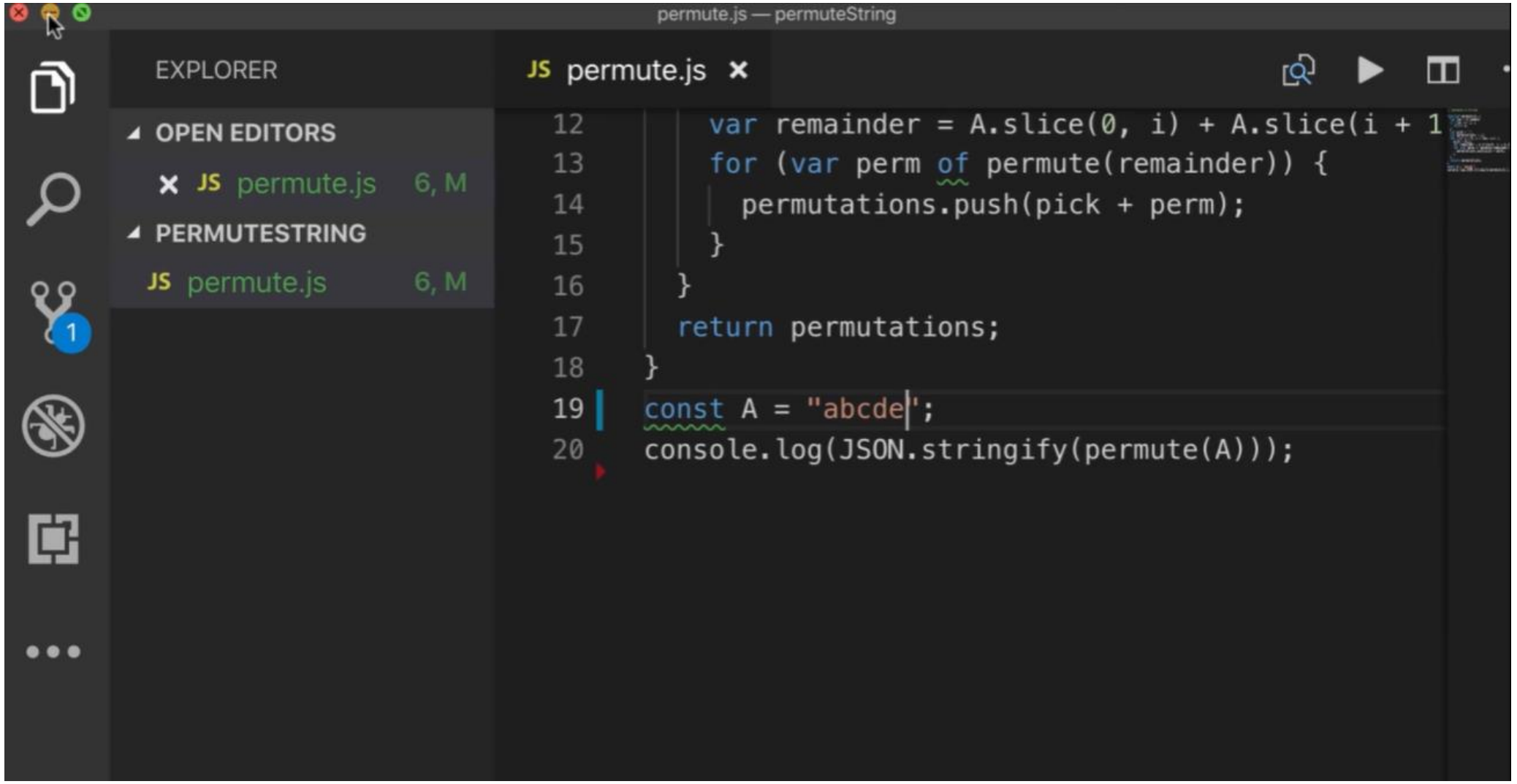



```
Permute(string){
for(i = 0; i< len;i++){
pick = string[i];
remainder =
string.slice(0,i)+string(i+1,len)
For (var perm of permute(remainder)){
permutations. push(pick + perm);}
return permutations;}
```

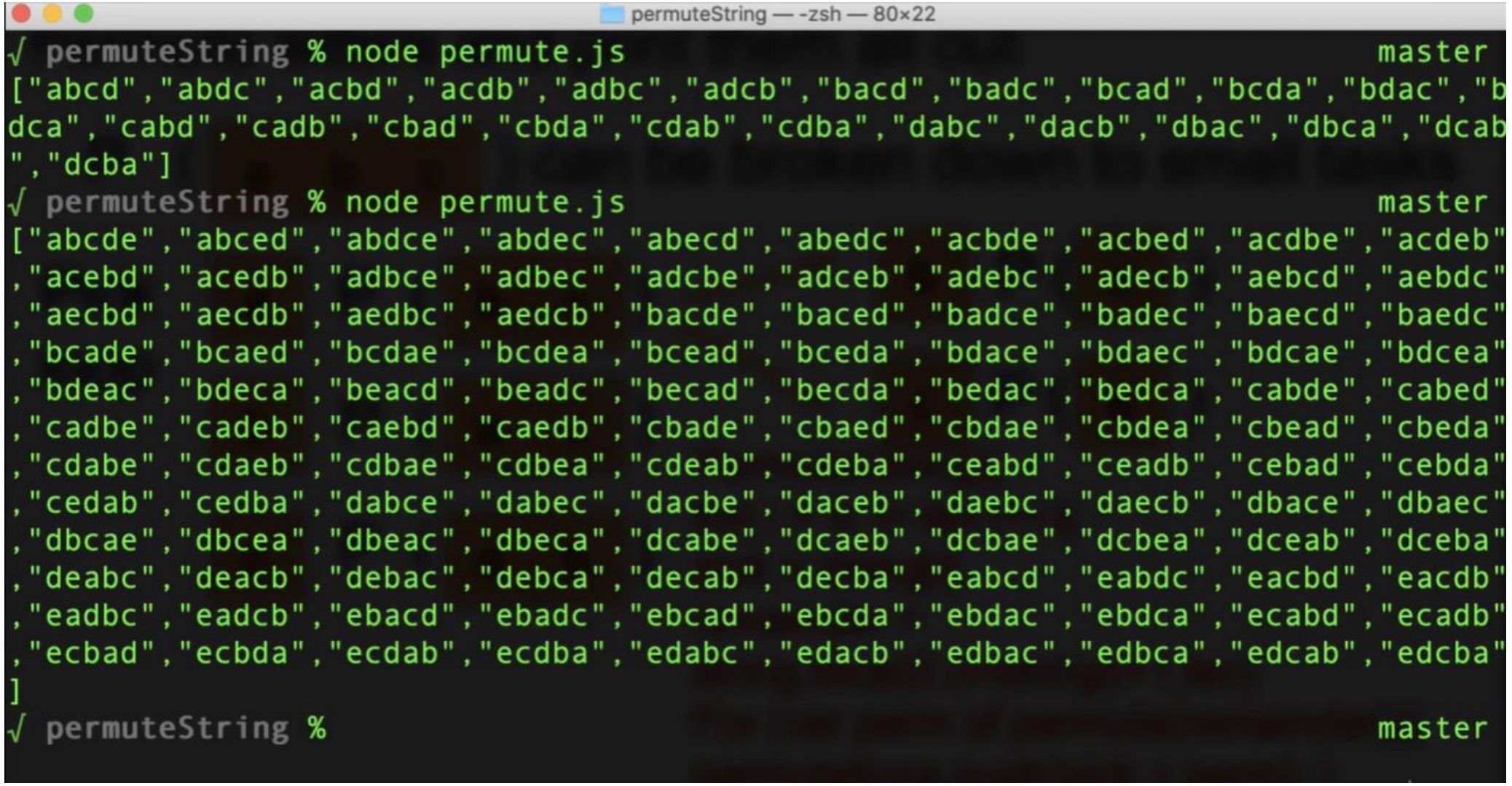
Permute Letters And Print Them All Out (9/12)



Permute Letters And Print Them All Out (10/12)



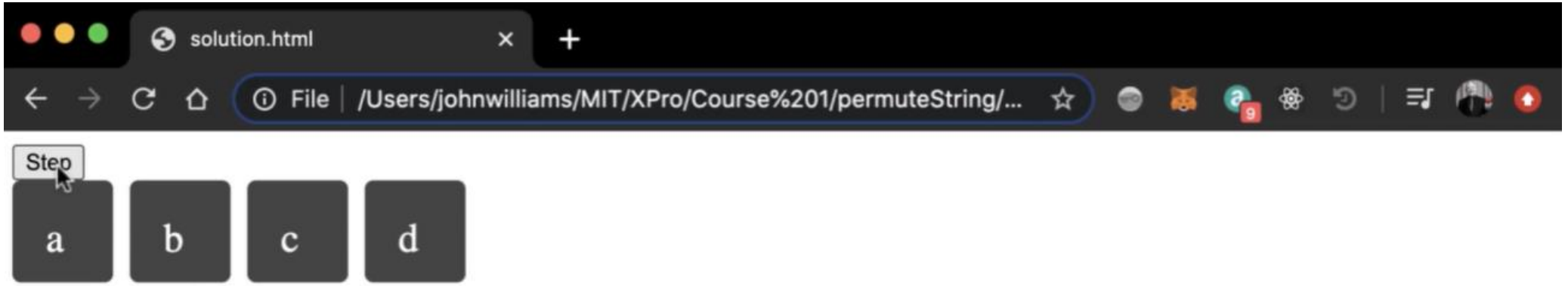
Permute Letters And Print Them All Out (11/12)



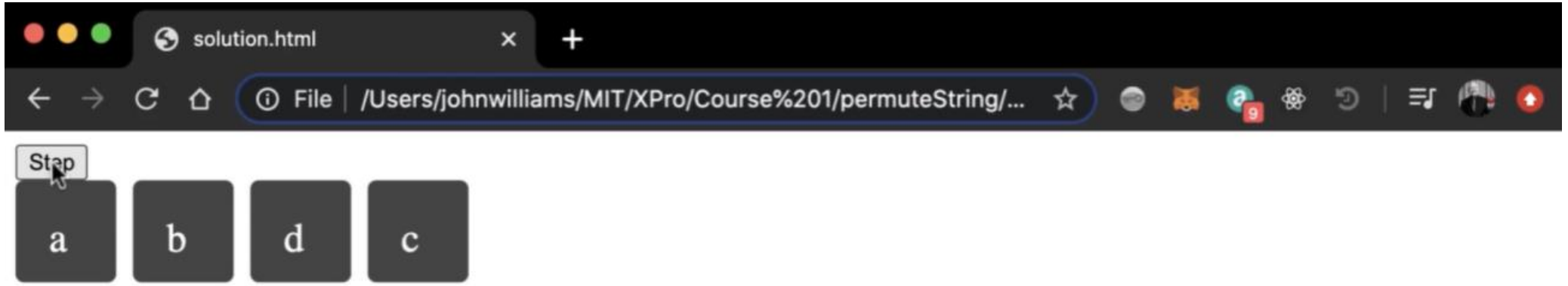
Permute Letters And Print Them All Out (12/12)

```
// permute a string
function permute(A) {
  let len = A.length;
  if (len === 1) return A;
  let pick = "";
  let permutations = [];
  for (let i = 0; i < len; i++) {
    pick = A[i];
    var remainder = A.slice(0, i) + A.slice(i + 1, len);
    for (var perm of permute(remainder)) {
      permutations.push(pick + perm);
    }
  }
  return permutations;
}
const A = "abcde";
console.log(JSON.stringify(permute(A)));
```

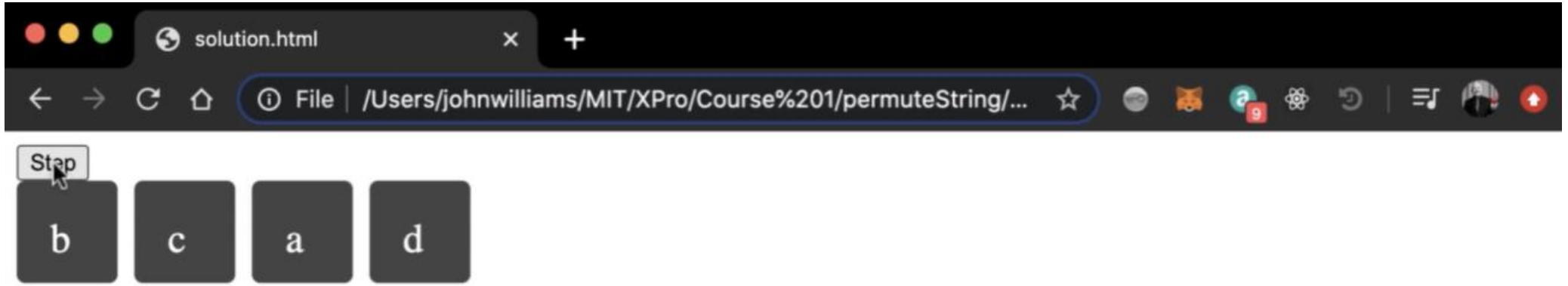
Permute Exercise (1/8)



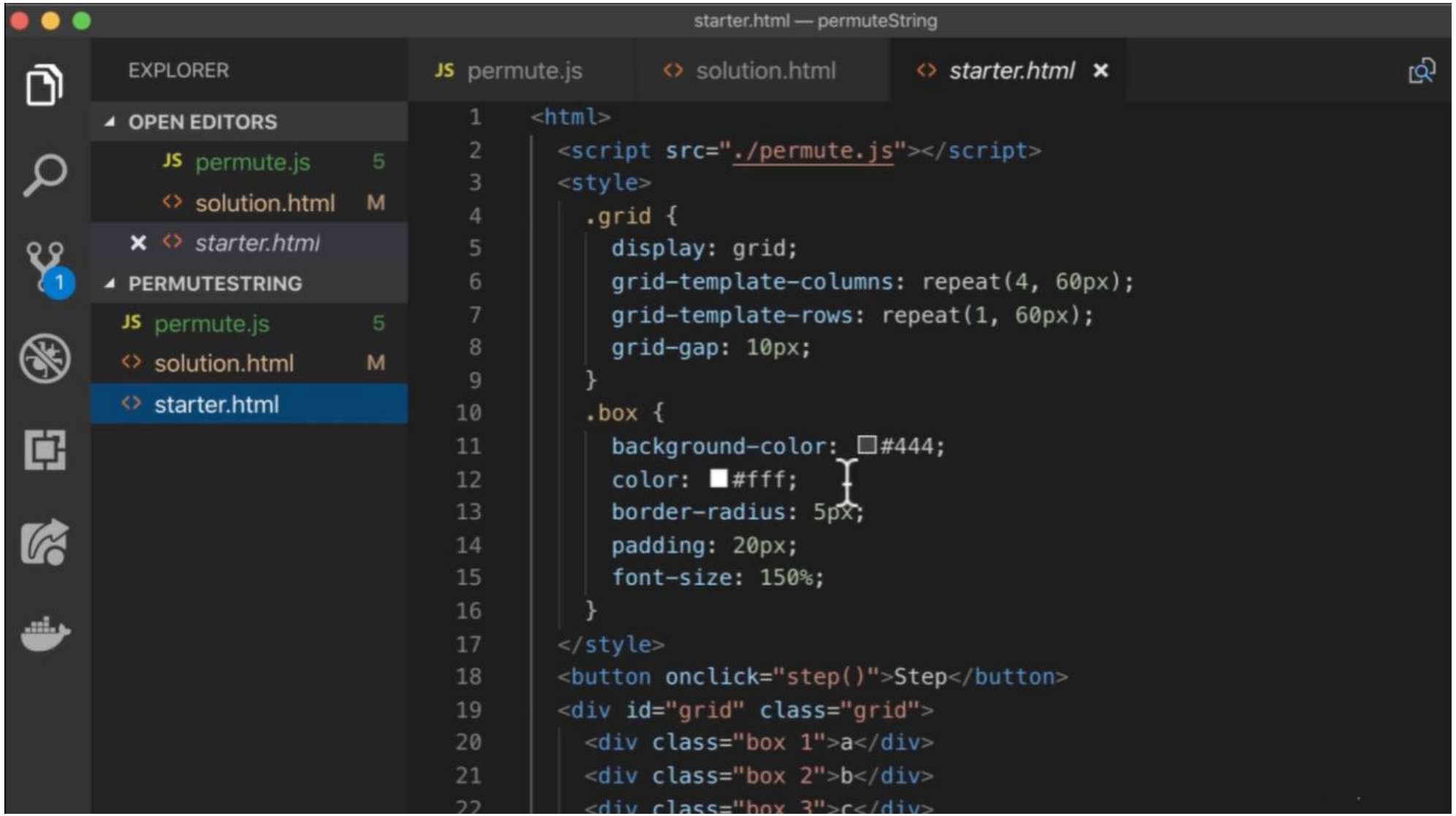
Permute Exercise (2/8)



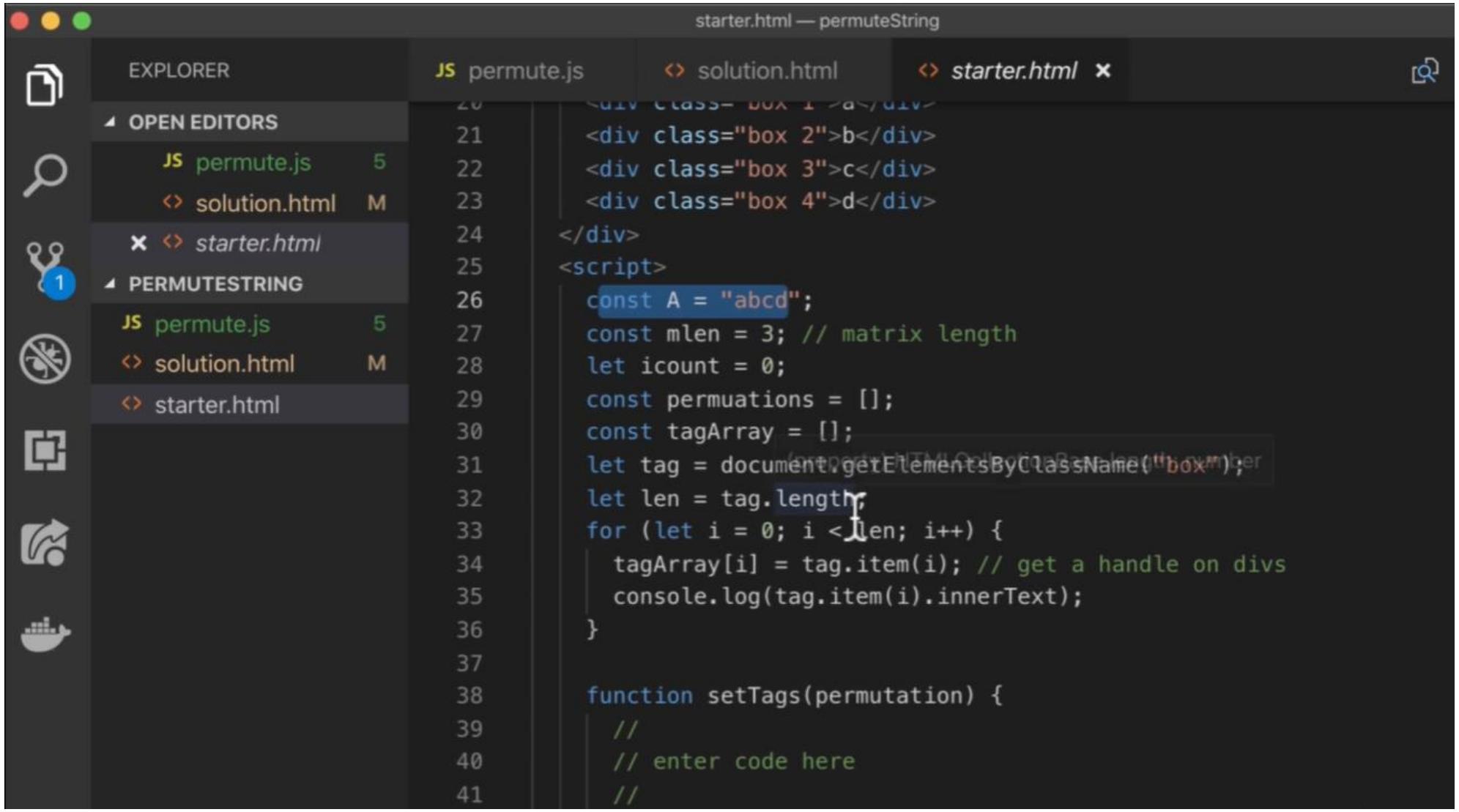
Permute Exercise (3/8)



