Lecture 4b
Hash Functions I

Stefan Dziembowski
www.crypto.edu.pl/Dziembowski

University of Warsaw

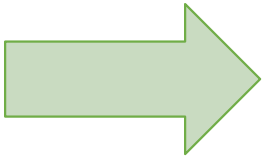
24.10.18 **version 1.0**

Secure communication

	encryption	**authentication**
private key	1 private key encryption	2 private key authentication
public key	3 public key encryption	4 signatures

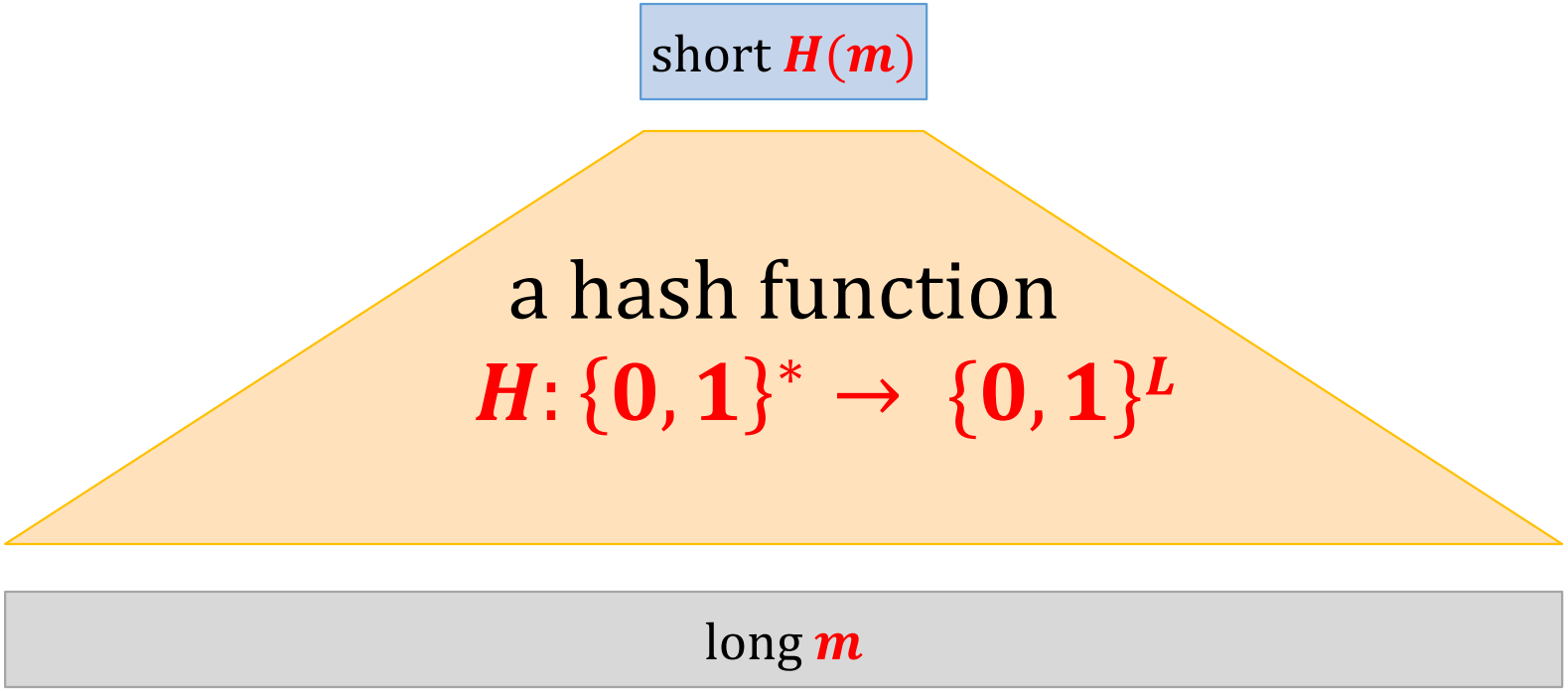
we will discus this on the next lecture, but first, we talk about the hash functions

Plan

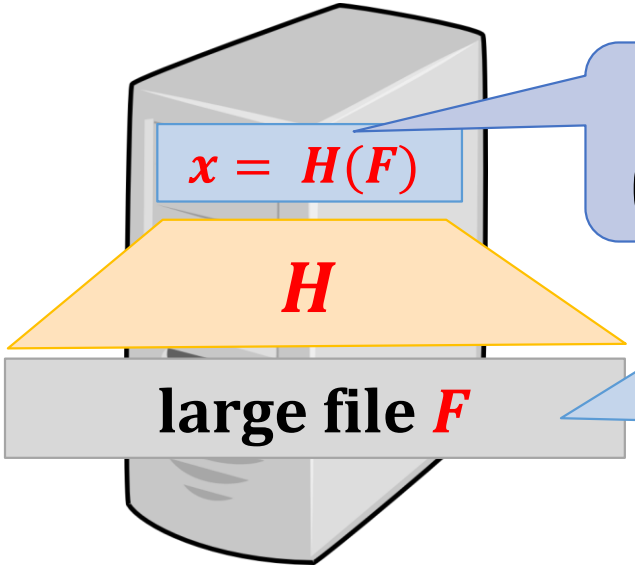


1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction

Hash functions



Example of an application: file fingerprinting



"**fingerprint**"
(publish this through a **secure channel**)

everybody who downloads F can verify its correctness by computing $F(H)$ and comparing with x.

Example (ubuntu.com):

64-bit PC (amd64, x86_64) (Recommended)

1. Ubuntu 15.10 "Wily Werewolf" 40MB (MD5: d1498749e073ef7fd09f84478f299bea, SHA1: aa659499dc300fe2be00d756180793093cc15014)
2. Ubuntu 15.04 "Vivid Vervet" 40MB (MD5: 3b00a4573b11fb1f85eaa05918971789, SHA1: 97282a3b066de4ee4c9409979737f3911f95ceab)

What properties should a hash function H have?

Minimal requirement: second preimage-resistance:

More precisely, the following problem should be **hard for any efficient adversary A**:

- **given**: "random" $m \in \{0,1\}^*$
- **find**: $m' \neq m$ such that

$$H(m) = H(m')$$

Q: Is it enough?

Second preimage resistance may be in many cases be too weak

What if the adversary can somehow influence the choice of m?

For example: Ubuntu has many contributors. What if one of them is malicious?

Idea: modify the definition by allowing the adversary to choose m himself.

New game for the adversary:

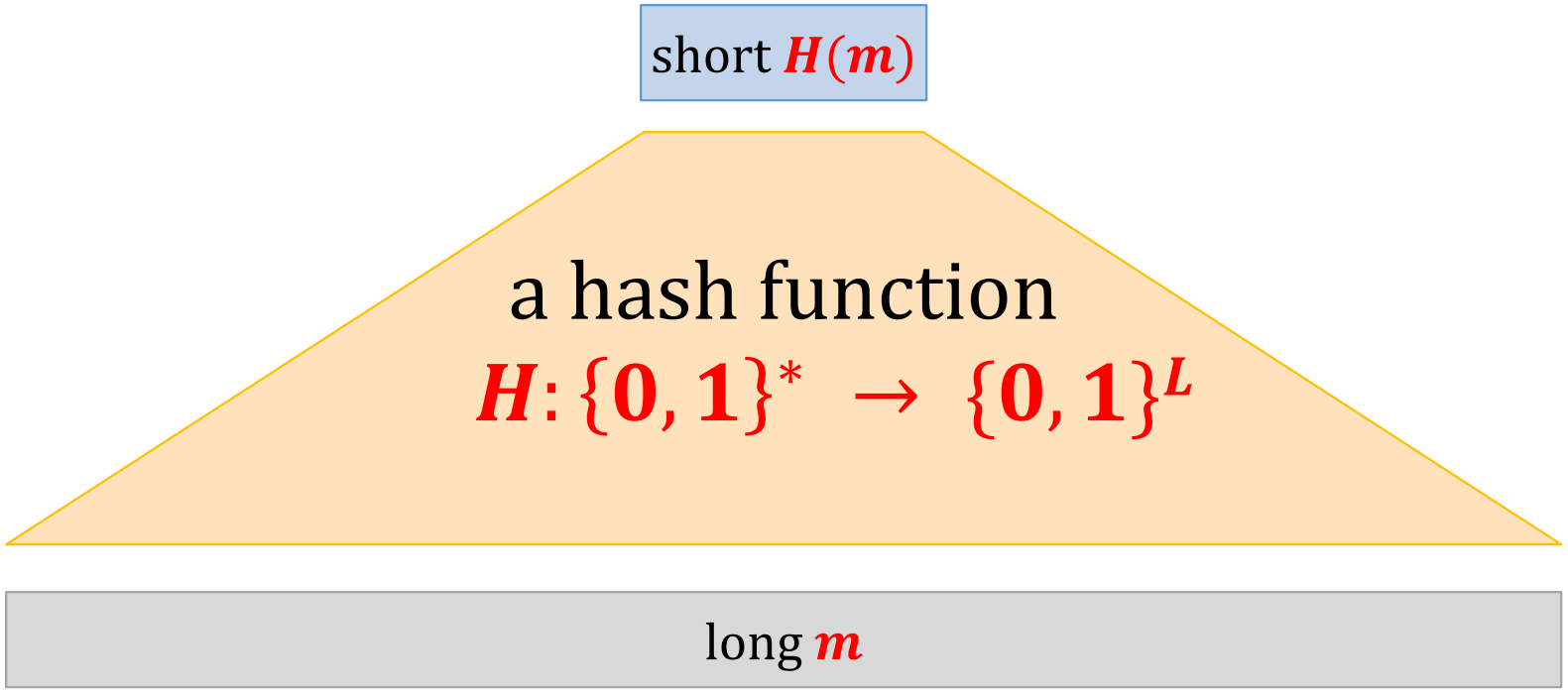
if this problem is hard then a function is called "**collision-resistant**"

find: m
find: $m' \neq m$ such that $H(m) = H(m')$

$\Leftrightarrow$

find: $m \neq m'$ such that $H(m) = H(m')$

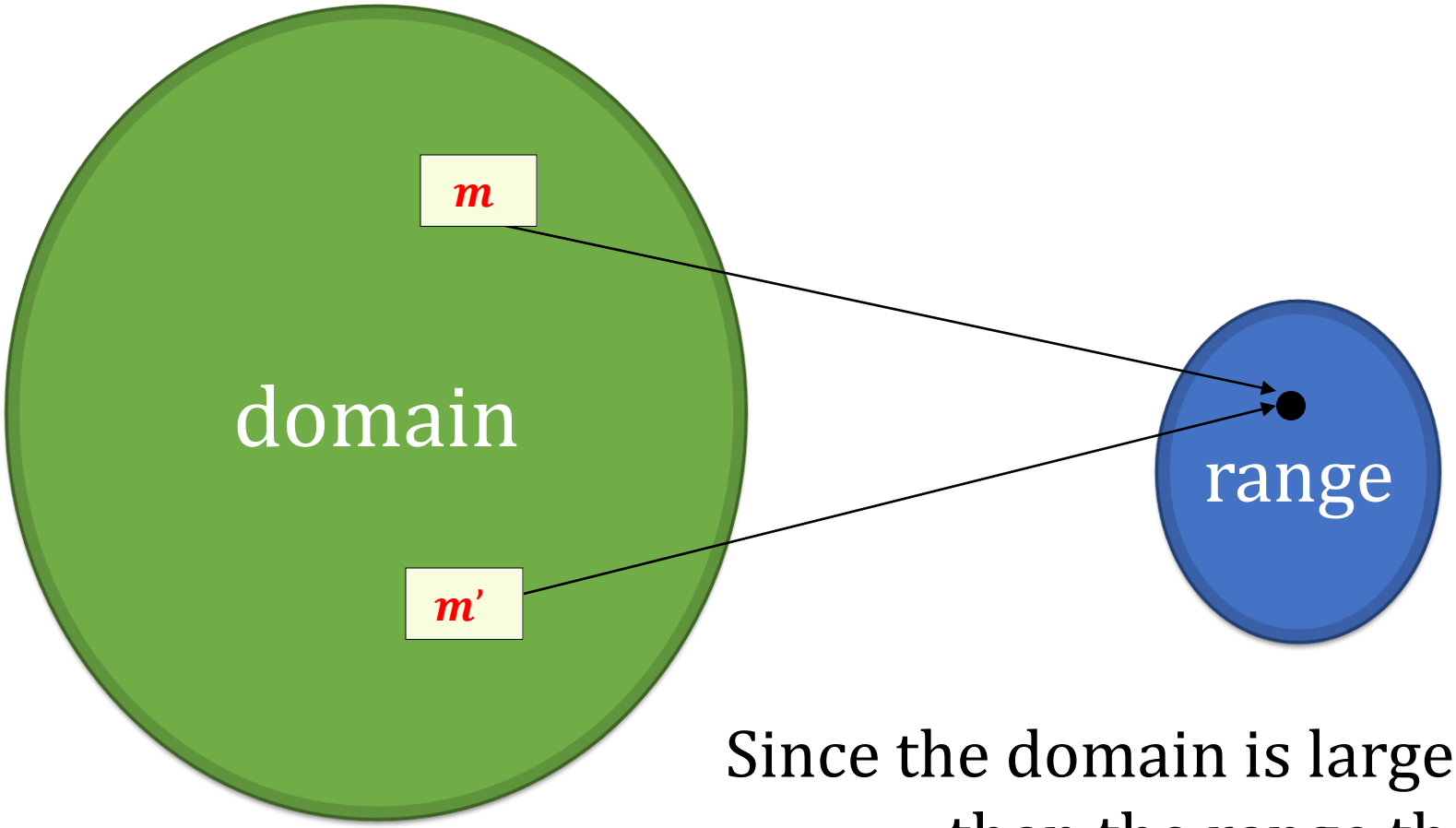
Collision-resistant hash functions



Requirement: it should be hard to find a pair (m, m') such that

$$H(m) = H(m')$$

Collisions always exist



Since the domain is larger than the range the collisions have to exist.

“Practical definition”

H is a **collision-resistant hash function** if it is “*practically impossible to find collisions in* ***H***”.

Popular hash functions (we will present them in more detail on the next lecture):

- **MD5** (now considered broken
- **SHA1** (also has weaknesses)**, SHA256**

} based on **the Merkle-Damgård transformation**

- **Keccak** ⇦ based on the **sponge construction**

Hash functions can also be constructed using mathematical tools like number theory.

How to formally define “collision resitance”?

Idea: Say something like: H is a **collision-resistant hash function** if

$\forall$ efficient adversary A $\quad P(A \text{ finds a collision in } H)$ is small

Problem

For a fixed H there **always exist** a constant-time algorithm that “finds a collision in H” in **constant time**.

It may be hard to **find** such an algorithm, but it always exists!