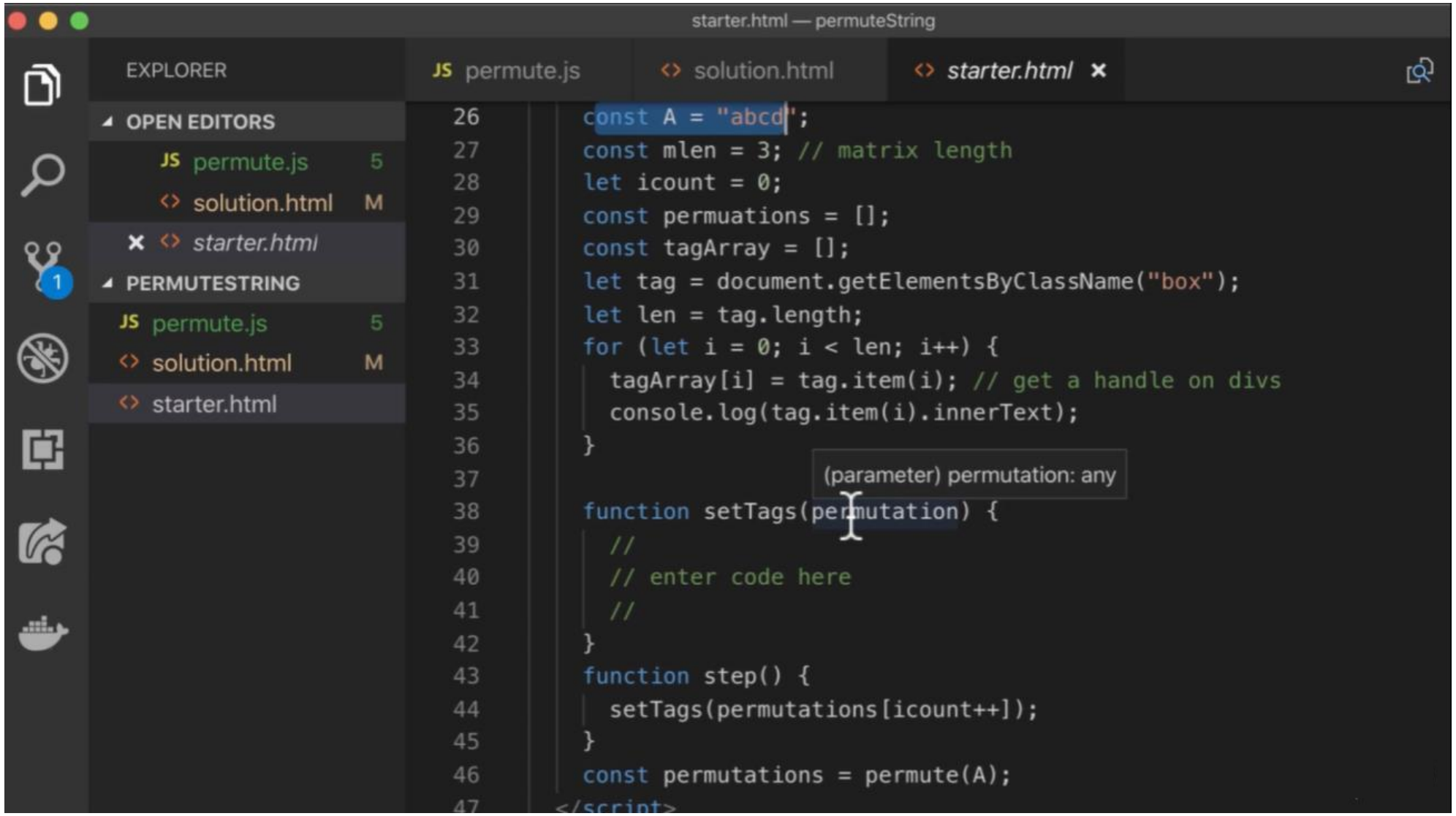
Permute Exercise (4/8)



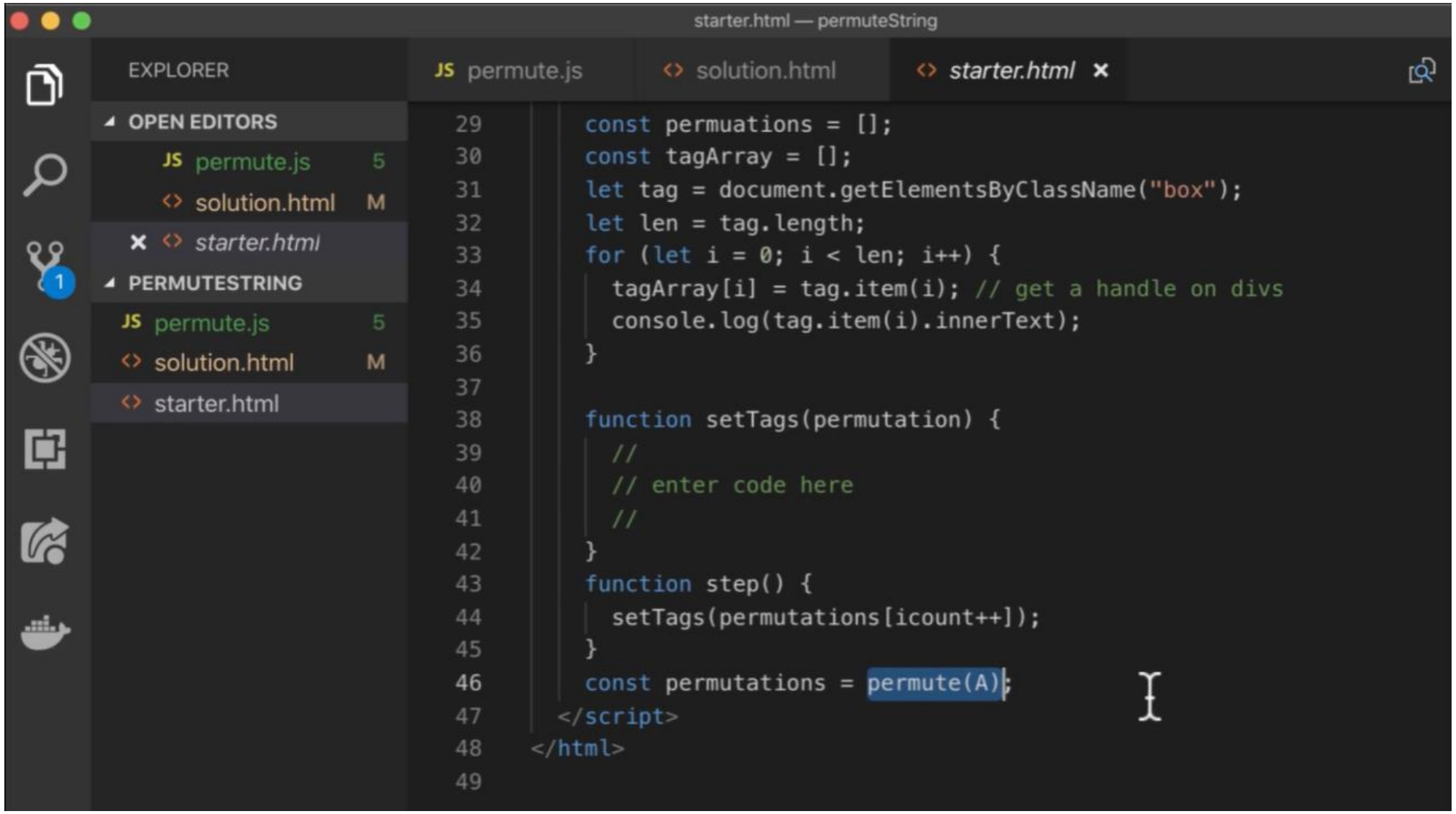
Permute Exercise (5/8)



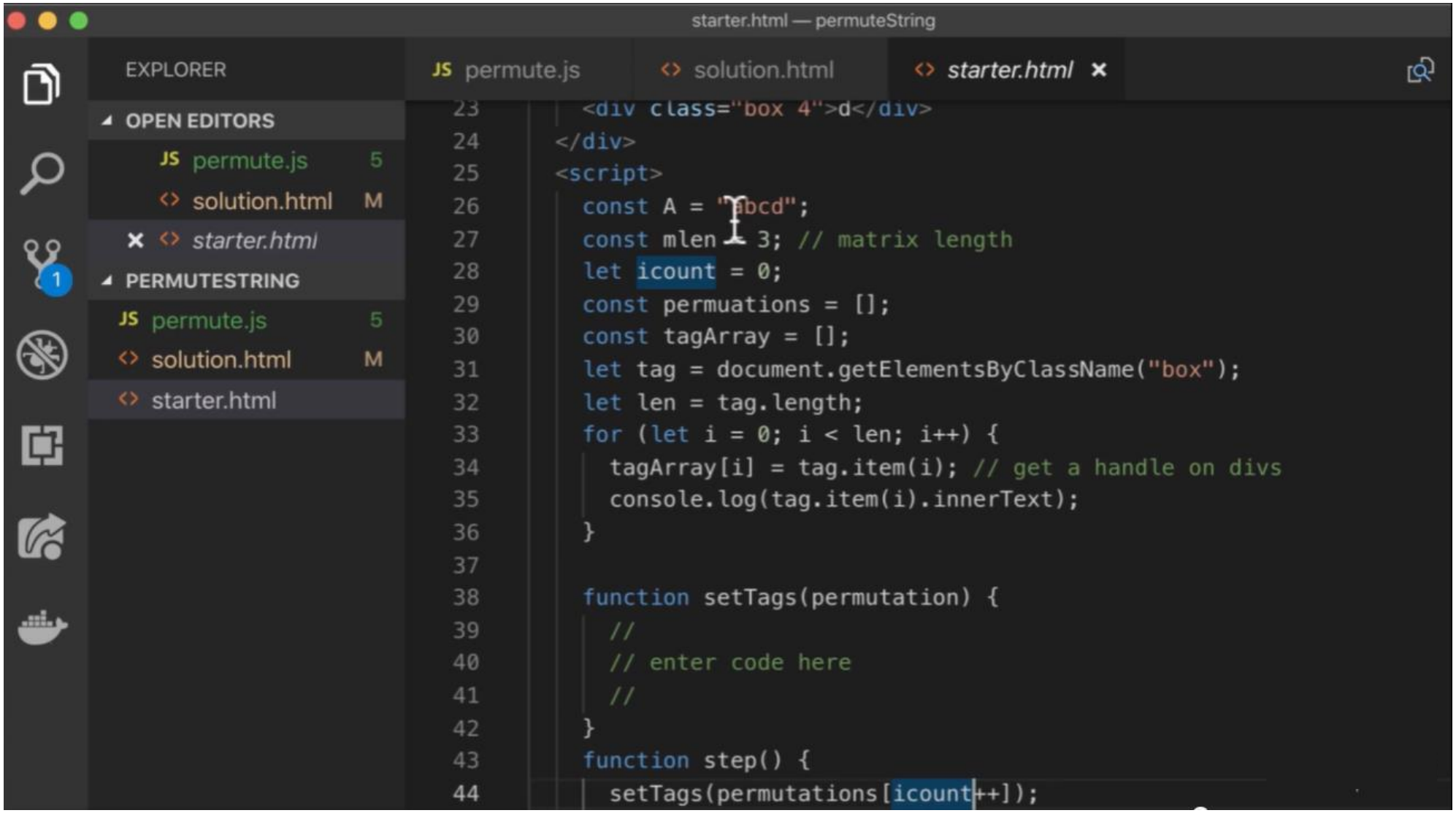
Permute Exercise (6/8)



Permute Exercise (7/8)

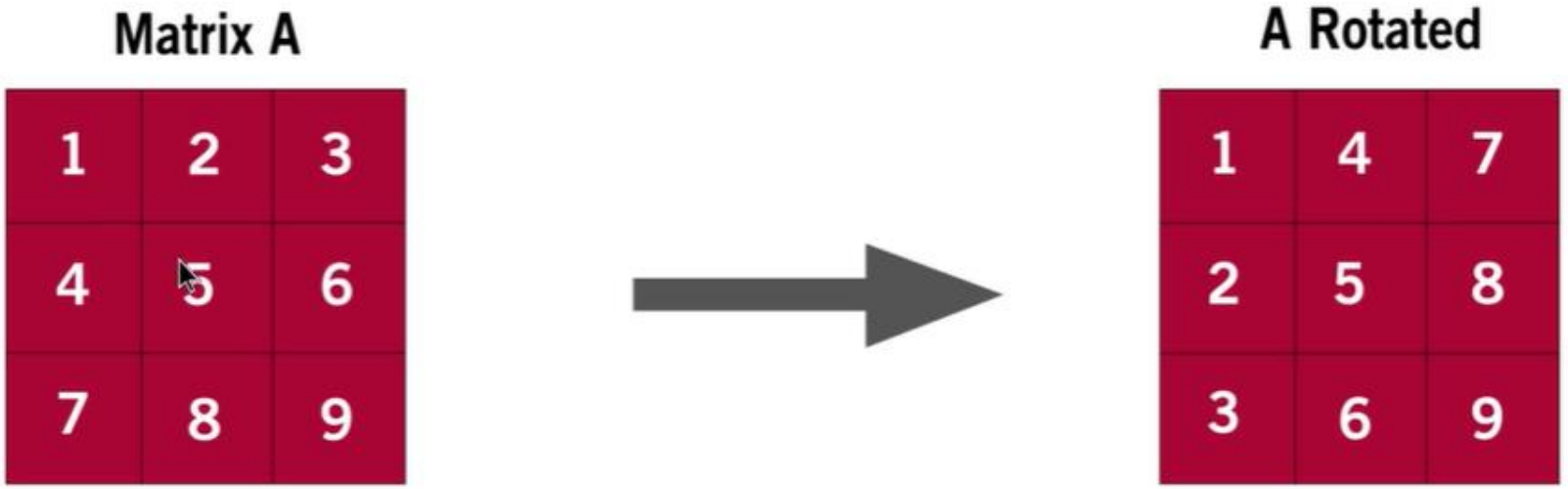


Permute Exercise (8/8)

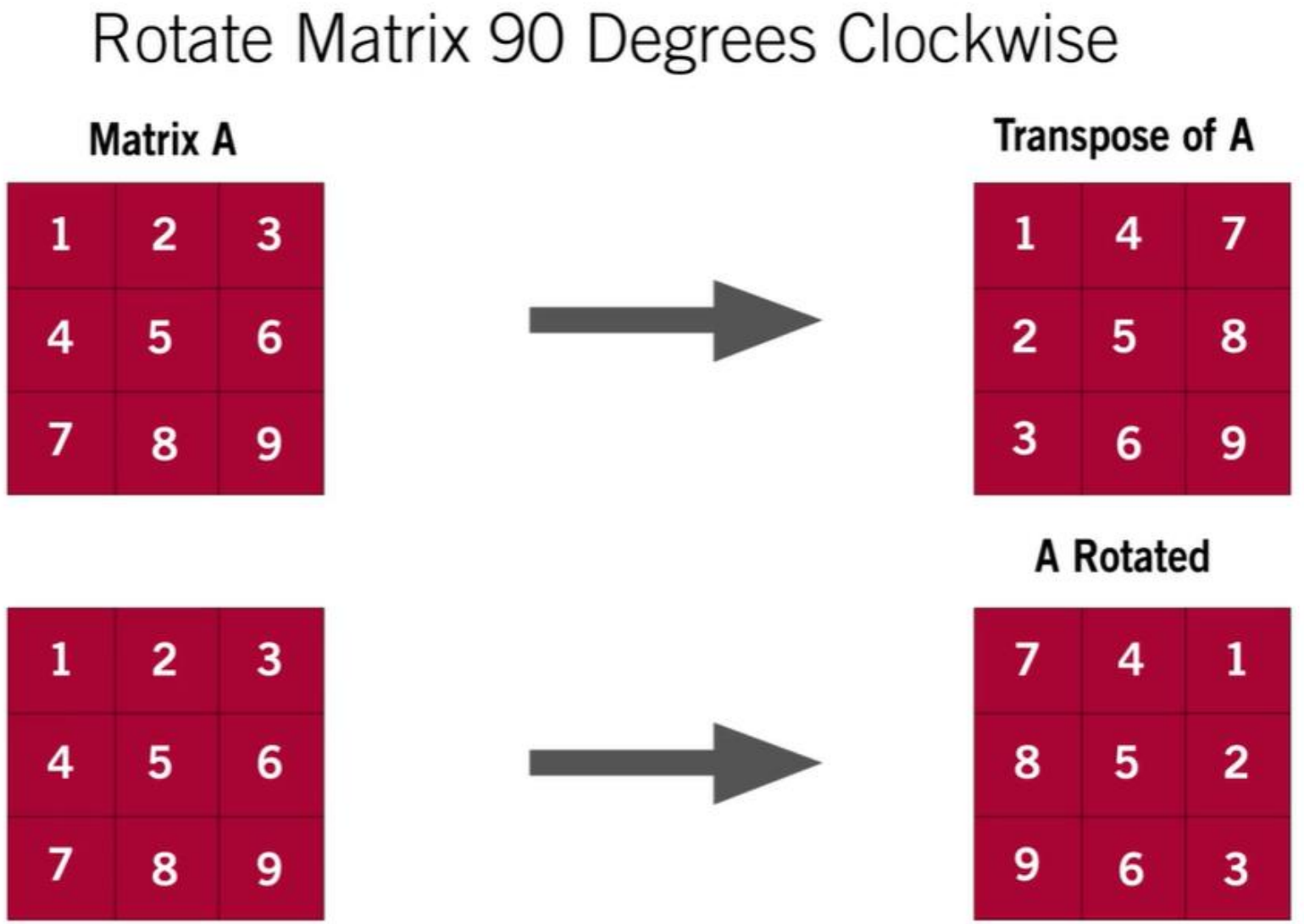


Rotate An Image 90 Degrees Clockwise (1/2)