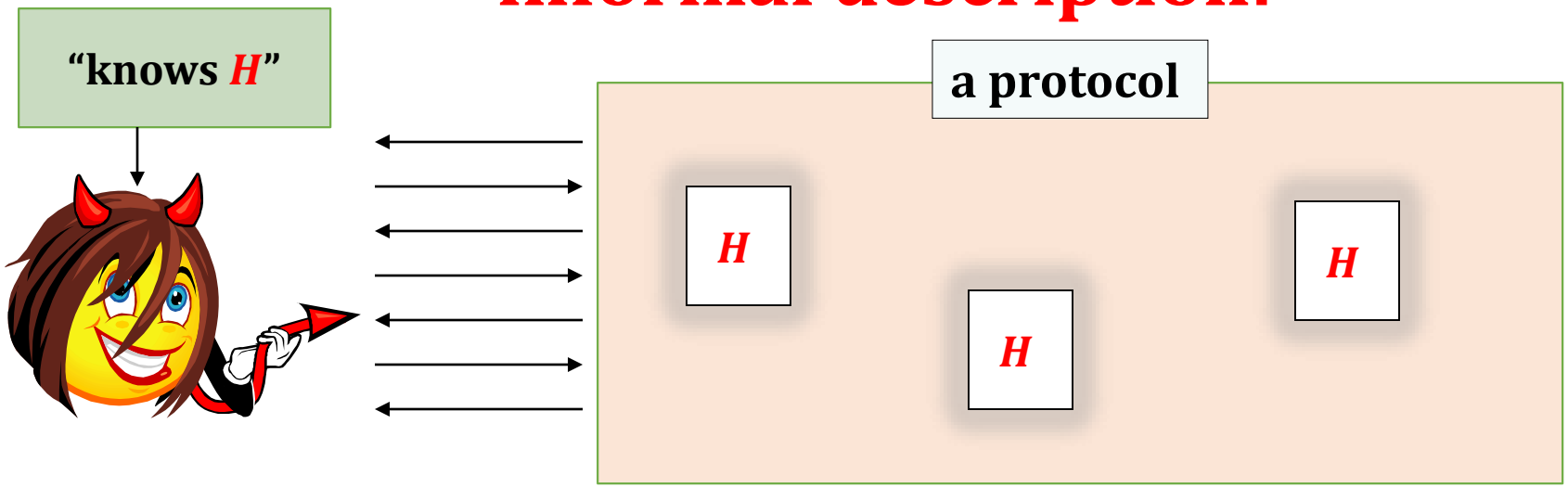
Solution

When we prove theorems we will always consider

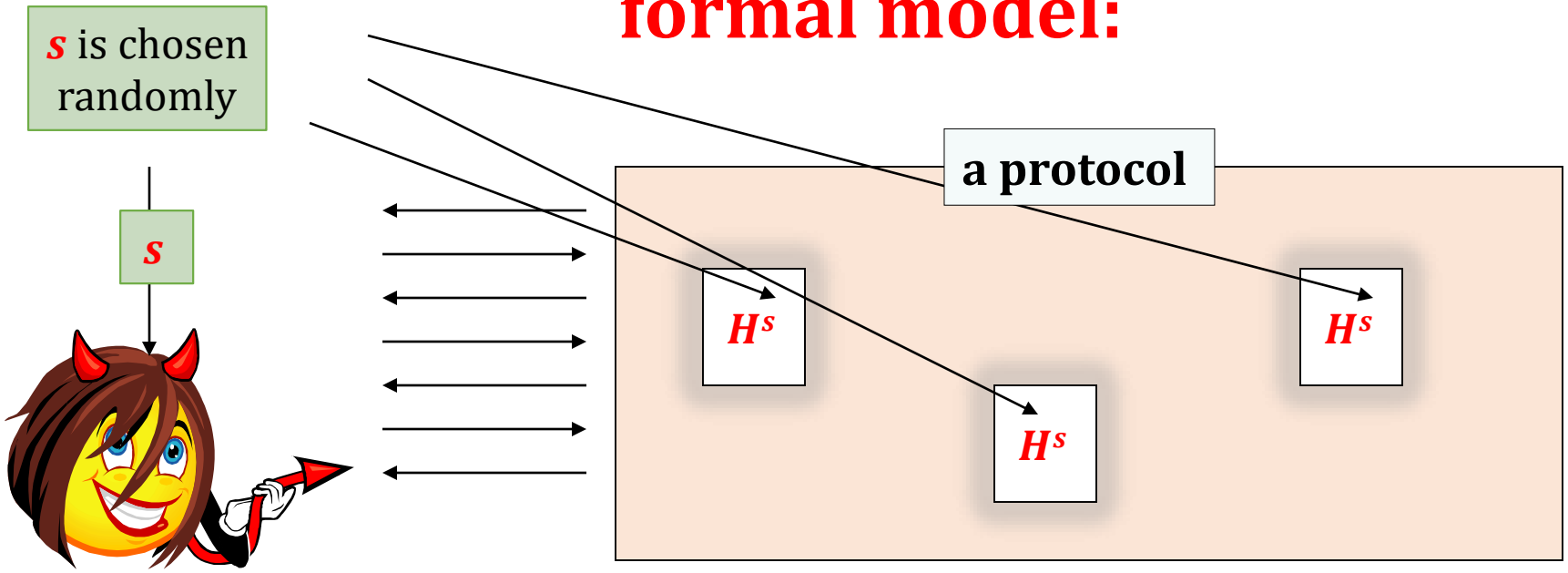
families of hash functions indexed by a key s:

$$\{H^s\}_{s \in \text{keys}}$$

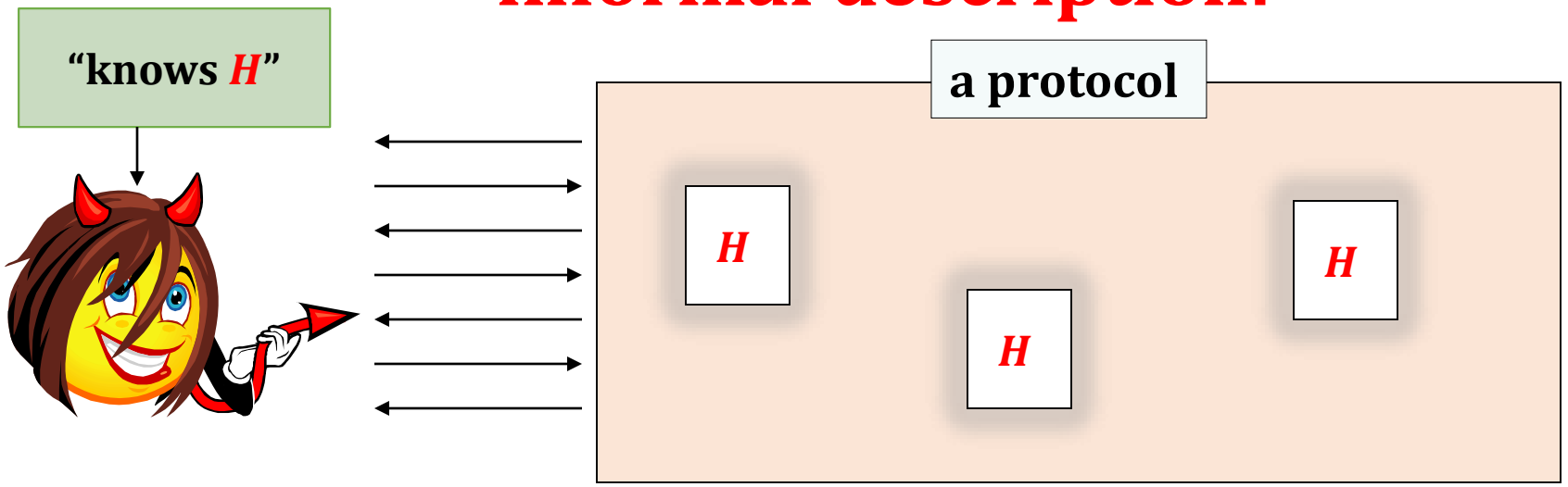
informal description:



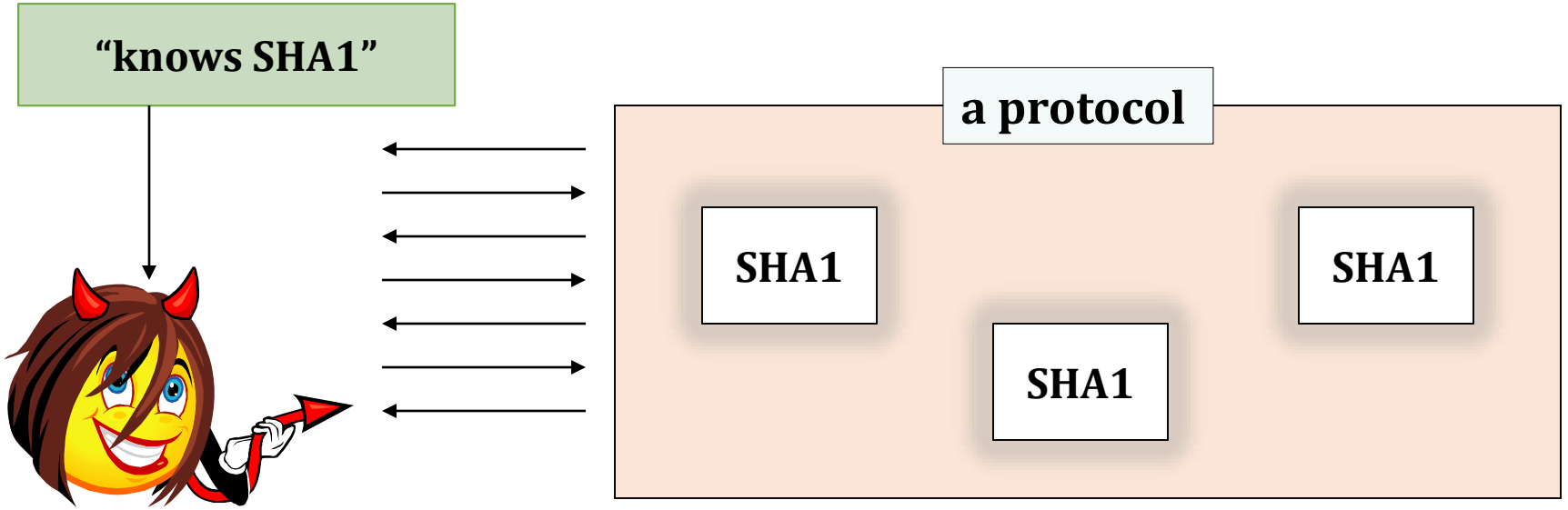
formal model:



informal description:



real-life implementation (example):



Hash functions – the functional definition

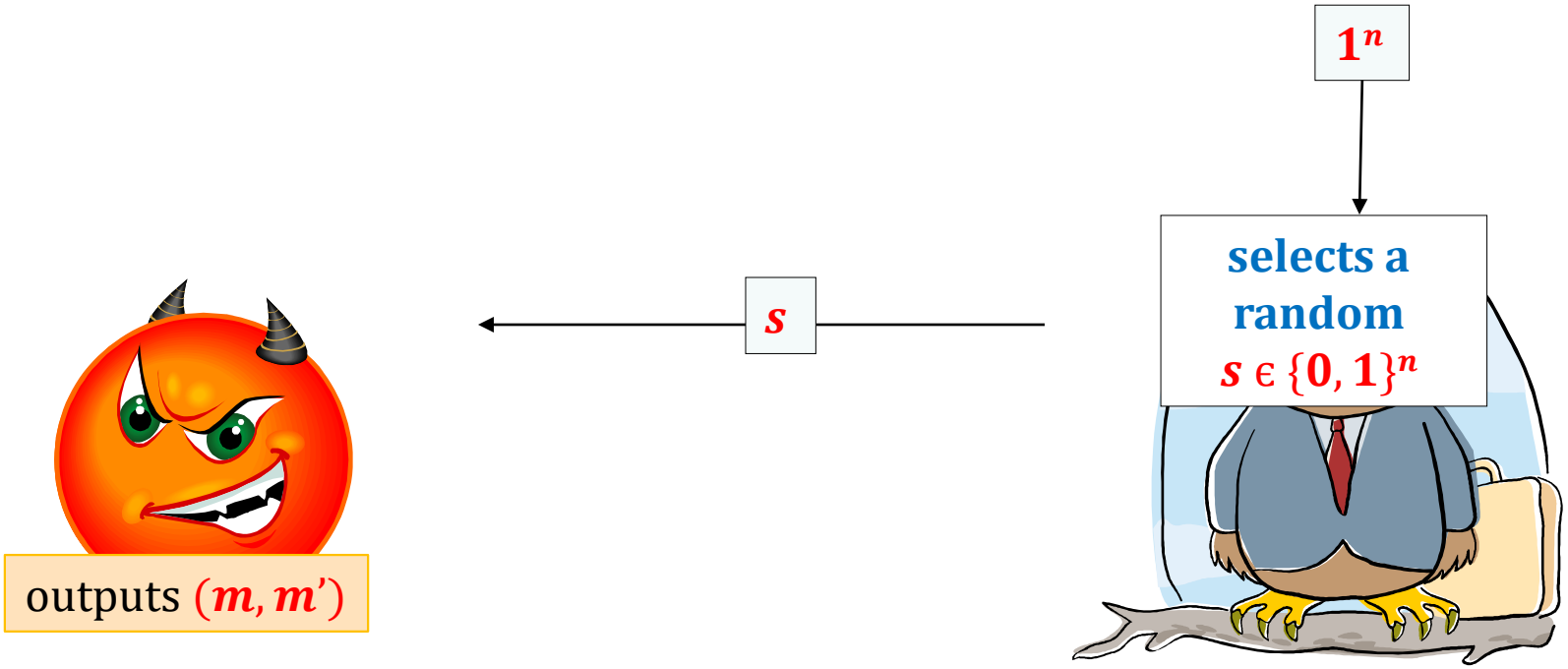
A **hash function** is a probabilistic polynomial-time algorithm H such that:

H takes as input a key $s \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ and outputs a string

$$H^s(m) \in \{0,1\}^{L(n)}$$

where $L(n)$ is some fixed function.

Hash functions – the security definition [1/2]



We say that adversary A breaks the function H if

$$H^s(m) = H^s(m').$$

Hash functions – the security definition [2/2]

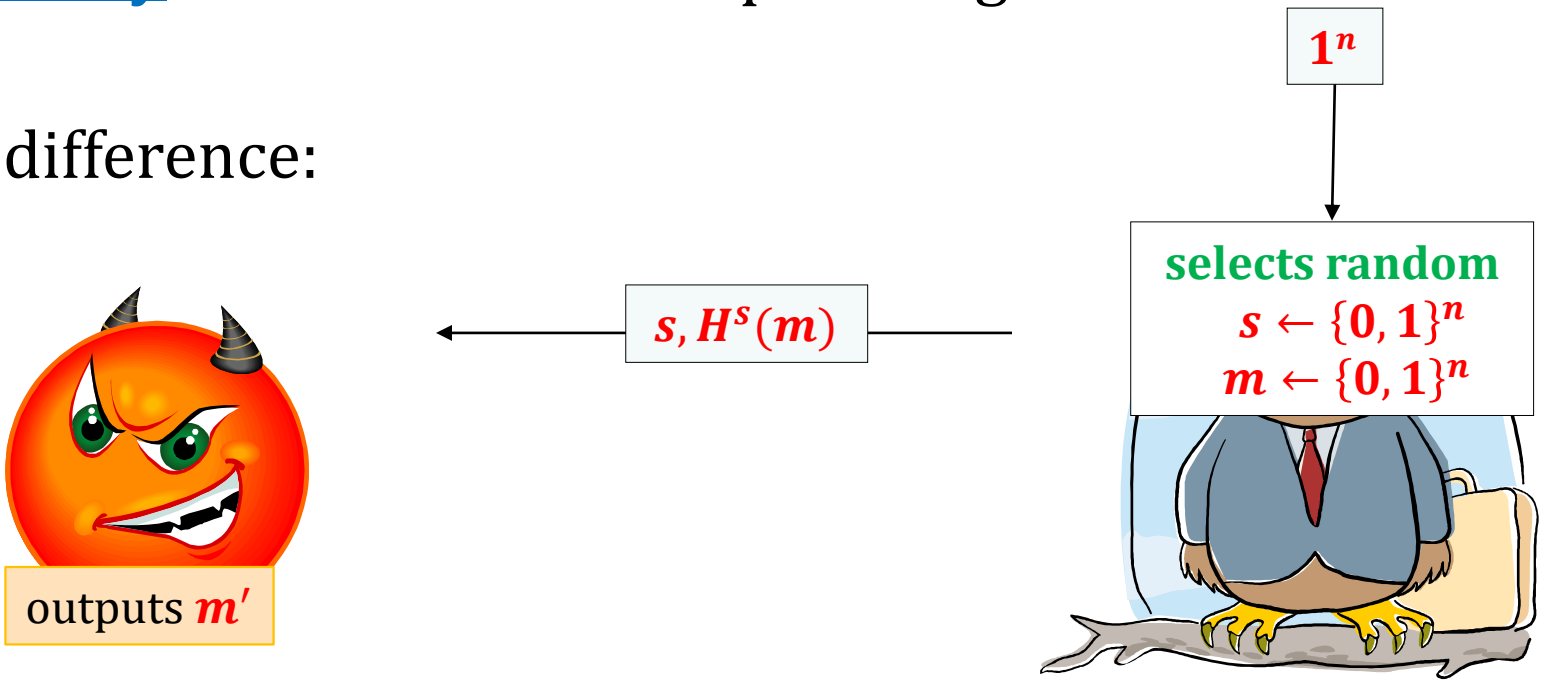
H is a **collision-resistant hash function** if