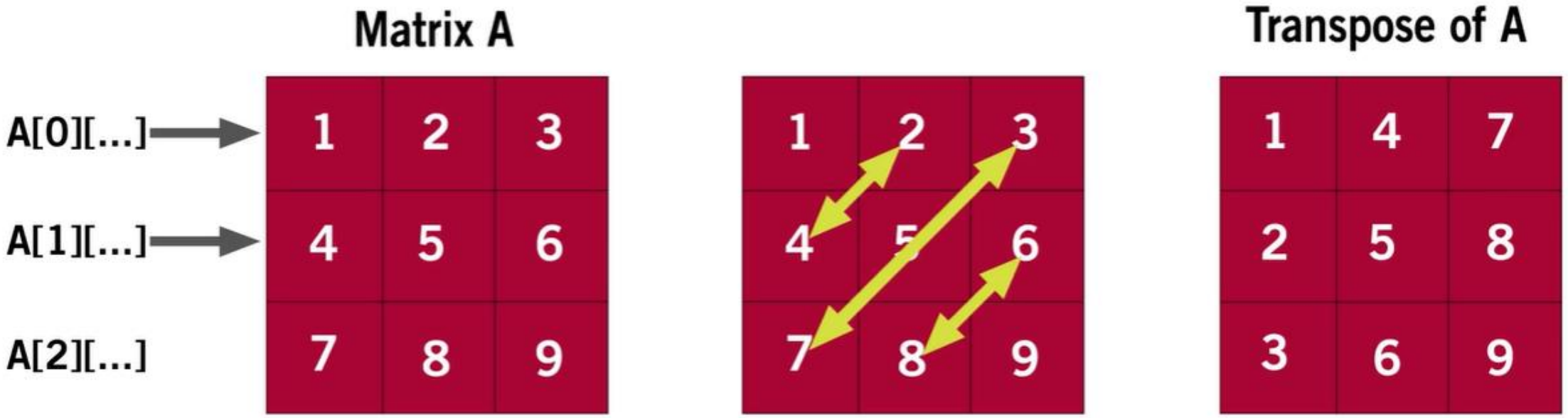
Rotate An Image 90 Degrees Clockwise (2/2)



Transpose A Matrix – Make The Rows Into Columns (1/8)



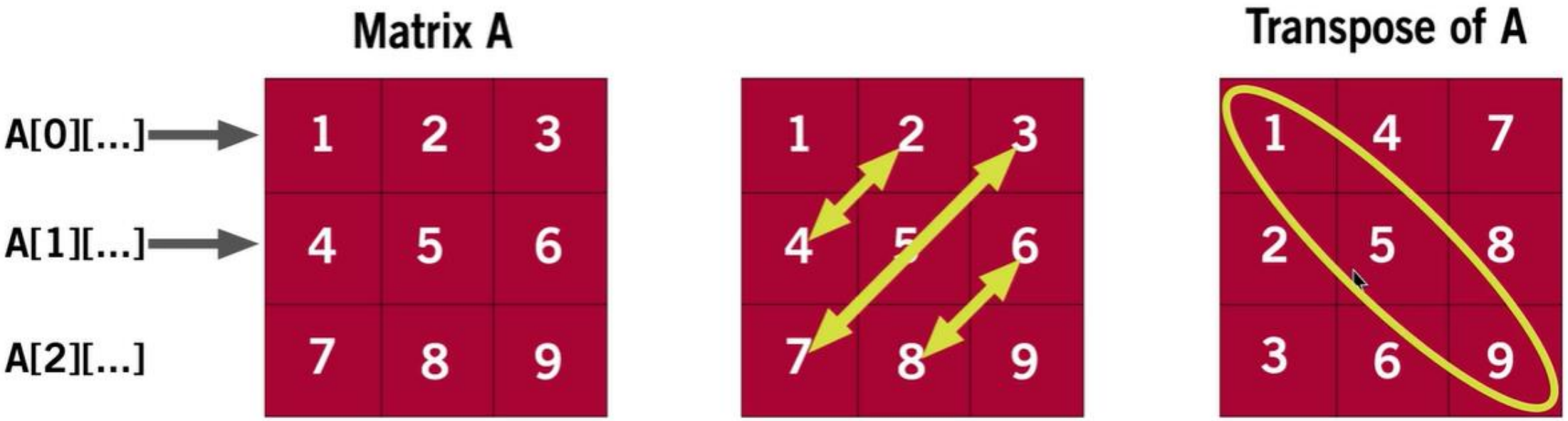
Transpose A Matrix – Make The Rows Into Columns (2/8)



A=[[1,2,3],
[4,5,6],
[7,8,9]]

SWAP A[i][j] with A[j][i]
SWAP A[0][1] – with A[1][0]
SWAP A[0][2] – with A[2][0]
SWAP A[1][2] – with A[2][1]

Transpose A Matrix – Make The Rows Into Columns (3/8)



A=[[1,2,3],
[4,5,6],
[7,8,9]]

SWAP A[i][j] with A[j][i]
SWAP A[0][1] – with A[1][0]
SWAP A[0][2] – with A[2][0]
SWAP A[1][2] – with A[2][1]

Transpose A Matrix – Make The Rows Into Columns (4/8)

```
17
18
19
20
21
22
23
24
25
26  function swap(A, i, j, k, l) {
27    let temp = A[j][i];
28    A[j][i] = A[l][k];
29    A[l][k] = temp;
30  }
31
32
33
34
35
36
```

Transpose A Matrix – Make The Rows Into Columns (5/8)

```
6
7   function transpose(A) {
8     // we need only swap upper triangle with lower
9     let N = A.length;
10    for (let i = 0; i < N - 1; i++) {
11      for (let j = i + 1; j < N; j++) {
12        swap(A, i, j, j, i);
13      }
14    }
15  }
16
17
18  function swap(A, i, j, k, l) {
19    let temp = A[j][i];
20    A[j][i] = A[l][k];
21    A[l][k] = temp;
22  }
23
24
25
```

Transpose A Matrix – Make The Rows Into Columns (6/8)

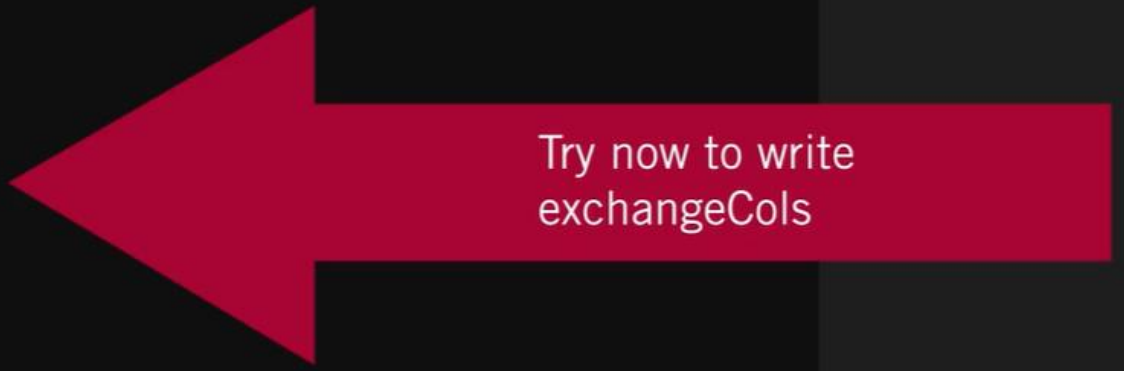
```
25
26
27
28
29
30  function exchangeCols(A) {
31    // work in from outside
32    let N = A.length; // we need only swap N/2 (rounded down)
33    let n = Math.floor(N / 2); // round down to integer
34    for (let col = 0; col < n; col++) {
35      for (let row = 0; row < N; row++) {
36        swap(A, col, row, N - 1 - col, row);
37      }
38    }
39  }
40
41  function rotateMatrix(A) {
42    transpose(A);
43    exchangeCols(A);
44  }
45  console.log('Original A = ' + JSON.stringify(A));
```

Transpose A Matrix – Make The Rows Into Columns (7/8)

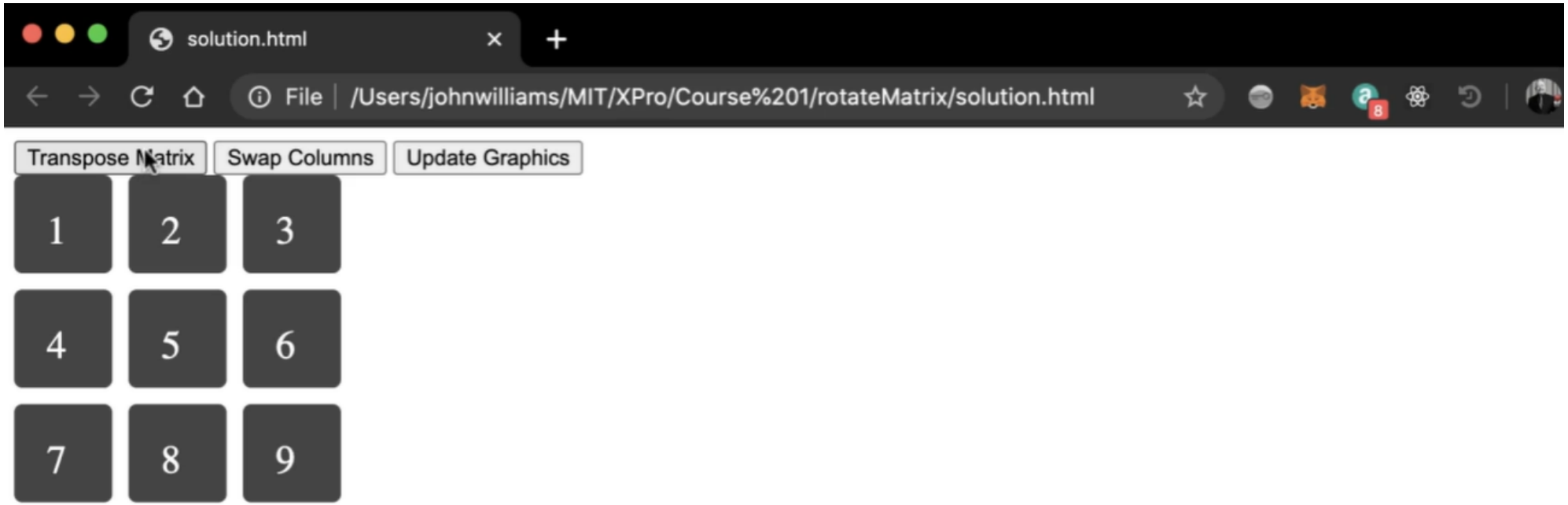
```
33      let n = Math.floor(N / 2); // round down to integer
34      for (let col = 0; col < n; col++) {
35        for (let row = 0; row < N; row++) {
36          swap(A, col, row, N - 1 - col, row);
37        }
38      }
39    }
40
41    function rotateMatrix(A) {
42      transpose(A);
43      exchangeCols(A);
44    }
45    console.log('Original A = ' + JSON.stringify(A));
46    transpose(A);
47    console.log('Transpose A = ' + JSON.stringify(A));
48    exchangeCols(A);
49    console.log('A Rotated = ' + JSON.stringify(A));
```

Transpose A Matrix – Make The Rows Into Columns (8/8)

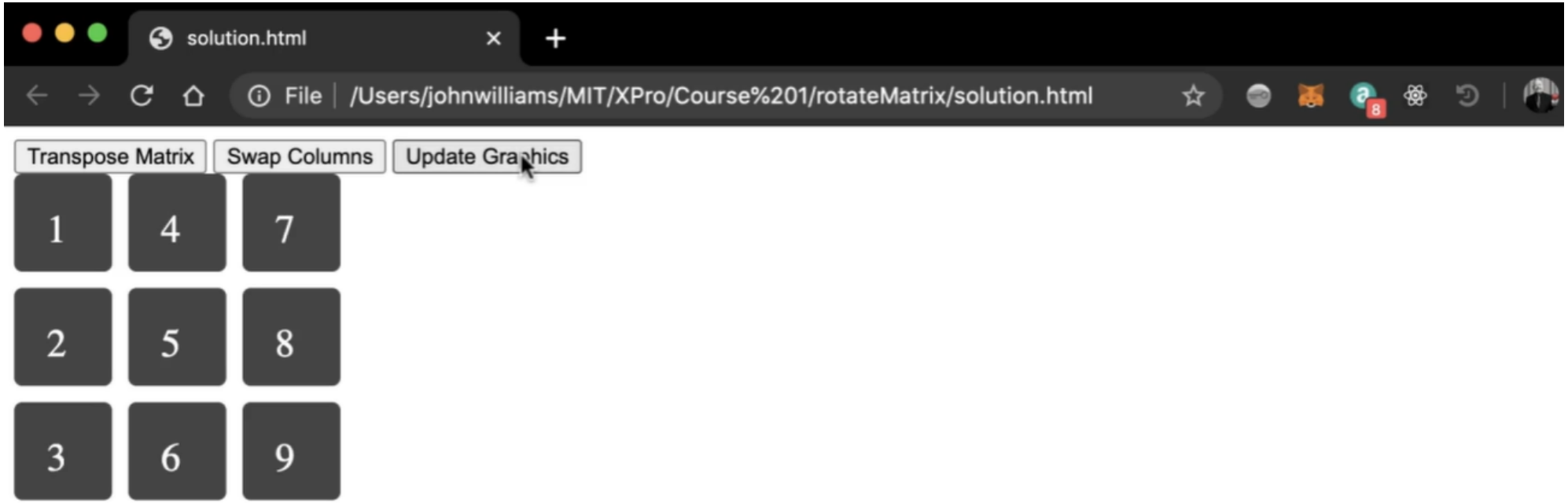
```
1  const A = [[1, 2, 3],[4, 5, 6],[7, 8, 9]];
2  function transpose(A) {
3    // we need only swap upper triangle with lower
4    let N = A.length;
5    for (let i = 0; i < N - 1; i++) {
6      for (let j = i + 1; j < N; j++) {
7        swap(A, i, j, j, i);
8      }
9    }
10 }
11 function swap(A, i, j, k, l) {
12   let temp = A[j][i];
13   A[j][i] = A[l][k];
14   A[l][k] = temp;
15 }
16 function exchangeCols(A) {…
25 }
26 function rotateMatrix(A) {
27   transpose(A);
28   exchangeCols(A);
29 }
30 console.log('Original A = ' + JSON.stringify(A));
31 transpose(A);
32 console.log('Transpose A = ' + JSON.stringify(A));
33 exchangeCols(A);
34 console.log('A Rotated = ' + JSON.stringify(A));
```



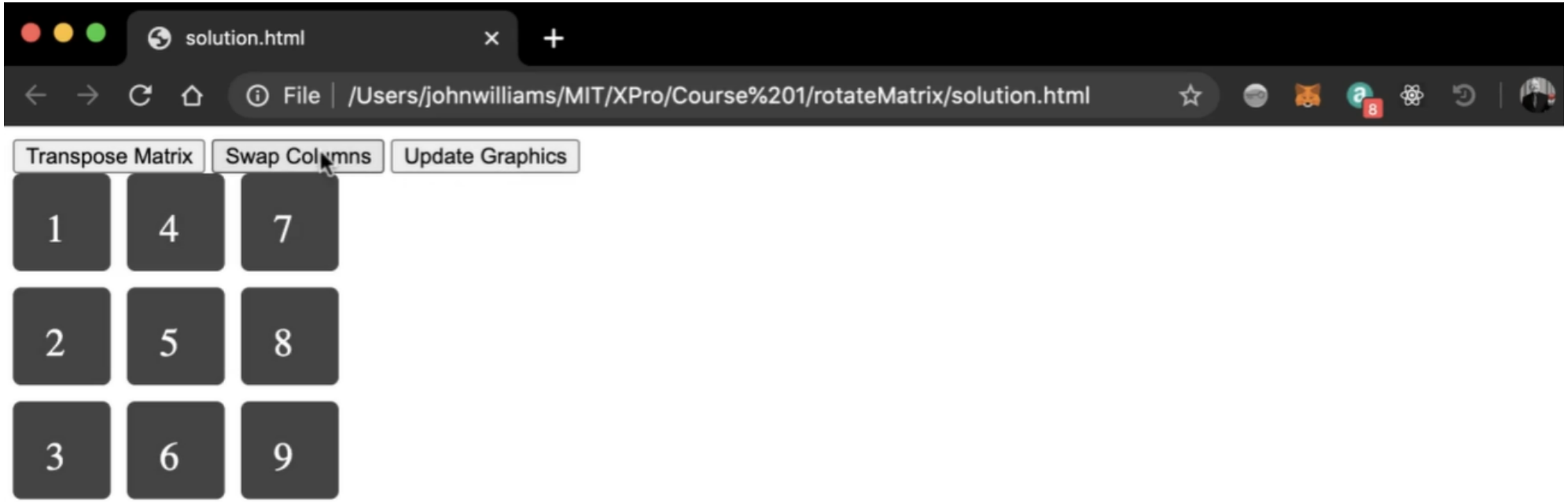
Transpose Matrix Exercise (1/4)



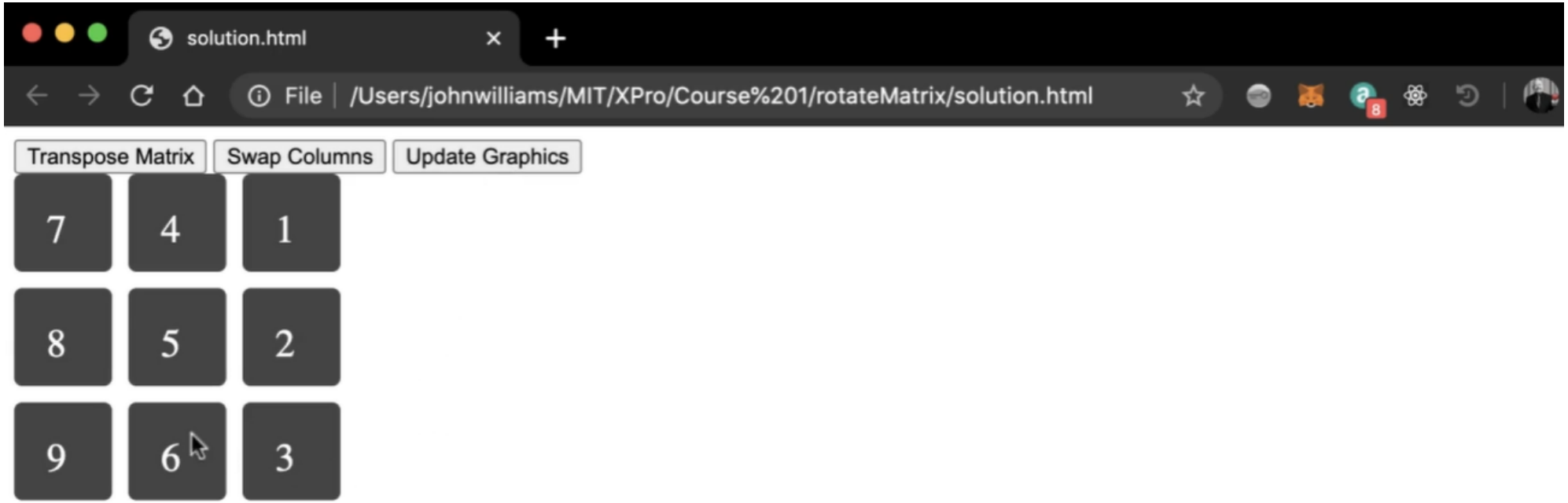
Transpose Matrix Exercise (2/4)



Transpose Matrix Exercise (3/4)

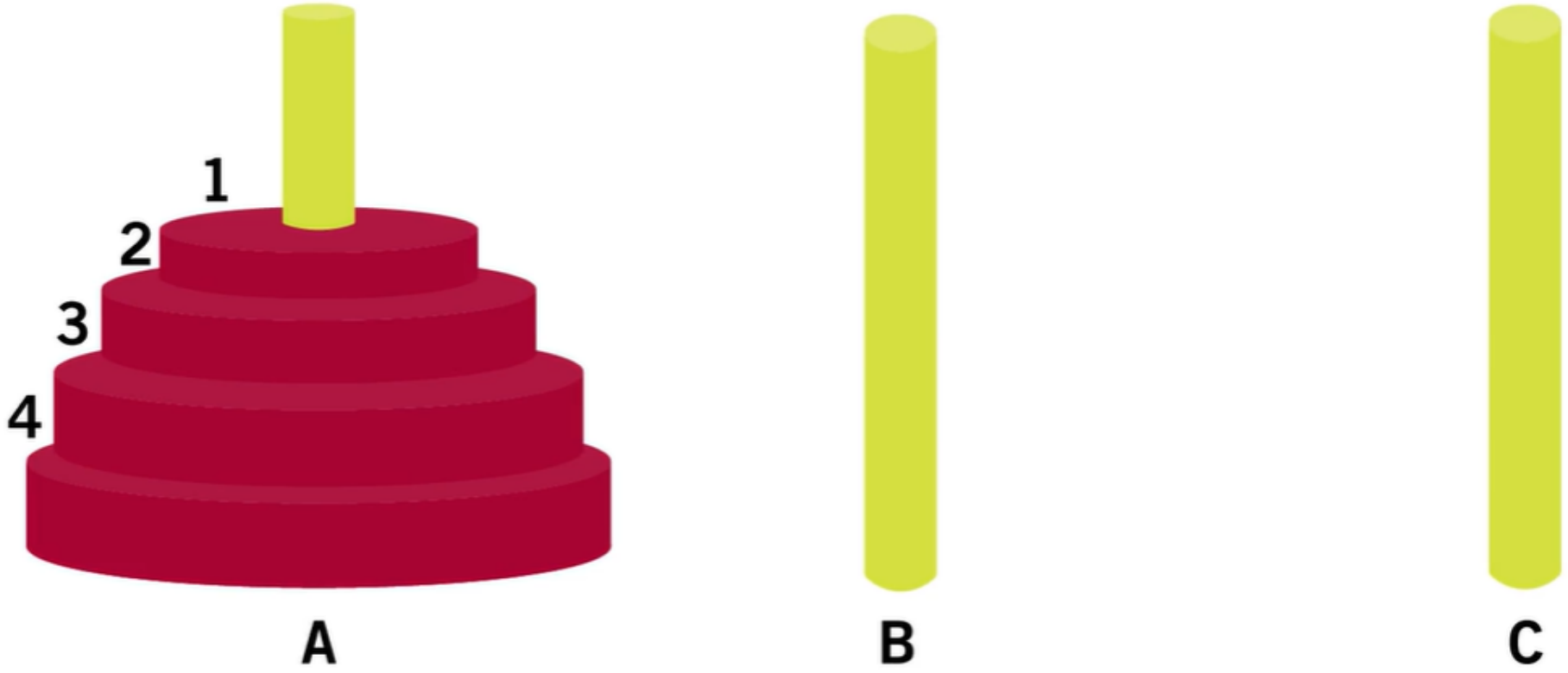


Transpose Matrix Exercise (4/4)



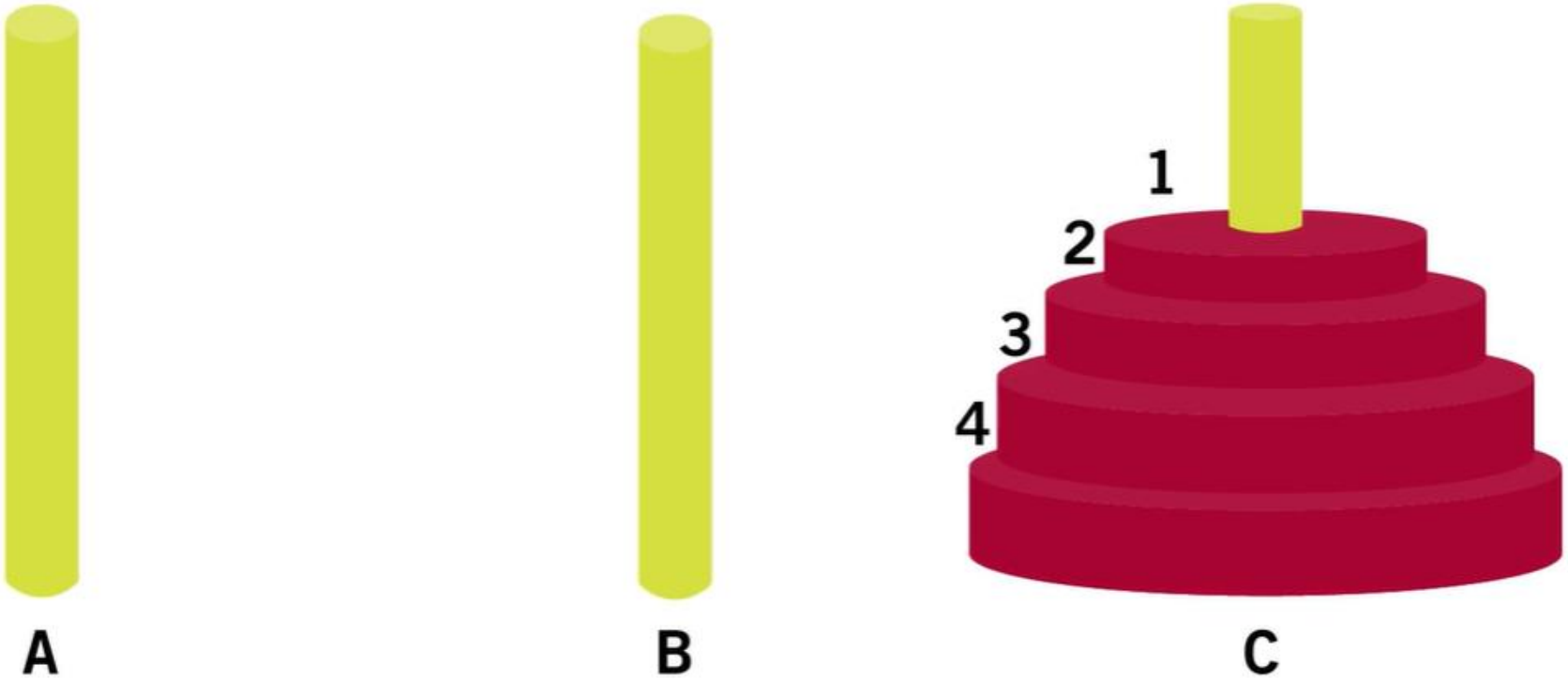
Tower Of Hanoi (1/19)

- Goal Move all disks from A to C
- Only top disc of any stack can be moved
- It must be moved to sit on a larger disk



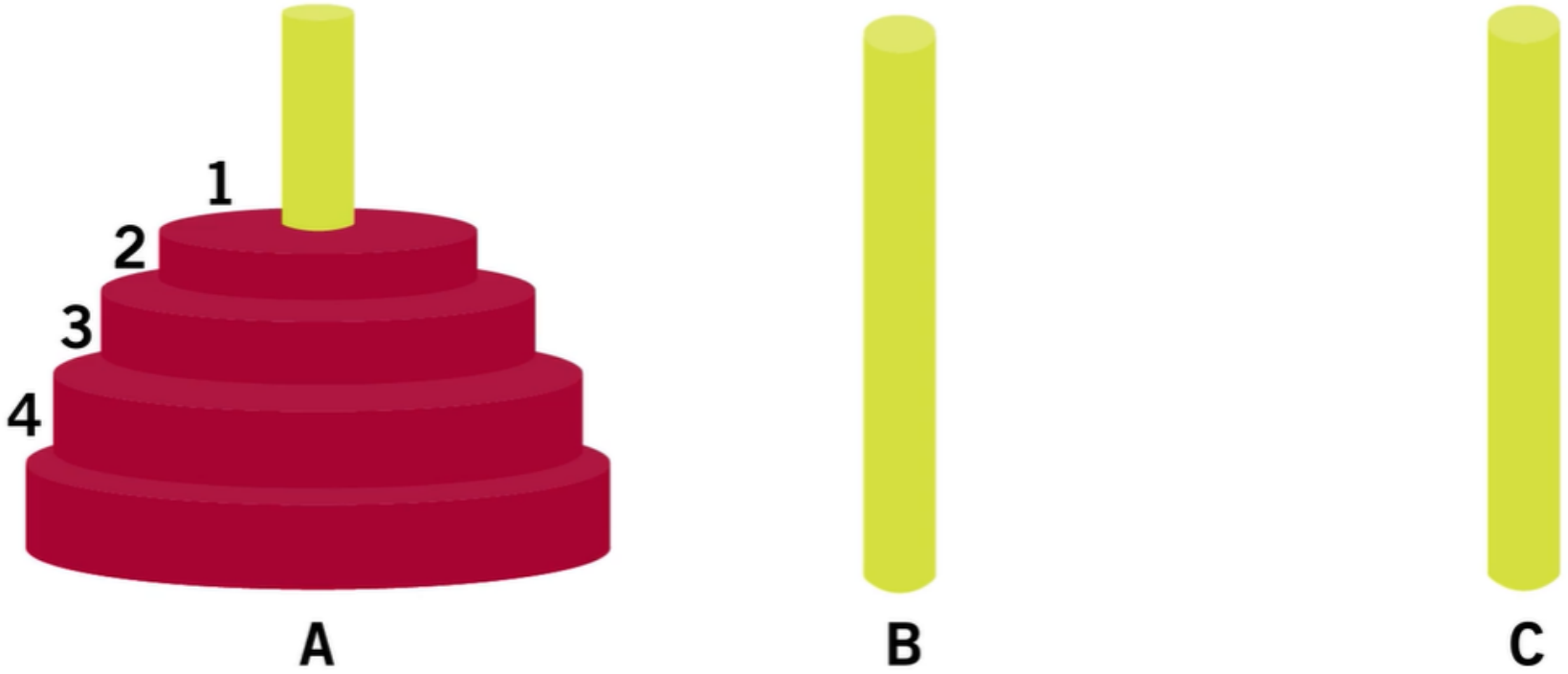
Tower Of Hanoi (2/19)

Move Disks From A To C



Tower Of Hanoi (3/19)

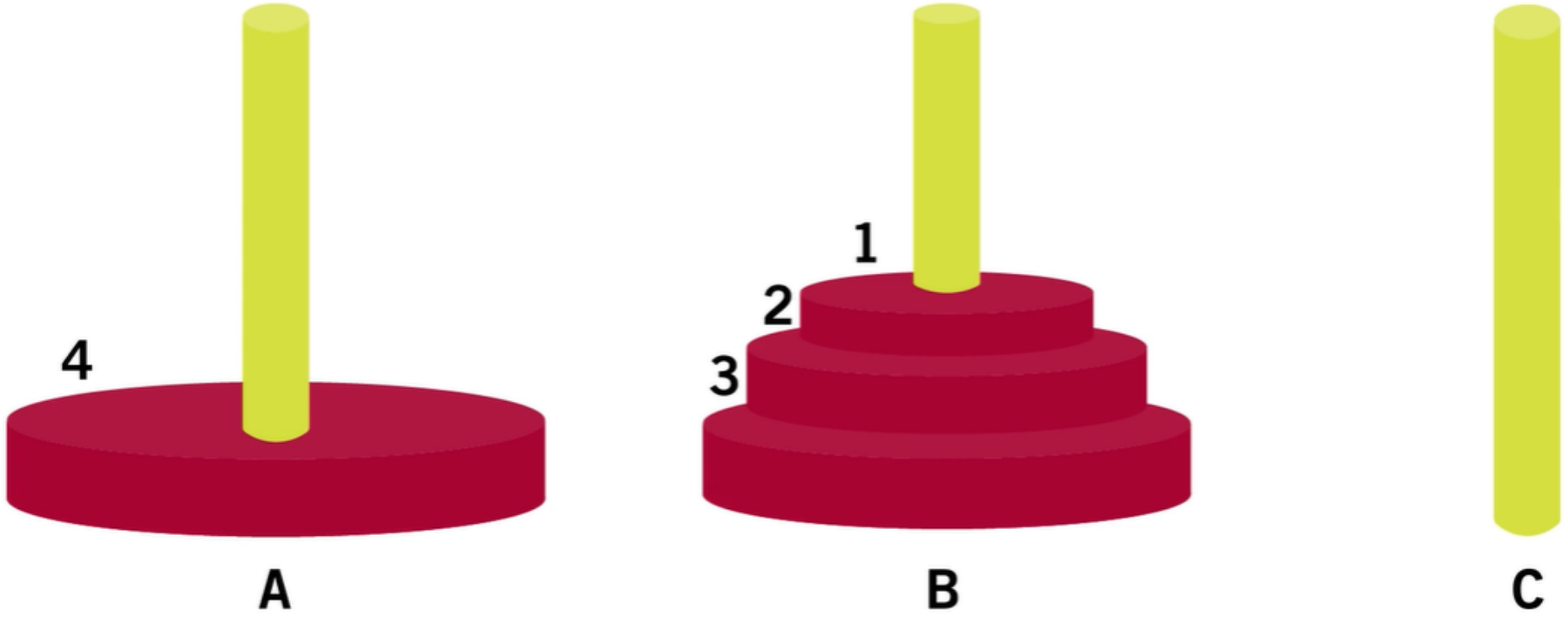
Move n disks from A to C, using B
(n, A, C, B)



Move n-1 (n-1, A, B, C)

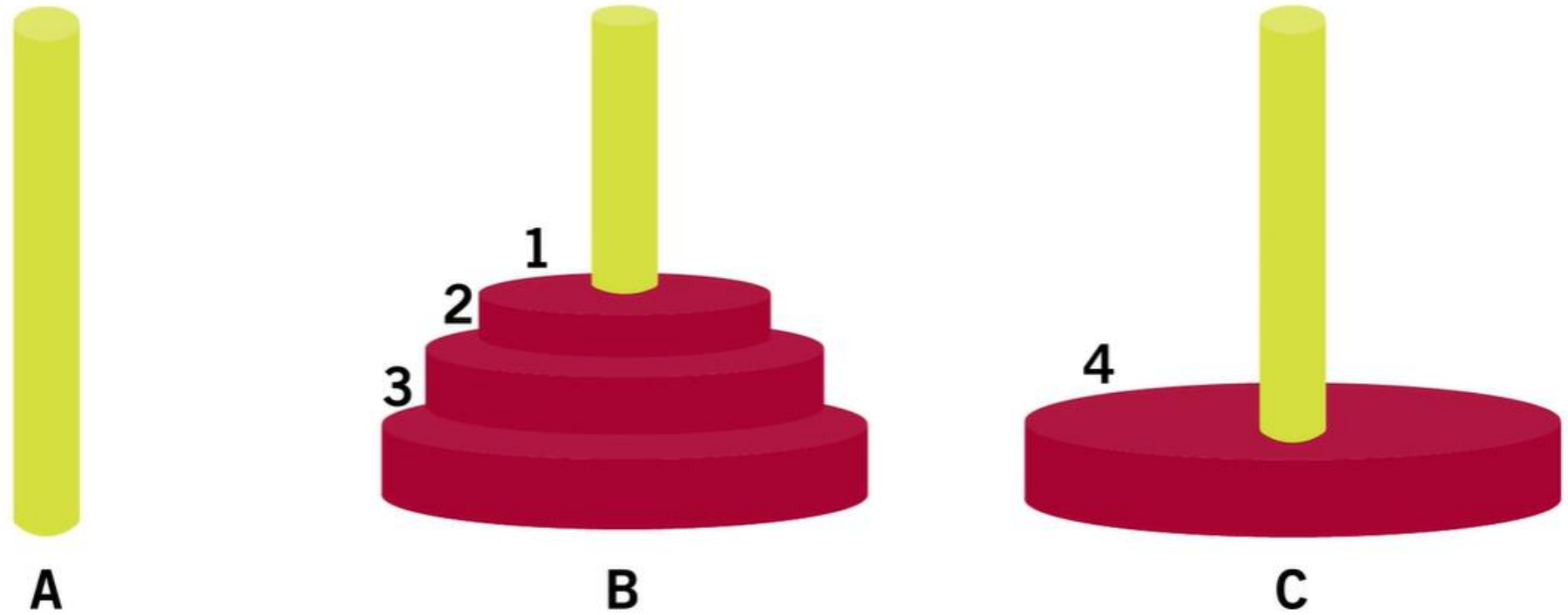
Tower Of Hanoi (4/19)

Move (n-1, A, B, C)