$\forall$ polynomial-time adversary ***A*** **$P(A$ breaks $H)$** is negligible

A weaker requirement: **pre-image resistance**

Intuitively: "it's hard to find a pre-image of H^s"

Main difference:

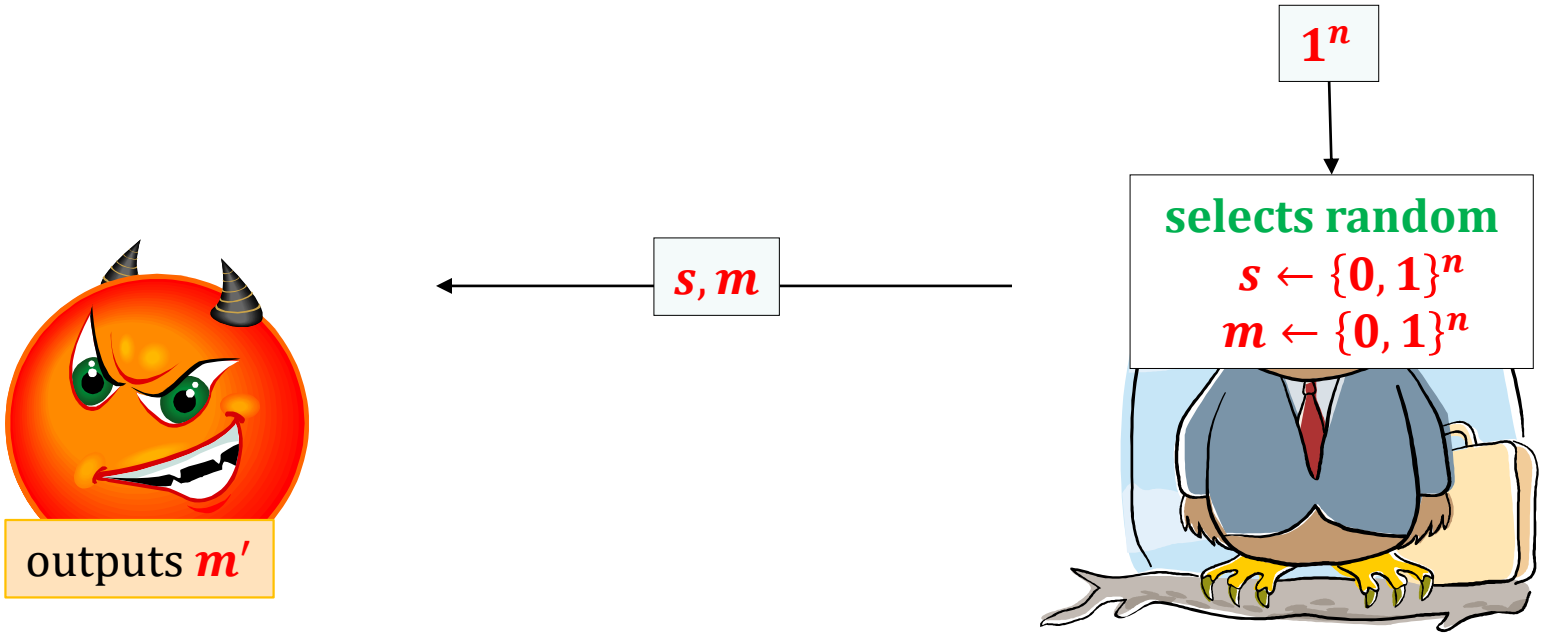


We say that adversary A breaks the function H if

$$H^s(m) = H^s(m')$$

Yet another requirement: **second pre-image resistance**

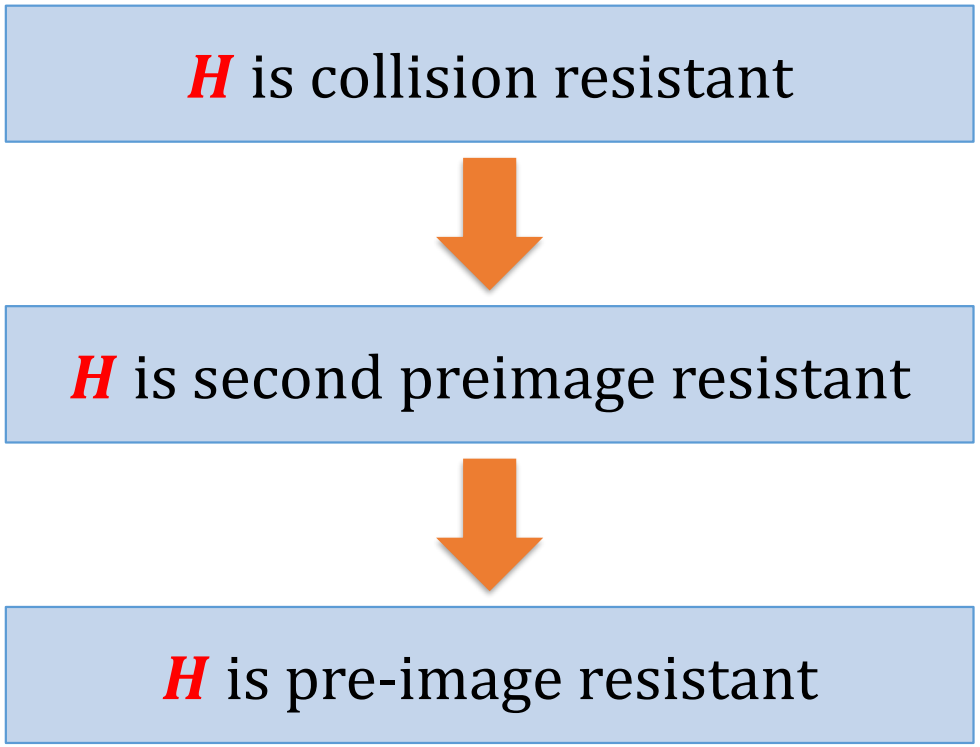
Intuitively: "it's hard to find a second pre-image of H^s"



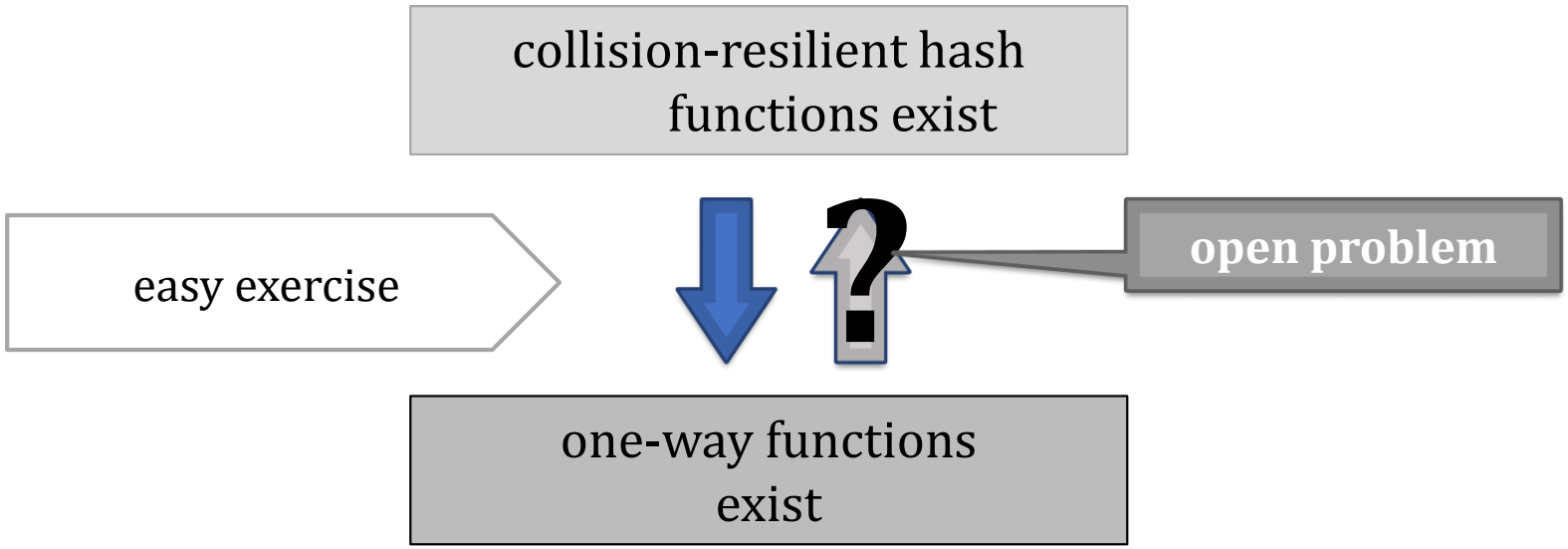
We say that adversary A breaks the function H if $m' \neq m$ and $H^s(m) = H^s(m')$

Fact

The following implications hold:



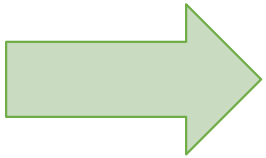
Do collision-resilient hash functions belong to minicrypt?



[D. Simon: Finding Collisions on a One-Way Street: Can Secure Hash Functions Be Based on General Assumptions? 1998]:

there is no "black-box reduction".

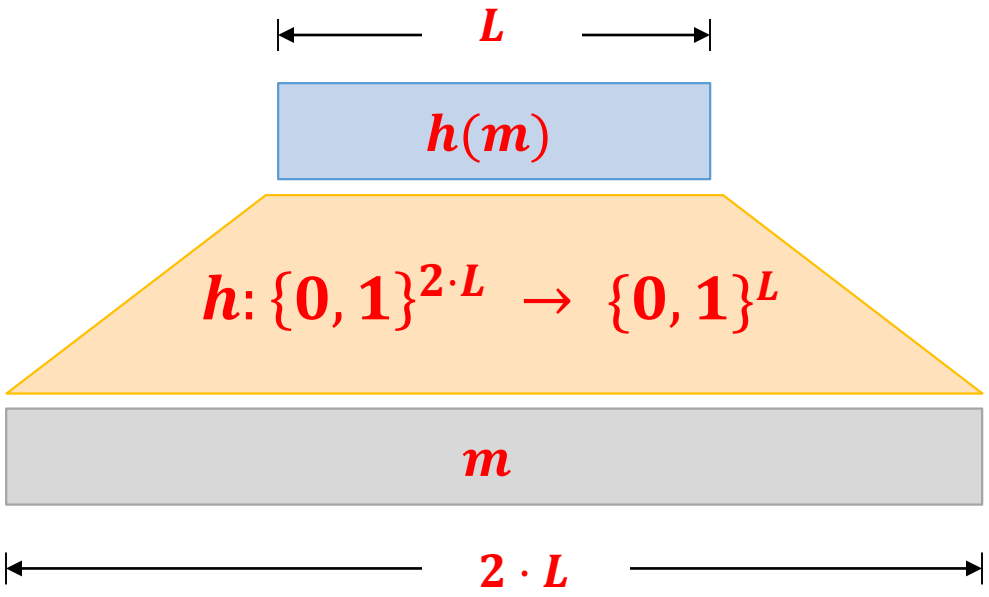
Plan



1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction

A common method for constructing hash functions

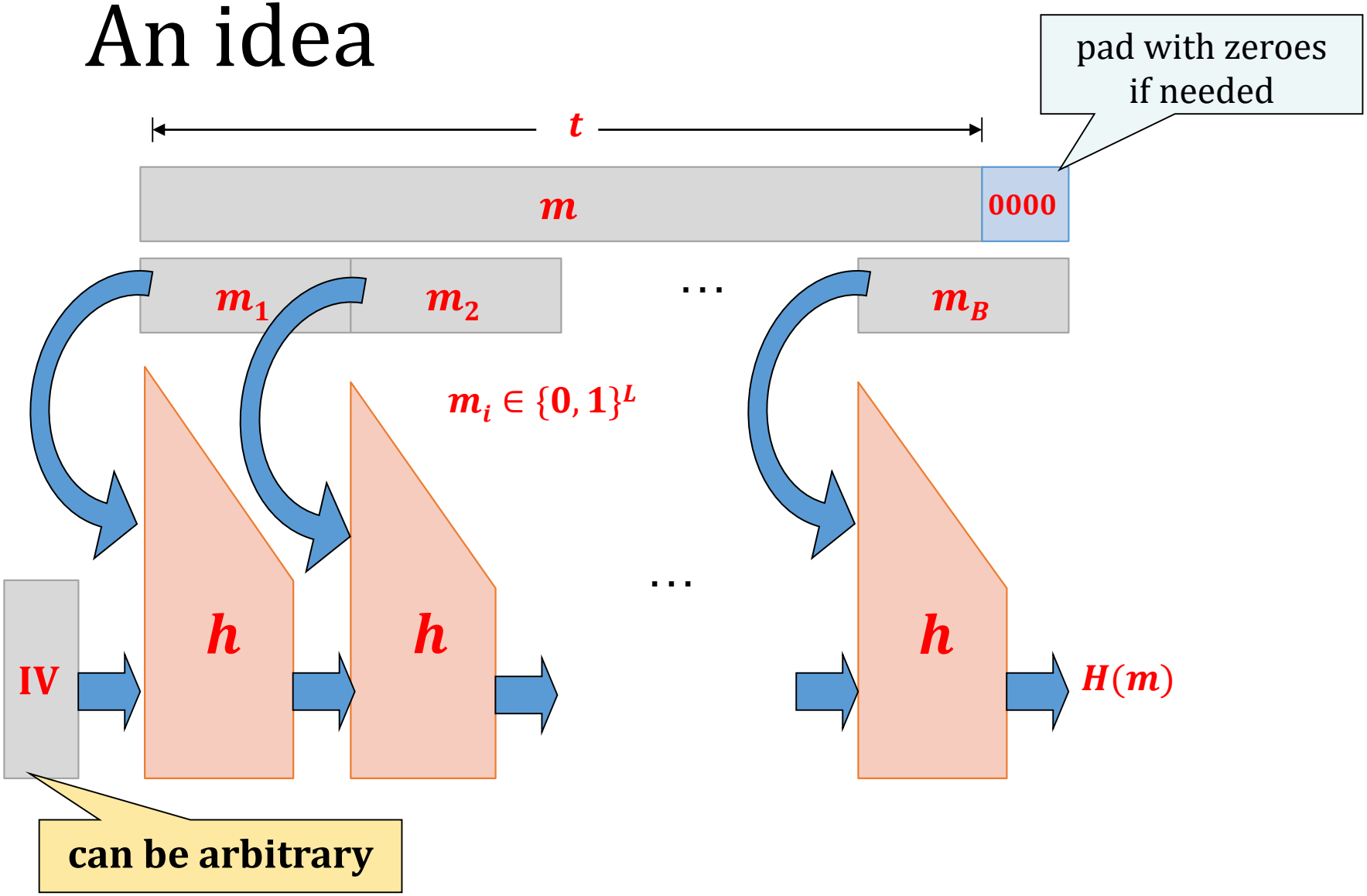
1. Construct a "***fixed-input-length***" collision-resistant hash function



   Call it: a collision-resistant **compression function**.