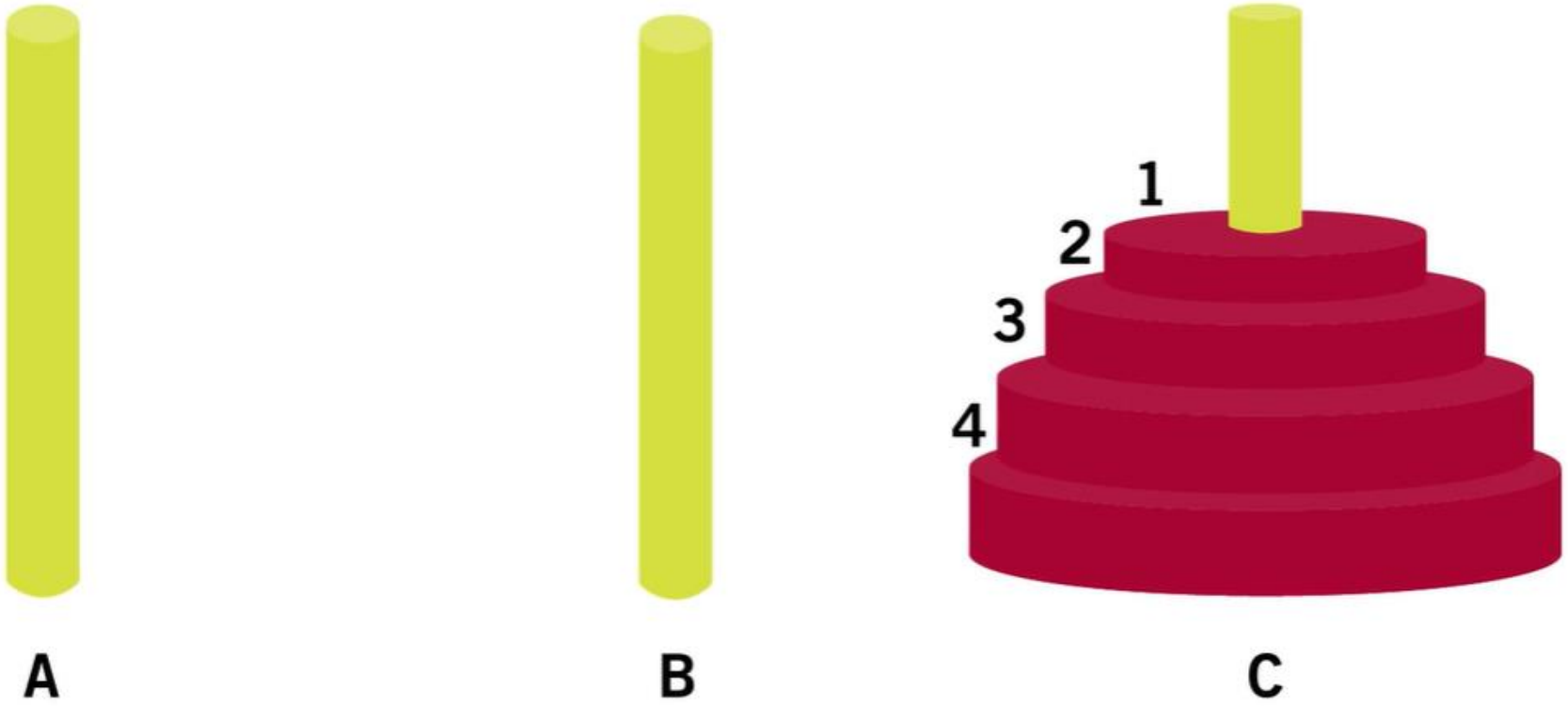
Tower Of Hanoi (5/19)

Move (n-1, A, B, C)
Move disk n from A to C



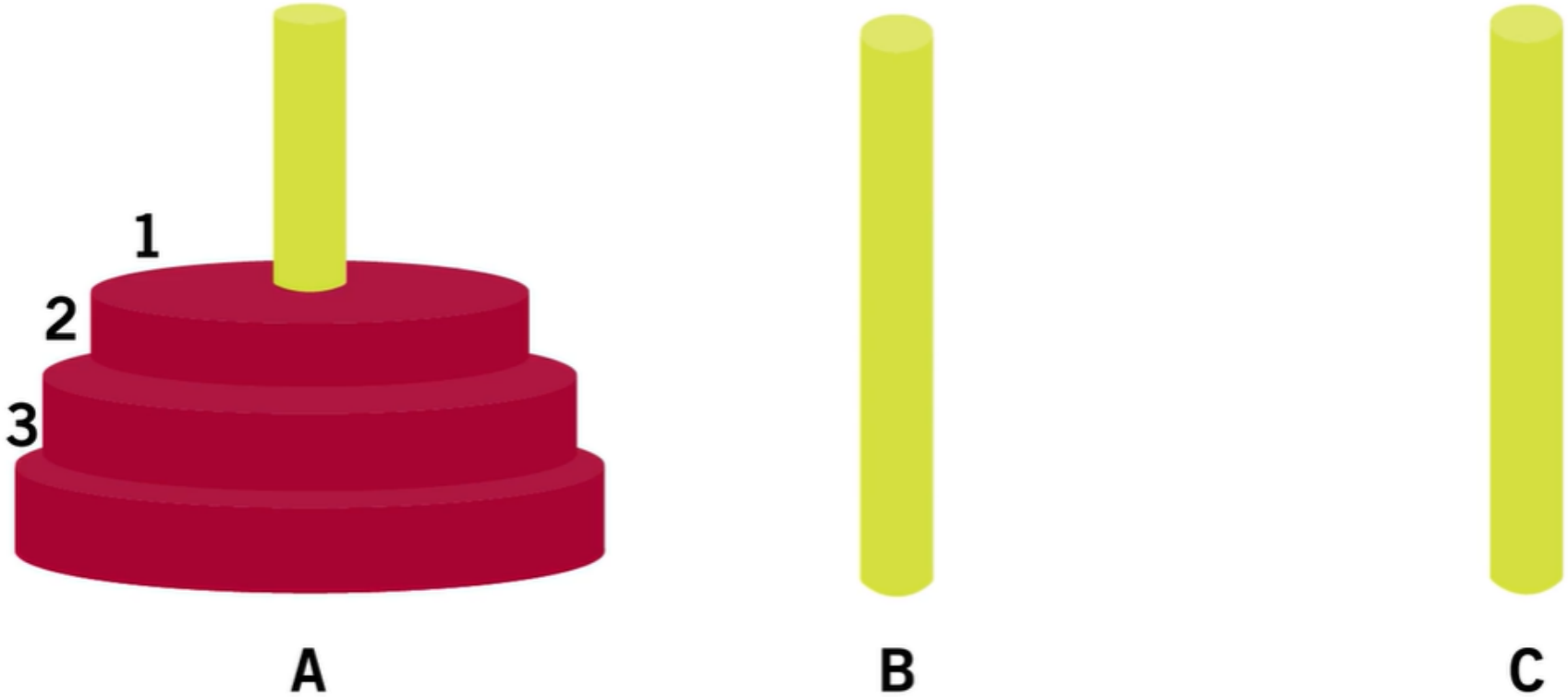
Tower Of Hanoi (6/19)

Move (n - 1, A, B, C)
Move disk n from A to C
Move (n - 1, B, C, A)



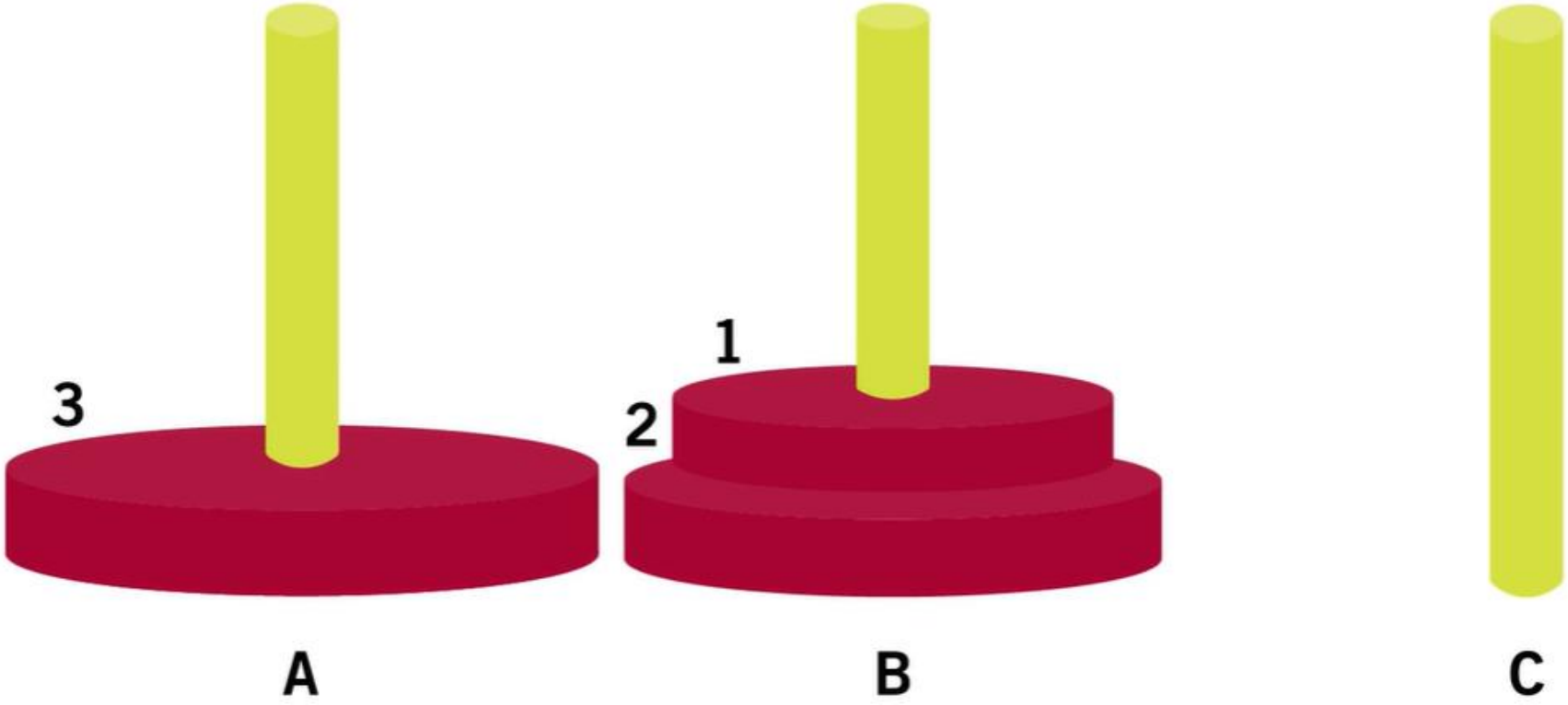
Tower Of Hanoi (7/19)

Move (3, A, C, B) = Move (2, A, B, C)
Move disk 3 A to C
Move (2, B, C, A)



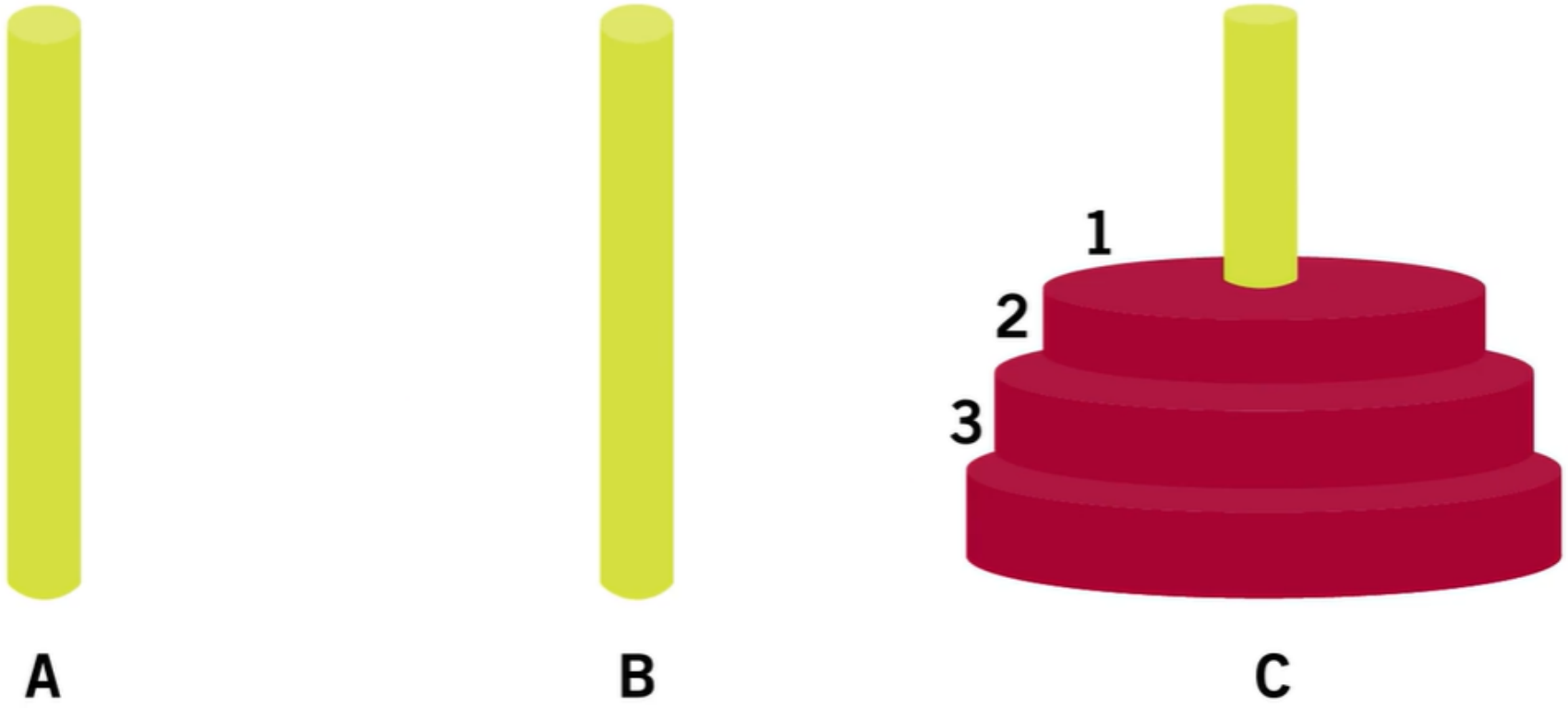
Tower Of Hanoi (8/19)

Move (3, A, C, B) = Move (2, A, B, C)
Move disk 3 A to C
Move (2, B, C, A)



Tower Of Hanoi (9/19)

Move (3, A, C, B) = Move (2, A, B, C)
Move disk 3 A to C
Move (2, B, C, A)

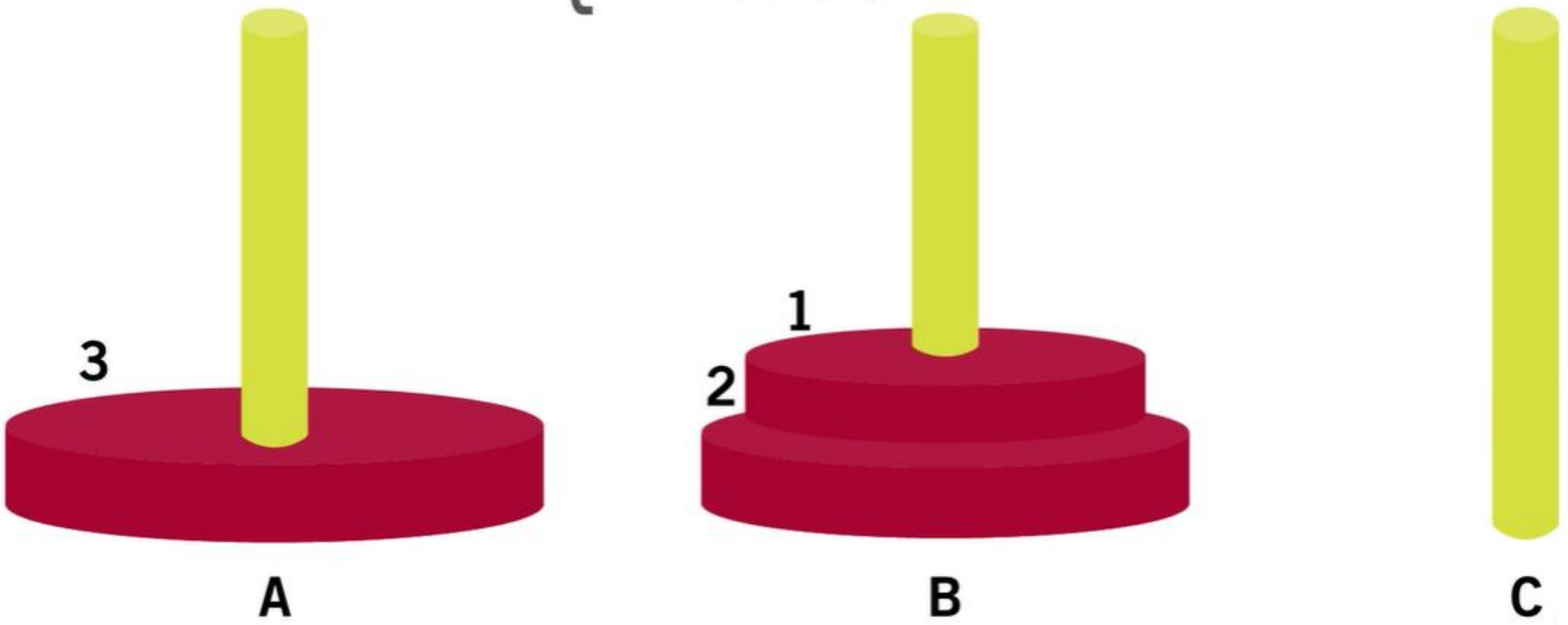


Tower Of Hanoi (10/19)

Move (2, A, B, C)
Move disk 3 A to C
Move (2, B, C, A)

Move (1, A, C, B)
Move disk 2 A to B
Move (1, C, B, A)

Move (1, B, A, C)
Move disk 2 B to C
Move (1, A, C, B)

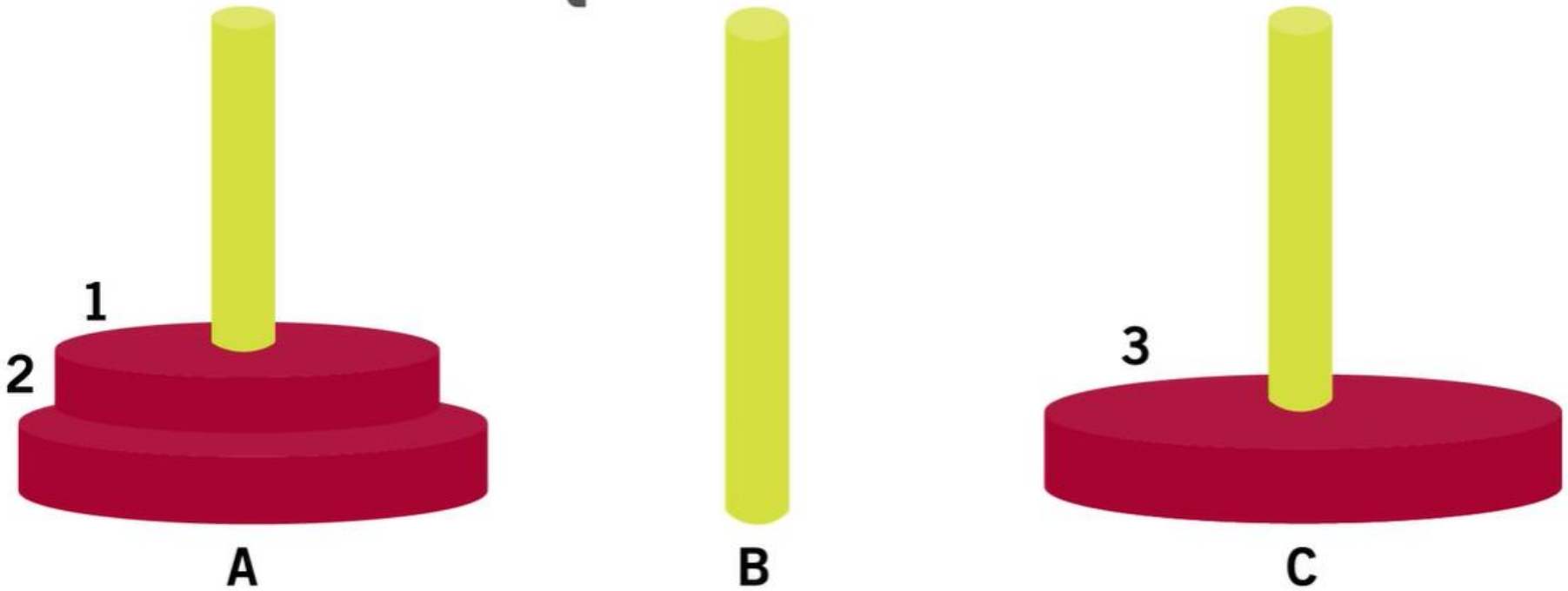


Tower Of Hanoi (11/19)

Move (2, A, B, C)
Move disk 3 A to C
Move (2, B, C, A)

Move (1, A, C, B)
Move disk 2 A to B
Move (1, C, B, A)

Move (1, B, A, C)
Move disk 2 B to C
Move (1, A, C, B)

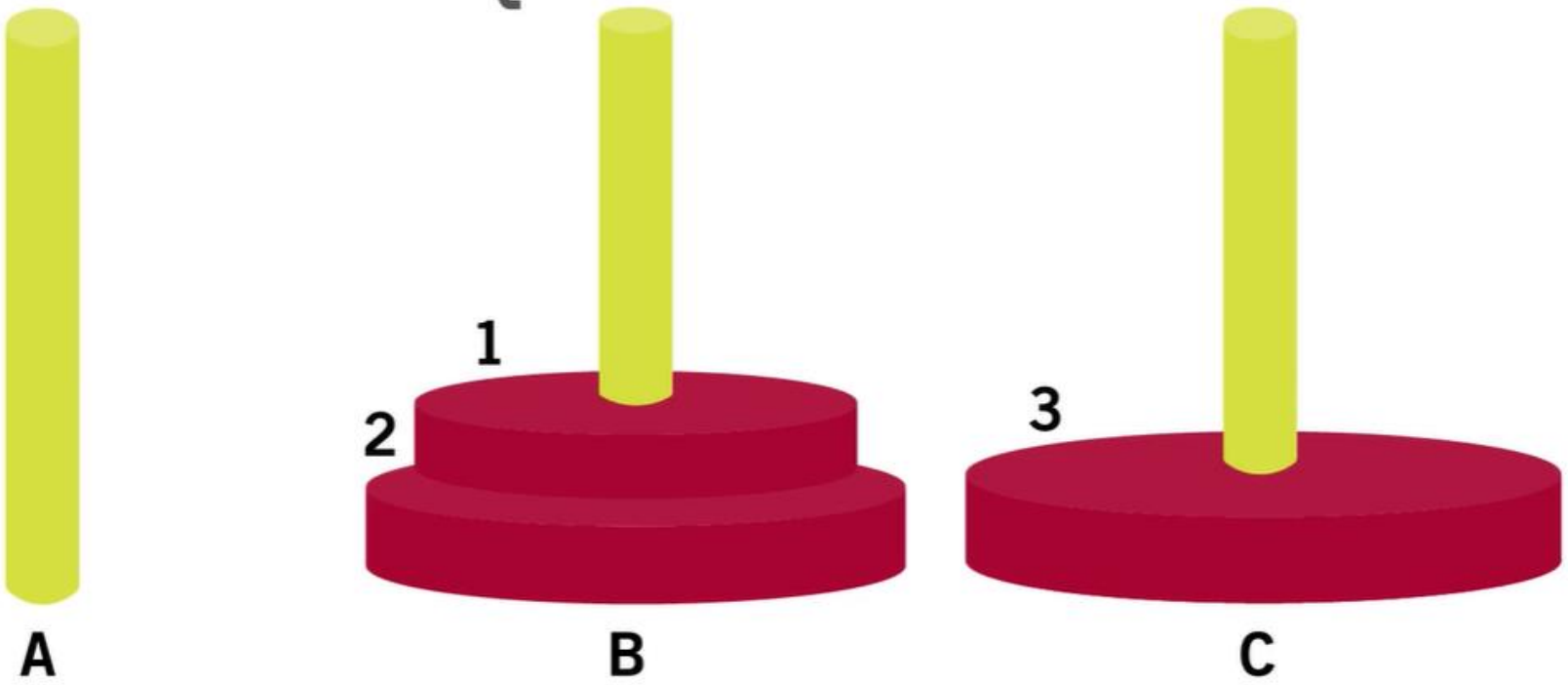


Tower Of Hanoi (12/19)

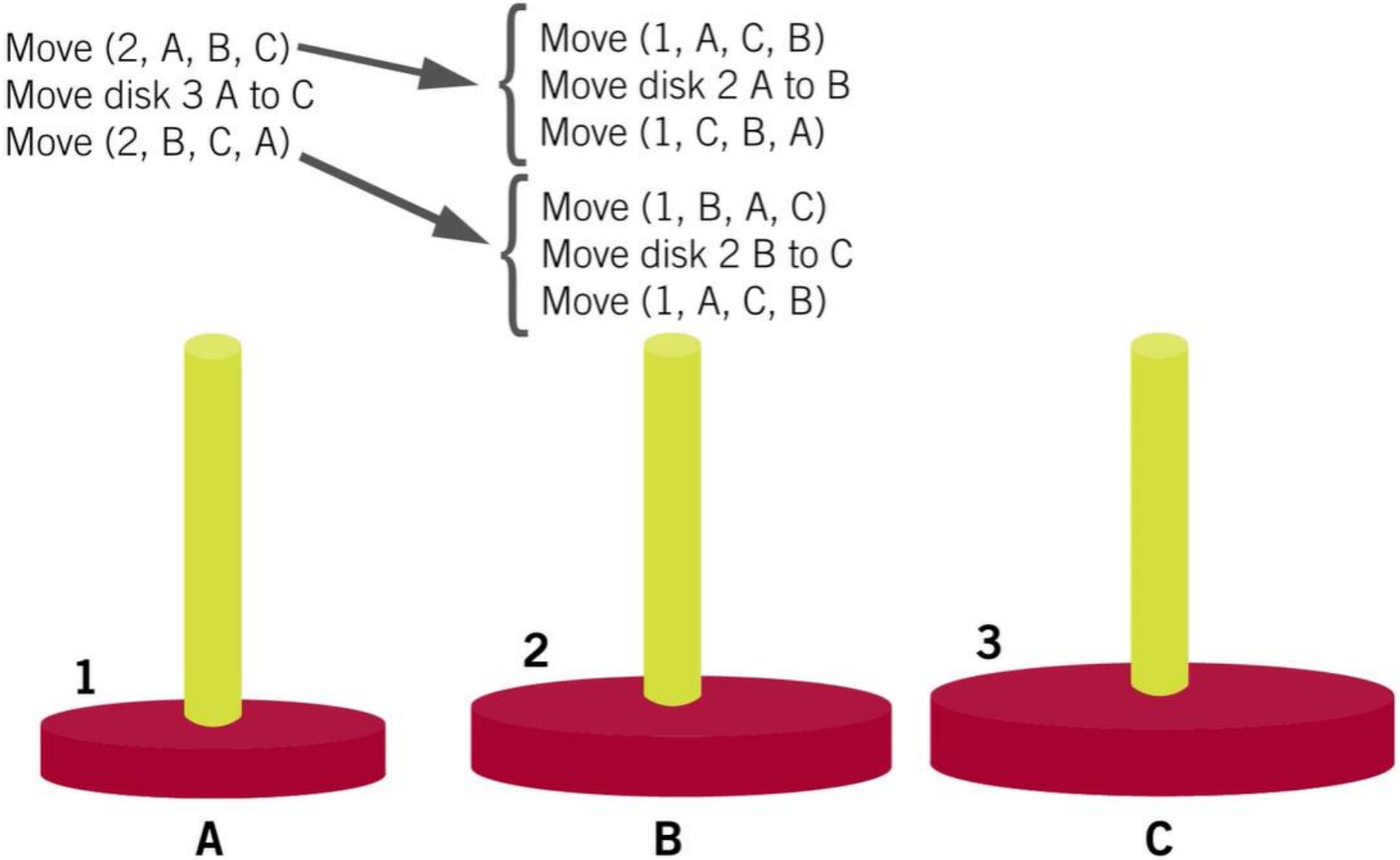
Move (2, A, B, C)
Move disk 3 A to C
Move (2, B, C, A)

Move (1, A, C, B)
Move disk 2 A to B
Move (1, C, B, A)

Move (1, B, A, C)
Move disk 2 B to C
Move (1, A, C, B)



Tower Of Hanoi (13/19)

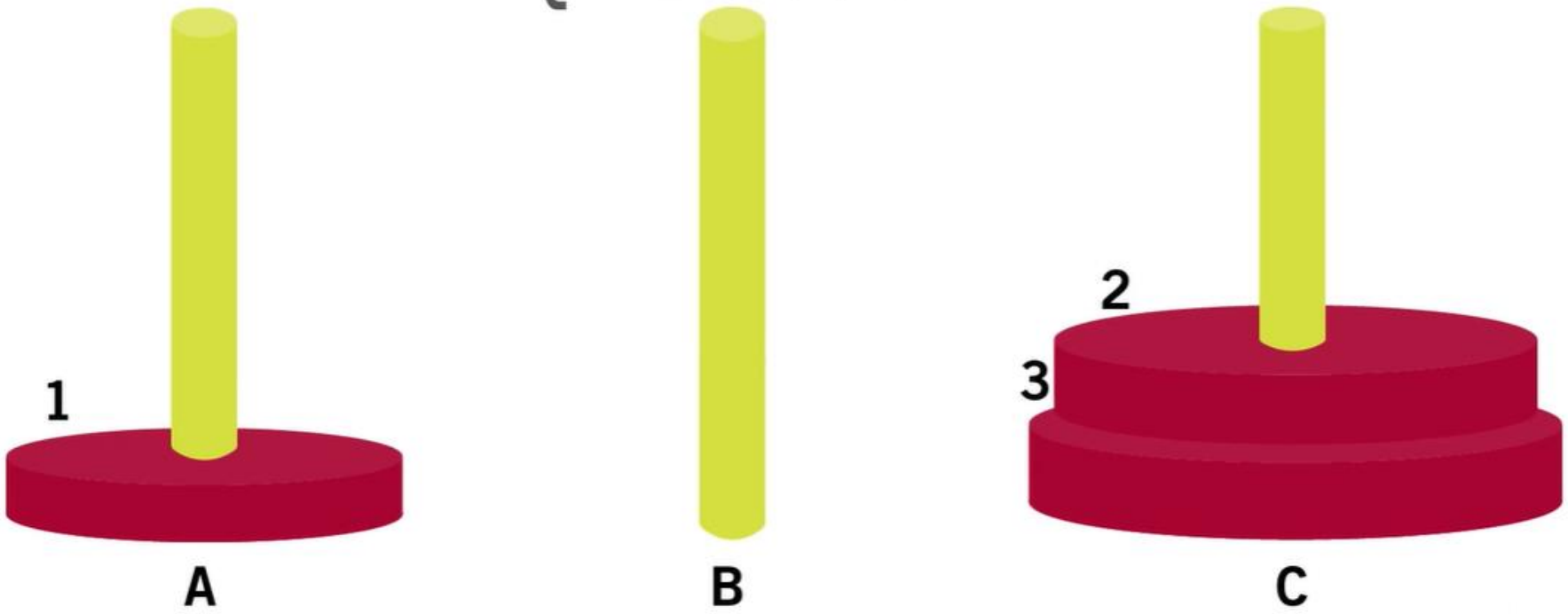


Tower Of Hanoi (14/19)

Move (2, A, B, C)
Move disk 3 A to C
Move (2, B, C, A)

Move (1, A, C, B)
Move disk 2 A to B
Move (1, C, B, A)

Move (1, B, A, C)
Move disk 2 B to C
Move (1, A, C, B)

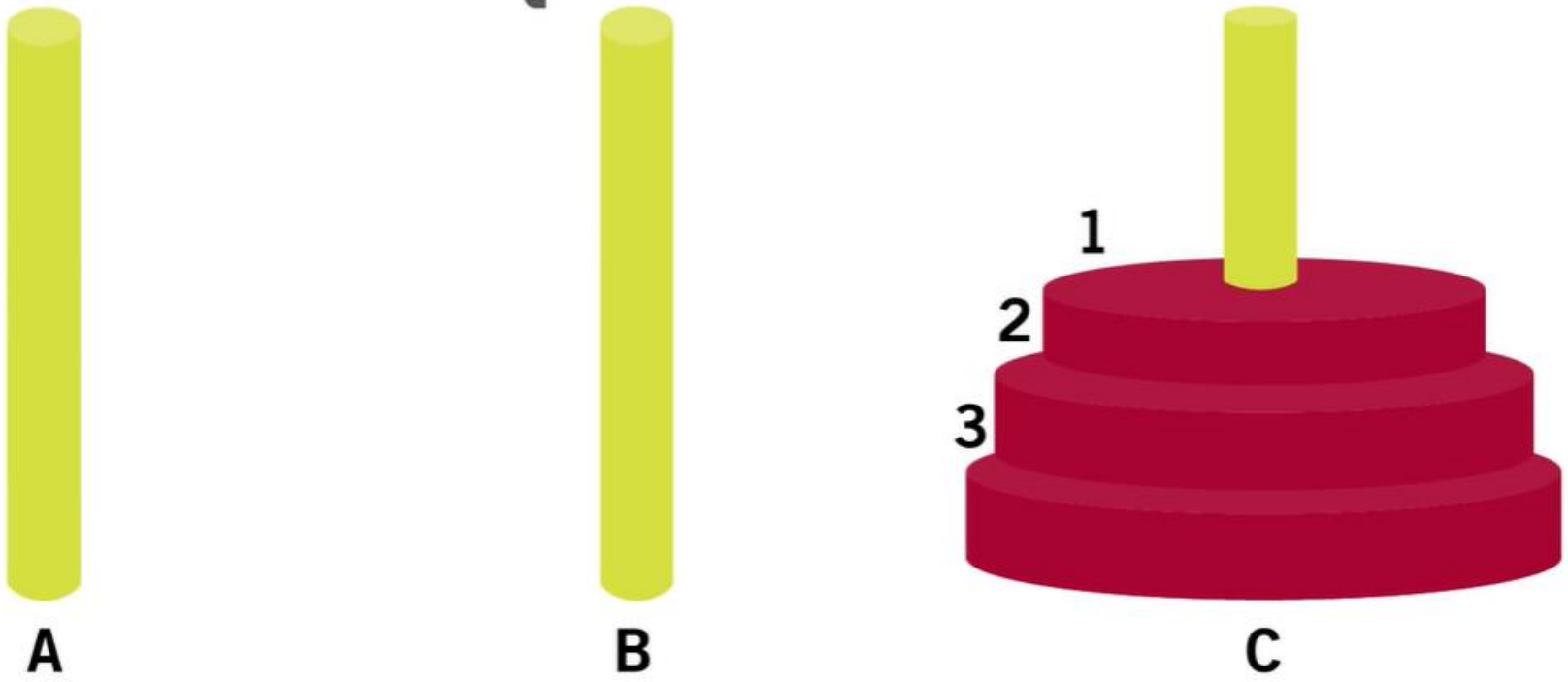


Tower Of Hanoi (15/19)

Move (2, A, B, C)
Move disk 3 A to C
Move (2, B, C, A)

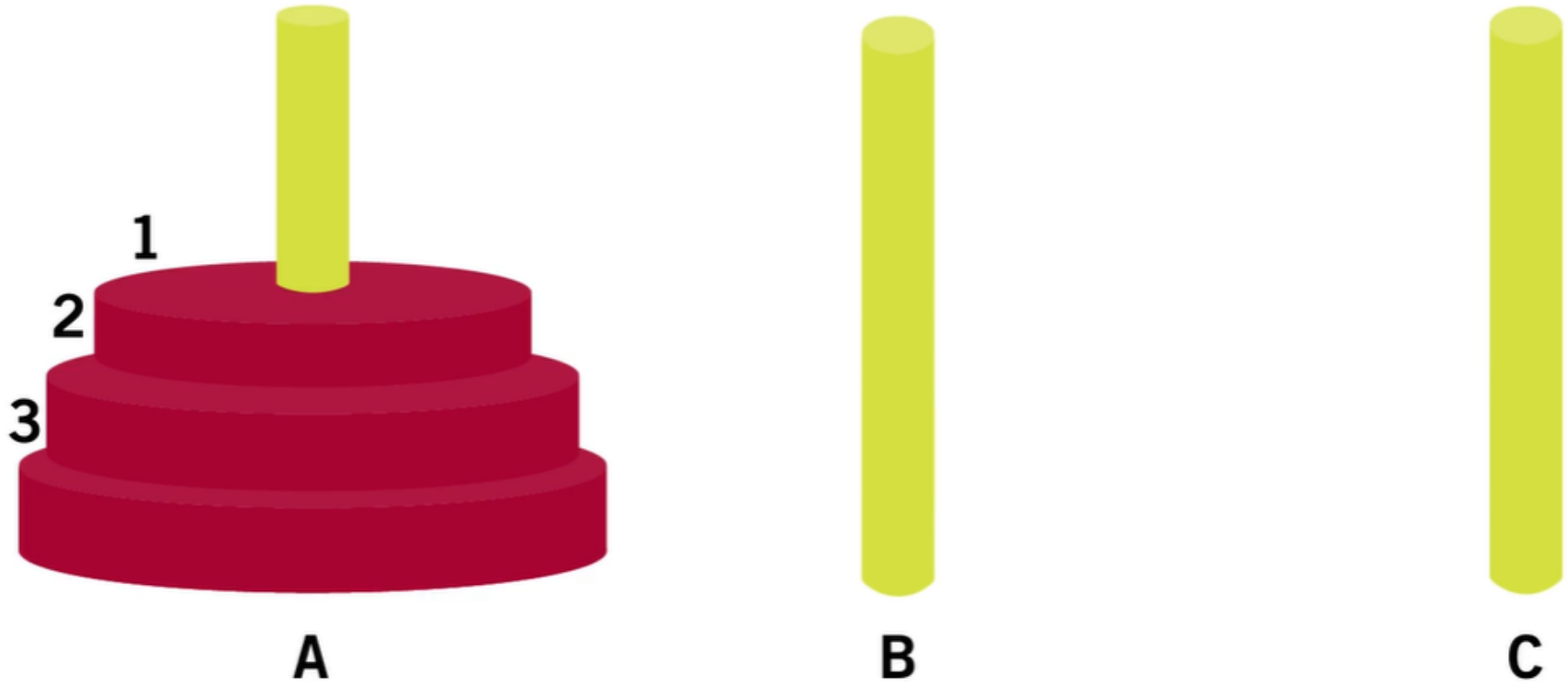
Move (1, A, C, B)
Move disk 2 A to B
Move (1, C, B, A)