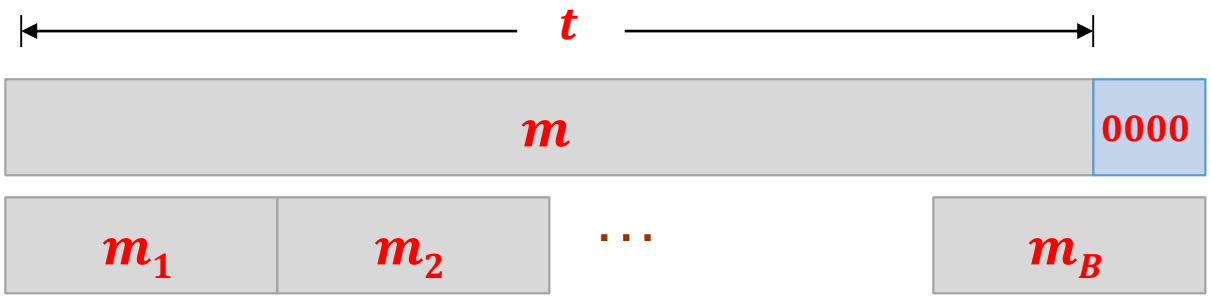
2. Use it to construct a hash function.

An idea



This doesn't work...

Why is it wrong?

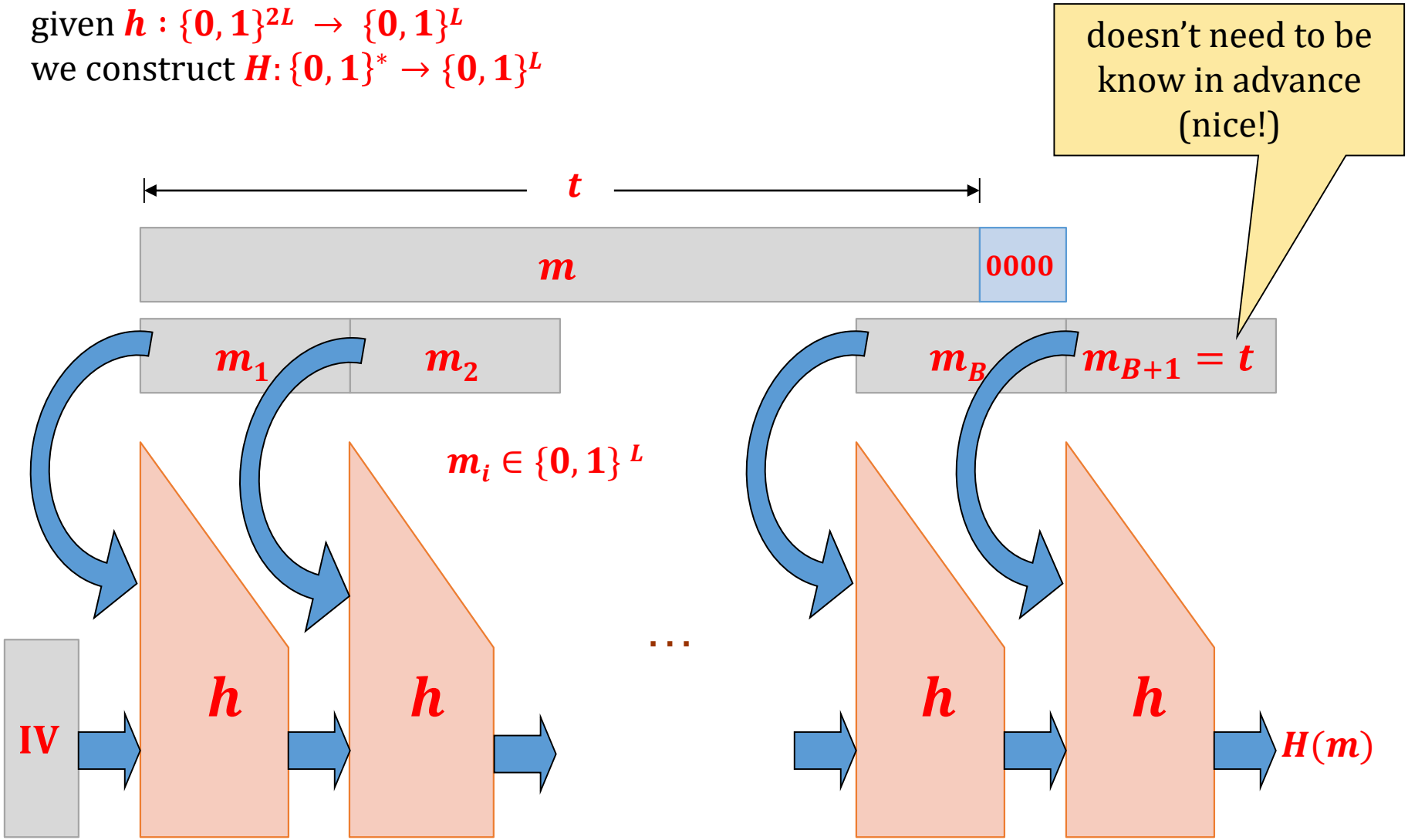


If we set $m' = m \,||\, 0000$ then $H(m') = H(m)$.

Solution: add a block encoding "t".

Merkle-Damgård transform

given $h : \{0,1\}^{2L} \rightarrow \{0,1\}^{L}$
we construct $H: \{0,1\}^{*} \rightarrow \{0,1\}^{L}$



This construction is secure

We would like to prove the following:

Theorem

If

$$h: \{0,1\}^{2L} \rightarrow \{0,1\}^{L}$$

is a collision-resistant **compression** function

then

$$H: \{0,1\}^{*} \rightarrow \{0,1\}^{L}$$

is a collision-resistant **hash** function.