Move (1, B, A, C)
Move disk 2 B to C
Move (1, A, C, B)



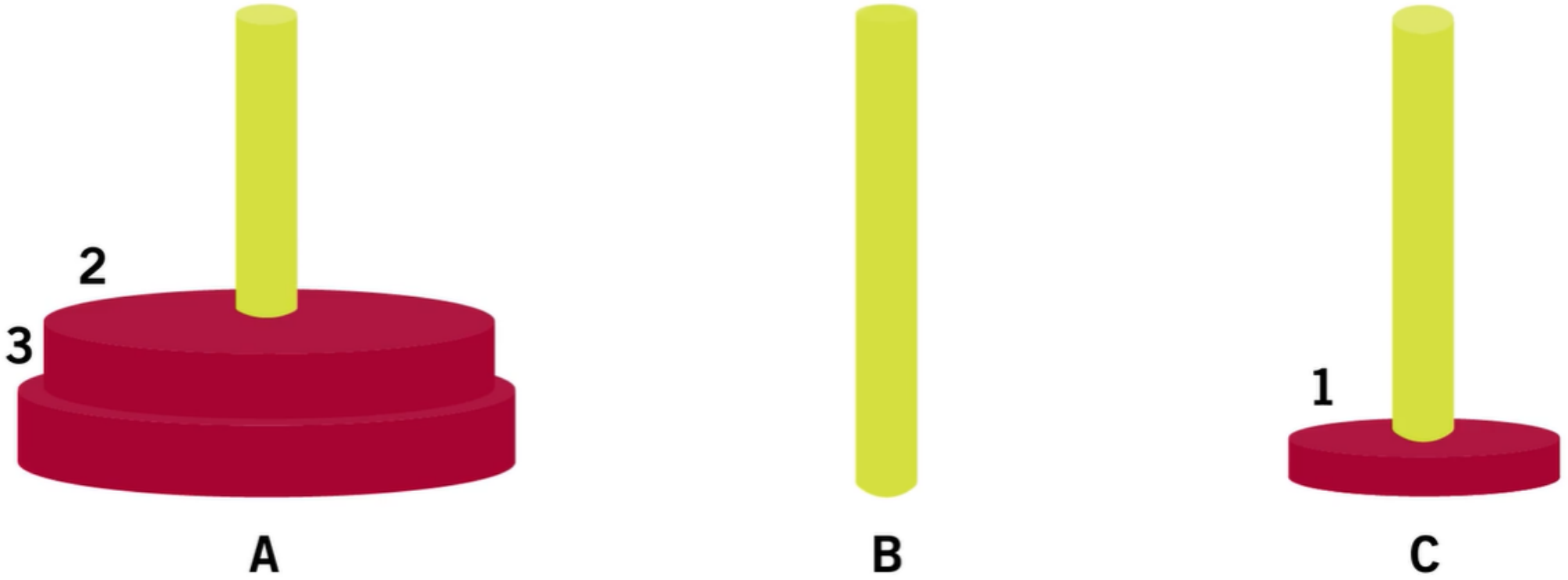
Tower Of Hanoi (16/19)

Move (2, A, B, C) = Move (1, A, C, B){if (n = 1): Move disk 1 A to C}
Move disk 2 A to B
Move (1, C, B, A){if (n = 1): Move disk C to B}



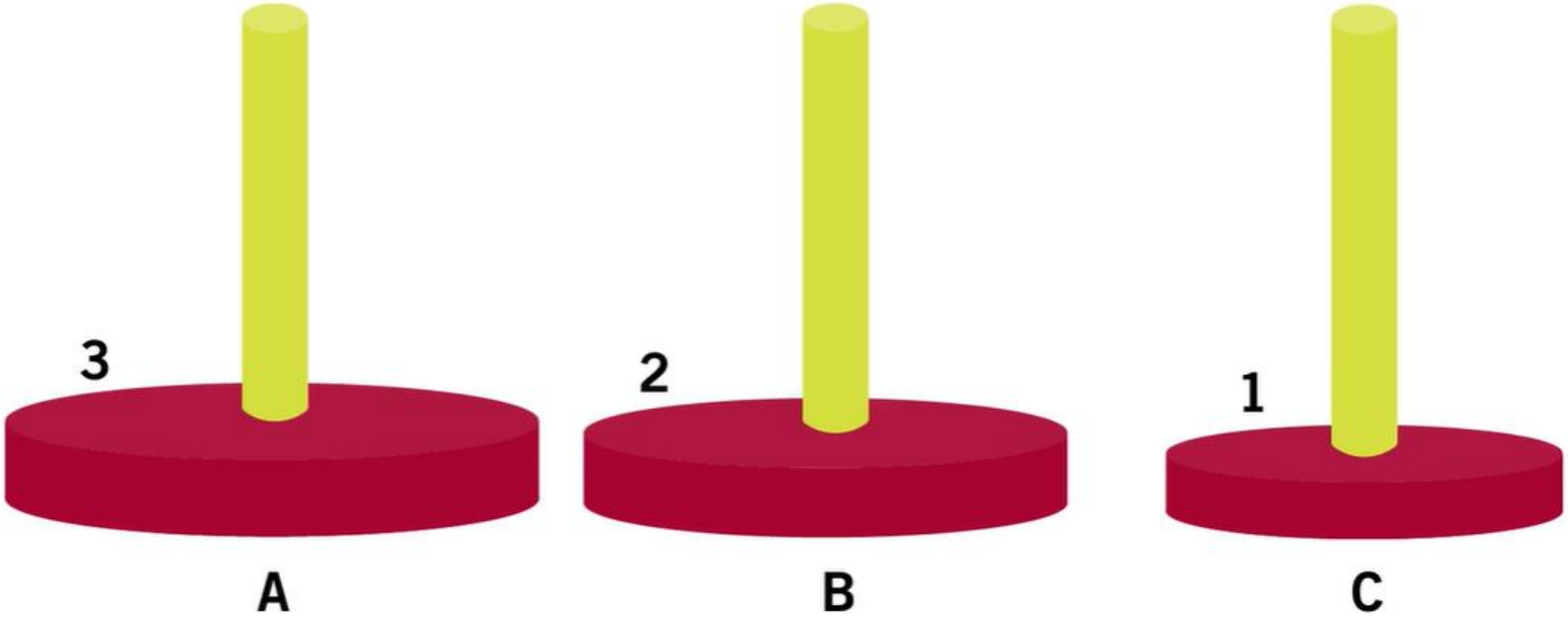
Tower Of Hanoi (17/19)

Move (2, A, B, C) = Move (1, A, C, B){if (n = 1): Move disk 1 A to C}
Move disk 2 A to B
Move (1, C, B, A){if (n = 1): Move disk C to B}



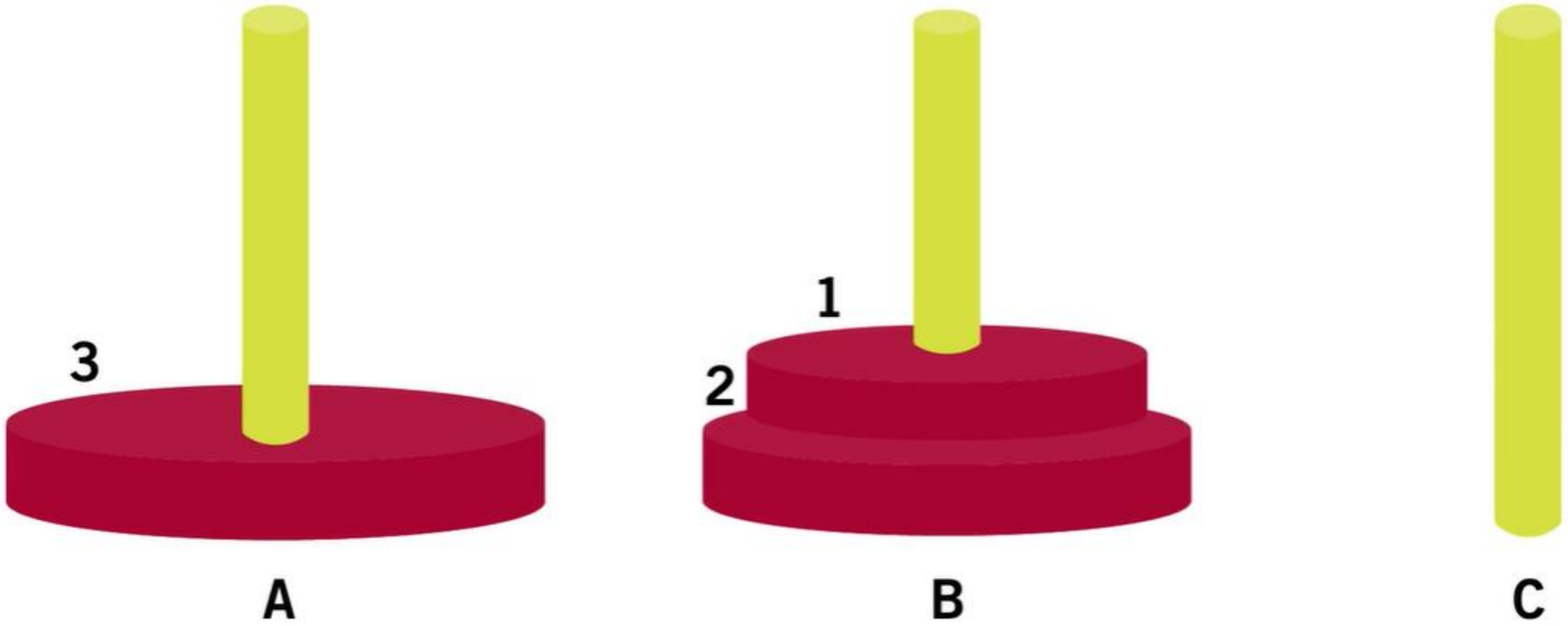
Tower Of Hanoi (18/19)

Move (2, A, B, C) = Move (1, A, C, B){if (n = 1): Move disk 1 A to C}
Move disk 2 A to B
Move (1, C, B, A){if (n = 1): Move disk C to B}



Tower Of Hanoi (19/19)

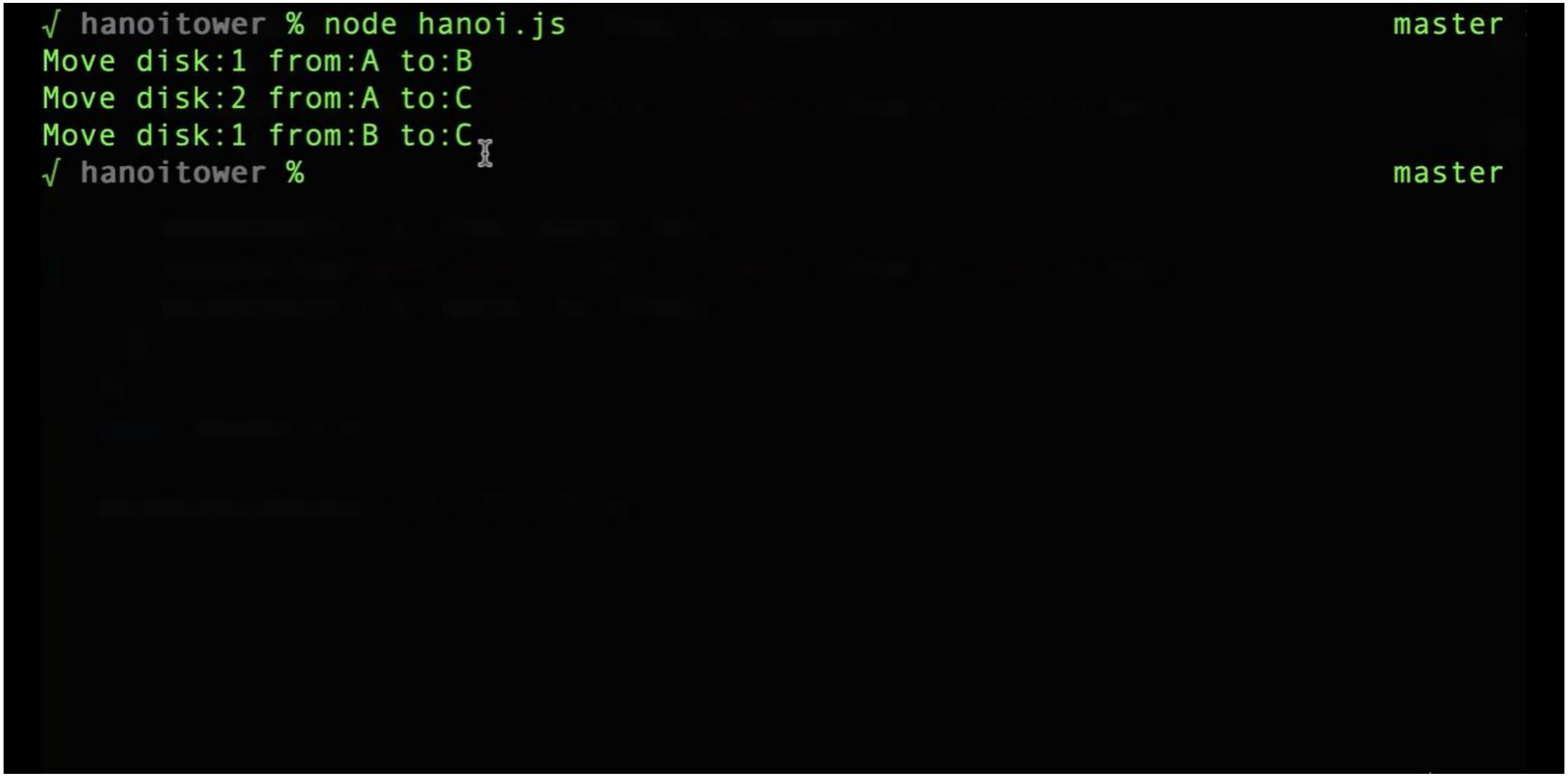
Move (2, A, B, C) = Move (1, A, C, B){if (n = 1): Move disk 1 A to C}
Move disk 2 A to B
Move (1, C, B, A){if (n = 1): Move disk C to B}



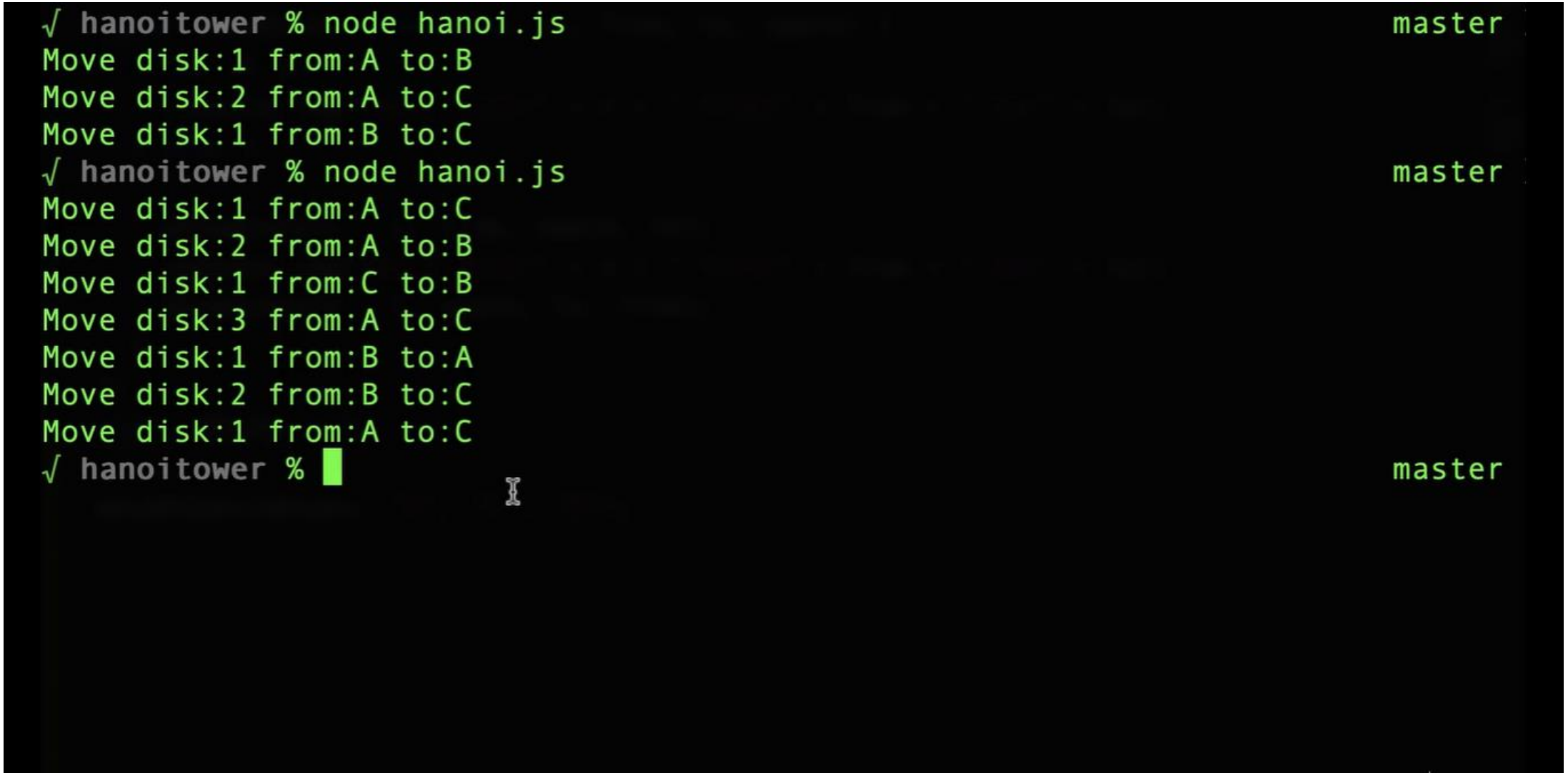
Tower Of Hanoi: Code

```
const moveDisks = function (n, from, to, spare) {
  if (n == 1) {
    console.log("Move disk:" + n + " from:" + from + " to:" + to);
  } else {

    moveDisks(n - 1, from, spare, to);
    console.log("Move disk:" + n + " from:" + from + " to:" + to);
    moveDisks(n - 1, spare, to, from);
  }
};
const nDisks = 2;

moveDisks(nDisks, "A", "C", "B");
```

Tower Of Hanoi: Output (1/4)



Tower Of Hanoi: Output (2/4)



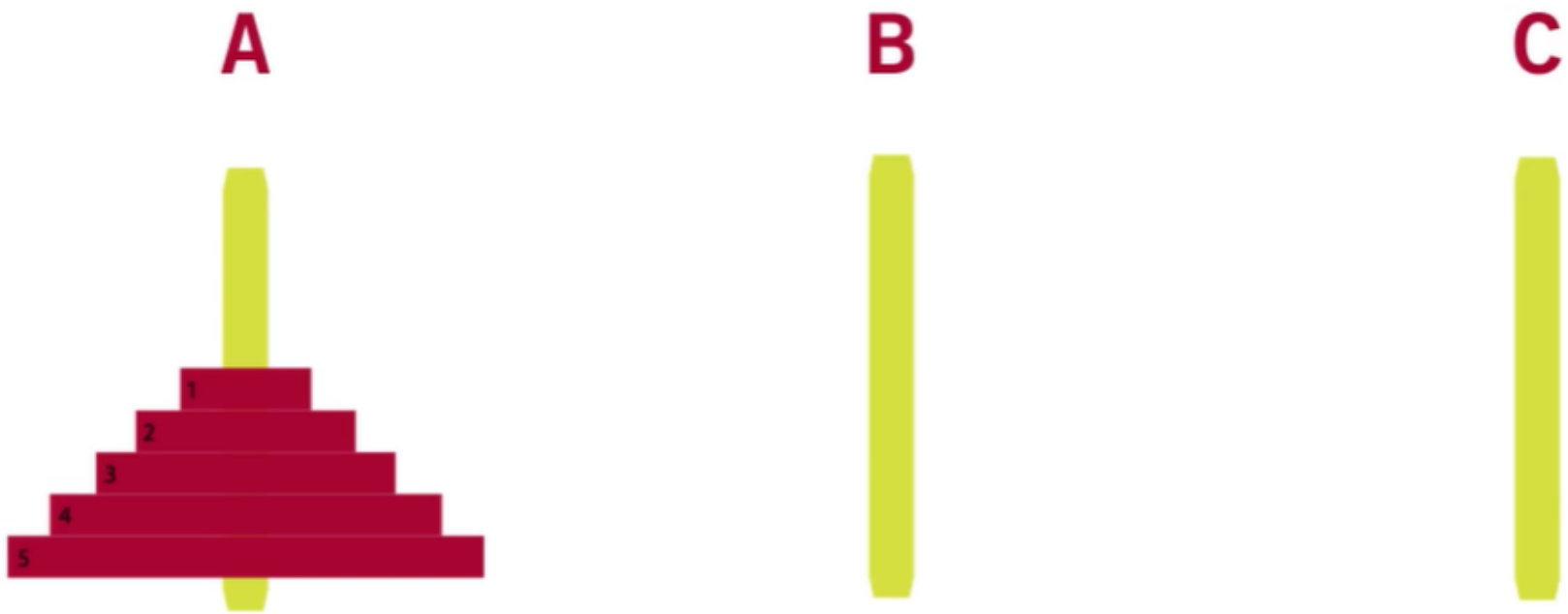
Tower Of Hanoi: Output (3/4)

```
const moveDisks = function (n, from, to, spare) {
  if (n == 1) {
    console.log("Move disk:" + n + " from:" + from + " to:" + to);
  } else {

    moveDisks(n - 1, from, spare, to);
    console.log("Move disk:" + n + " from:" + from + " to:" + to);
    moveDisks(n - 1, spare, to, from);
  }
};
const nDisks = 4;

moveDisks(nDisks, "A", "C", "B");
```

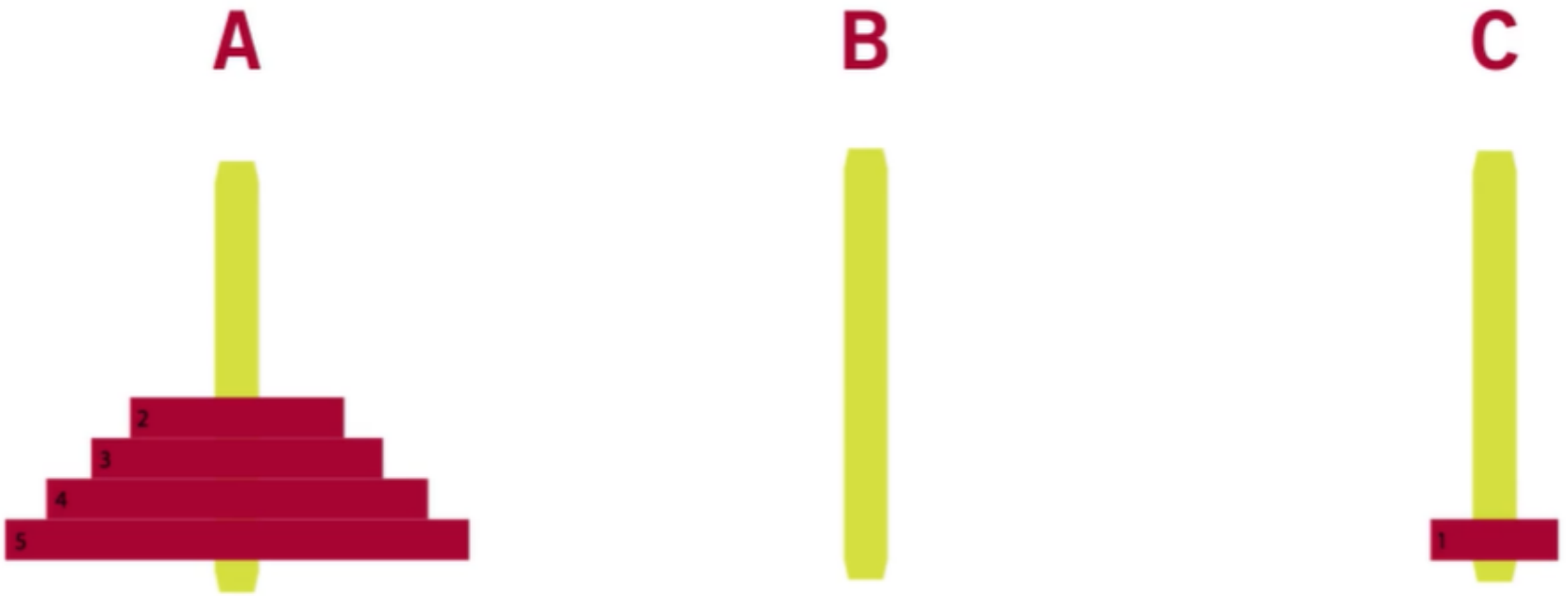
Tower Of Hanoi: Output (4/4)

```
√ hanoitower % node hanoi.js                                    master
Move disk:1 from:A to:B
Move disk:2 from:A to:C
Move disk:1 from:B to:C
Move disk:3 from:A to:B
Move disk:1 from:C to:A
Move disk:2 from:C to:B
Move disk:1 from:A to:B
Move disk:4 from:A to:C
Move disk:1 from:B to:C
Move disk:2 from:B to:A
Move disk:1 from:C to:A
Move disk:3 from:B to:C
Move disk:1 from:A to:B
Move disk:2 from:A to:C
Move disk:1 from:B to:C
√ hanoitower %                                                  master
```

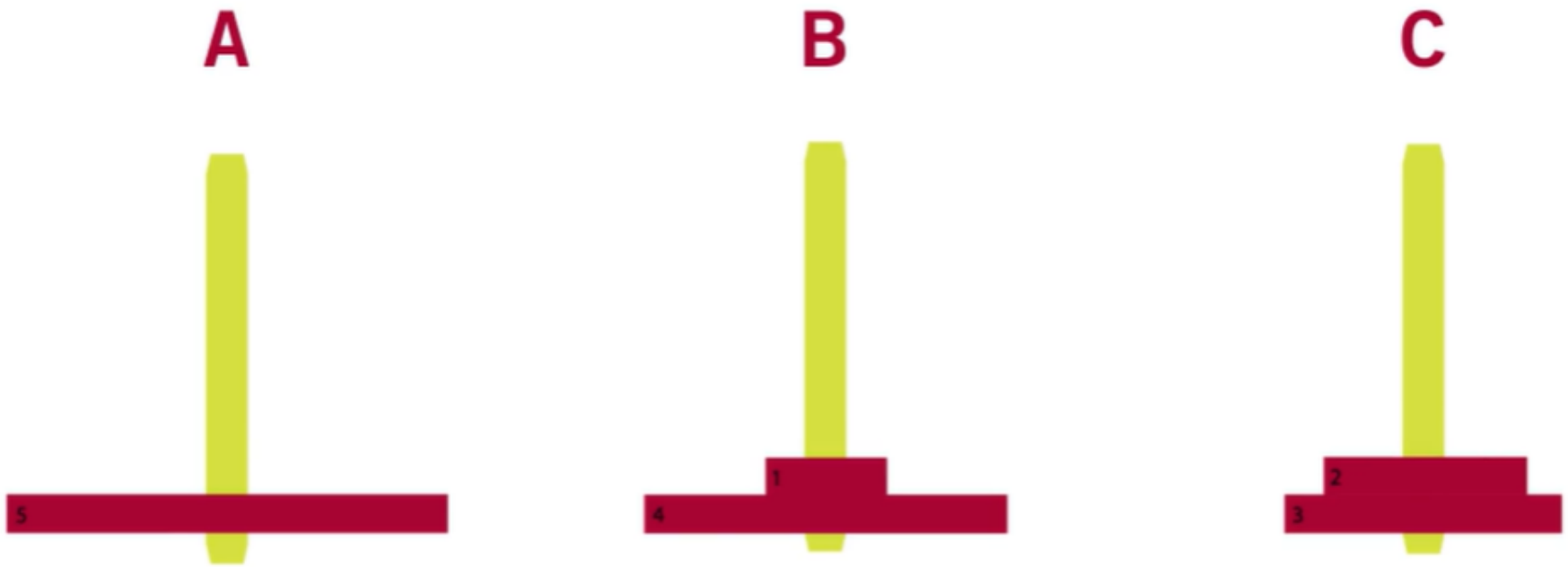
Hanoi Tower Solution (1/7)



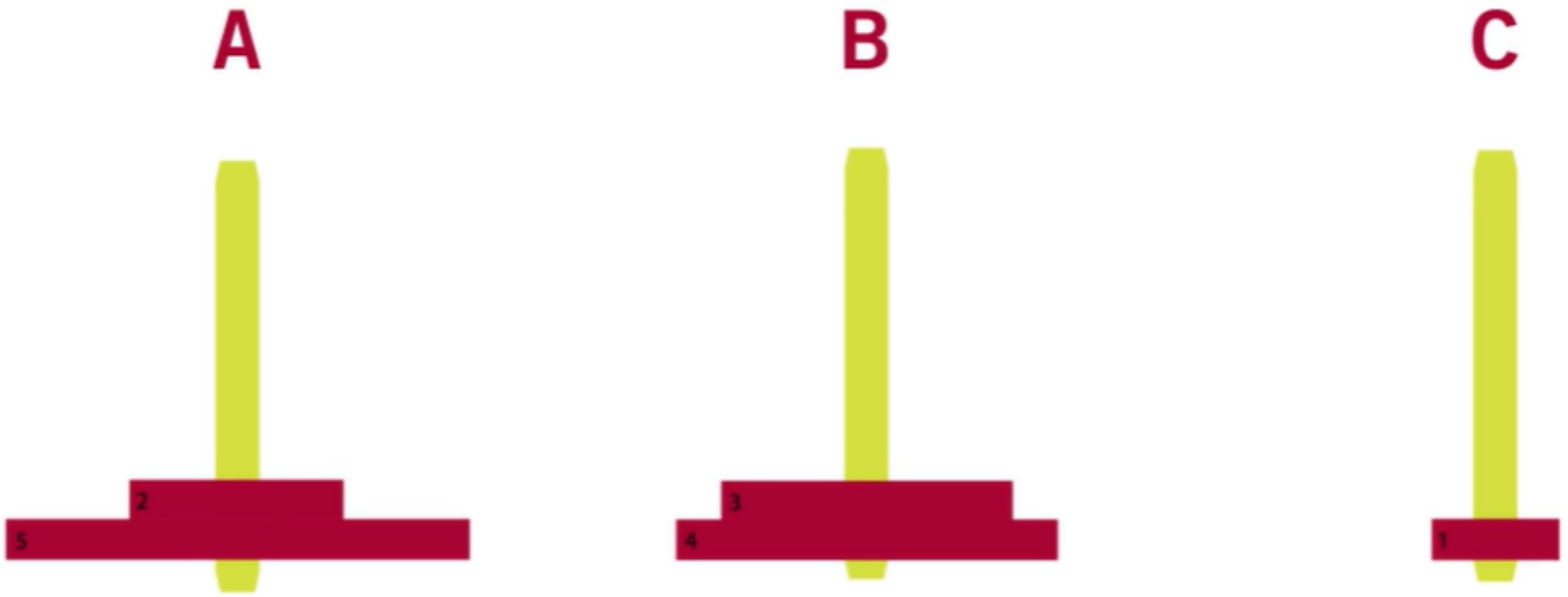
Hanoi Tower Solution (2/7)



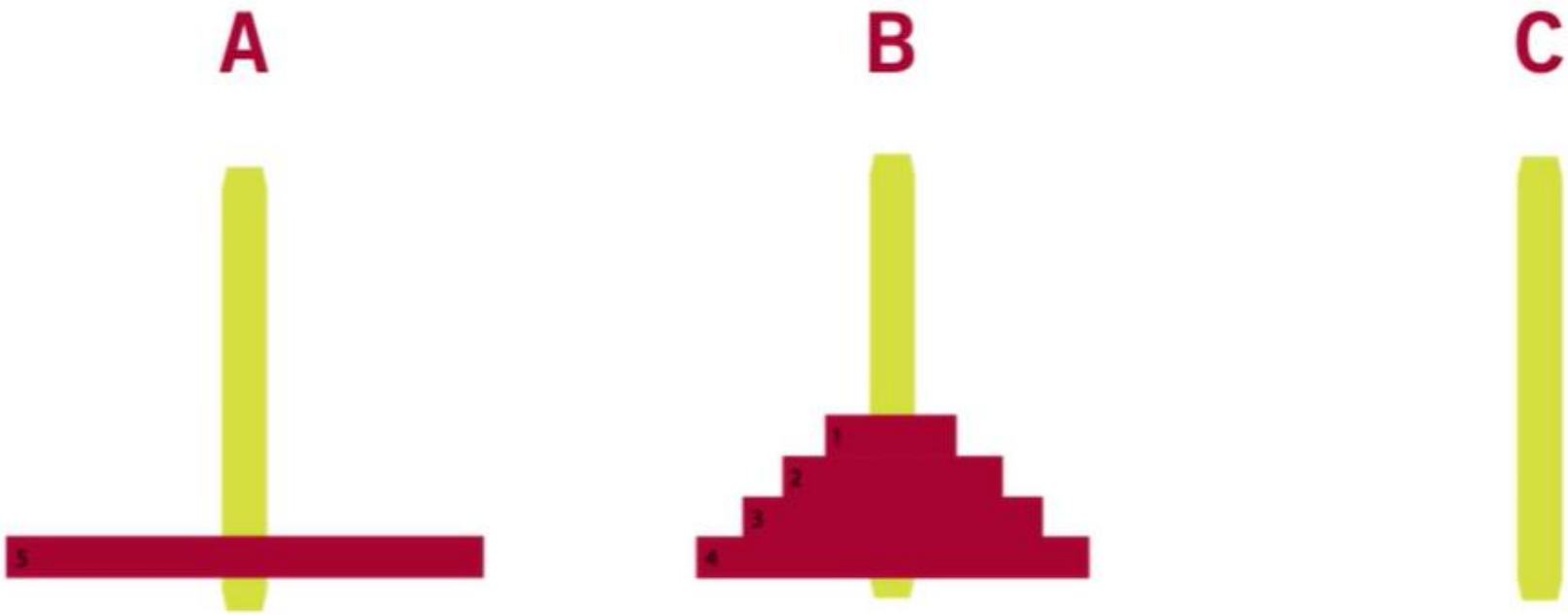
Hanoi Tower Solution (3/7)



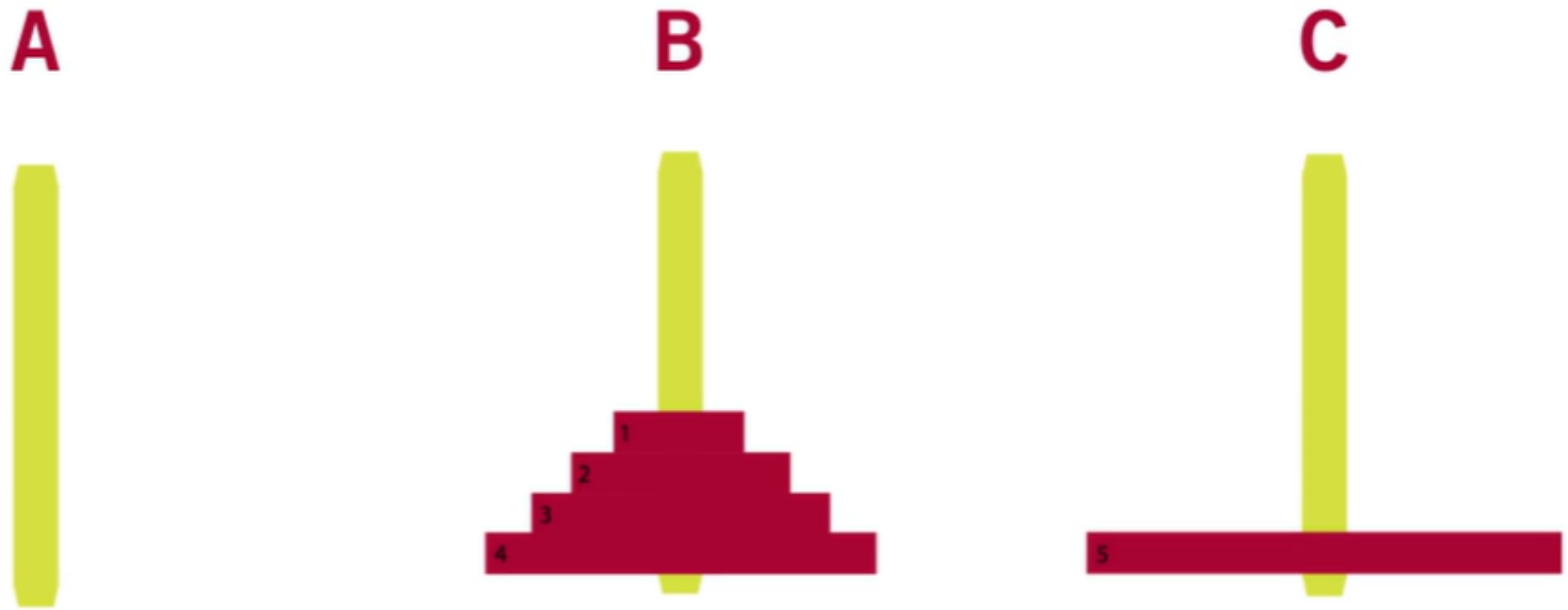
Hanoi Tower Solution (4/7)



Hanoi Tower Solution (5/7)



Hanoi Tower Solution (6/7)



Hanoi Tower Solution (7/7)