But wait….
It doesn't make sense…

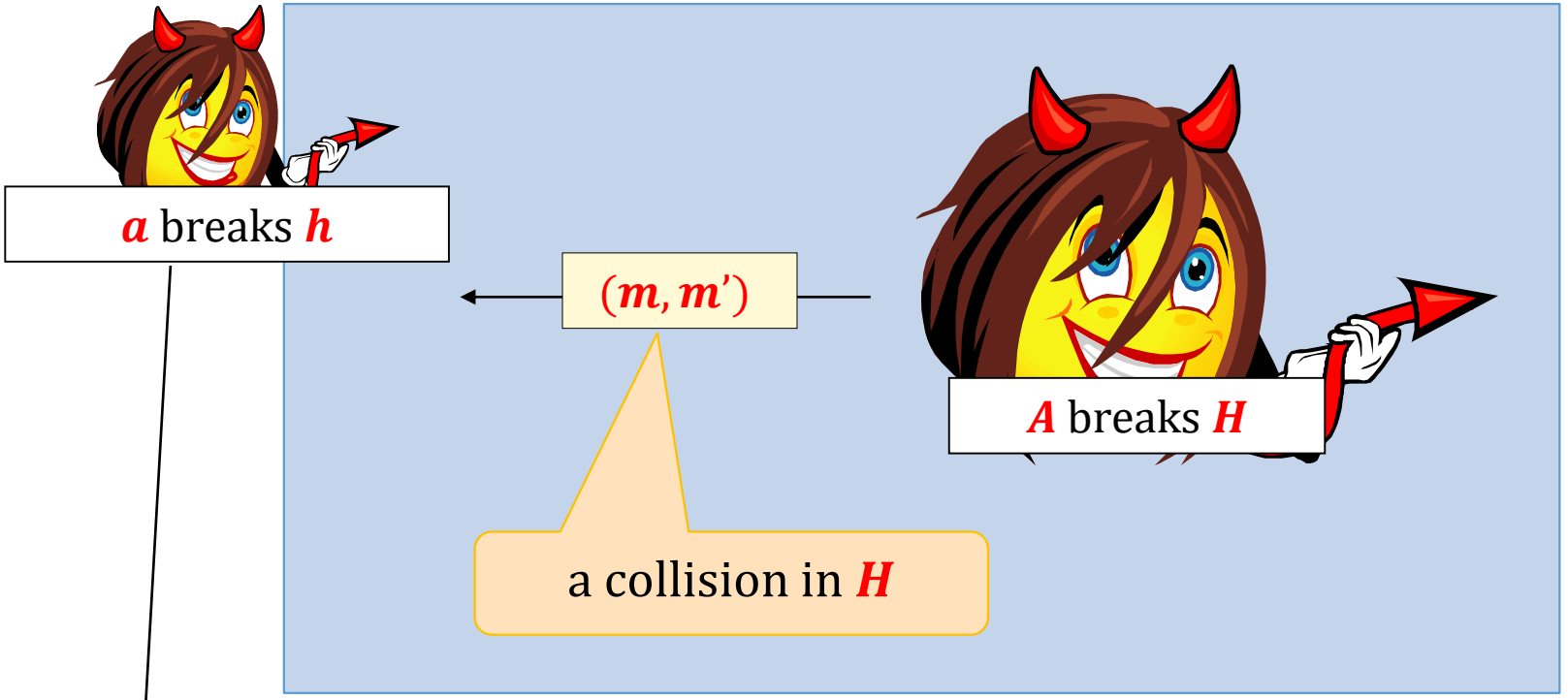
What to do?

To be formal, we would need to consider

families of functions
h and ***H***
indexed by key ***s***

Let's stay on the **informal level** and argue that:

"if one can find a collision in ***H*** then one can find a collision in ***h***"


a breaks h
(m, m')
A breaks H
a collision in H
outputs a collision (x, y) in h

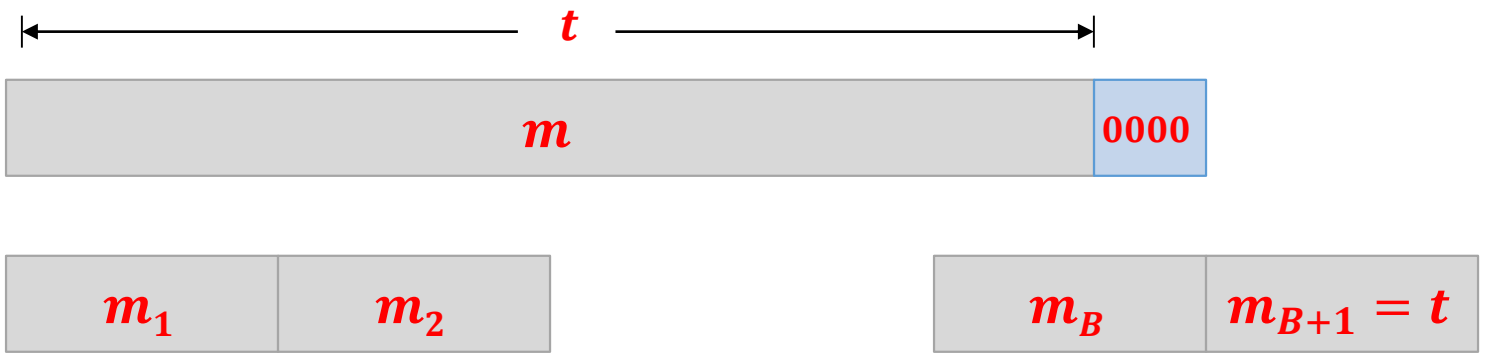
How to compute a collision (x, y) in h from a collision (m, m') in H?

We consider two cases:

1. $|m| = |m'|$

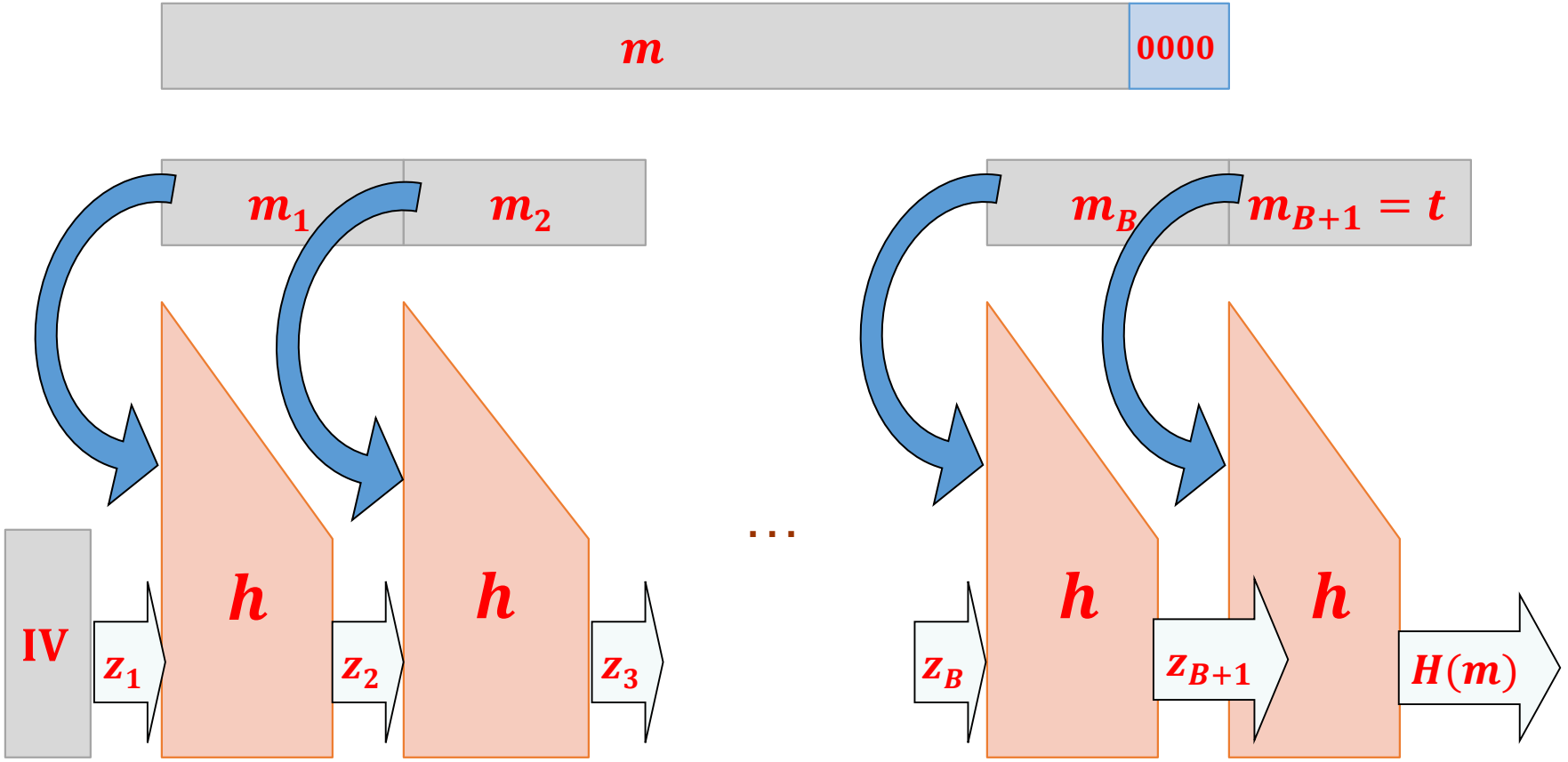
2. $|m| \neq |m'|$

Case 1: $|m| = |m'|$



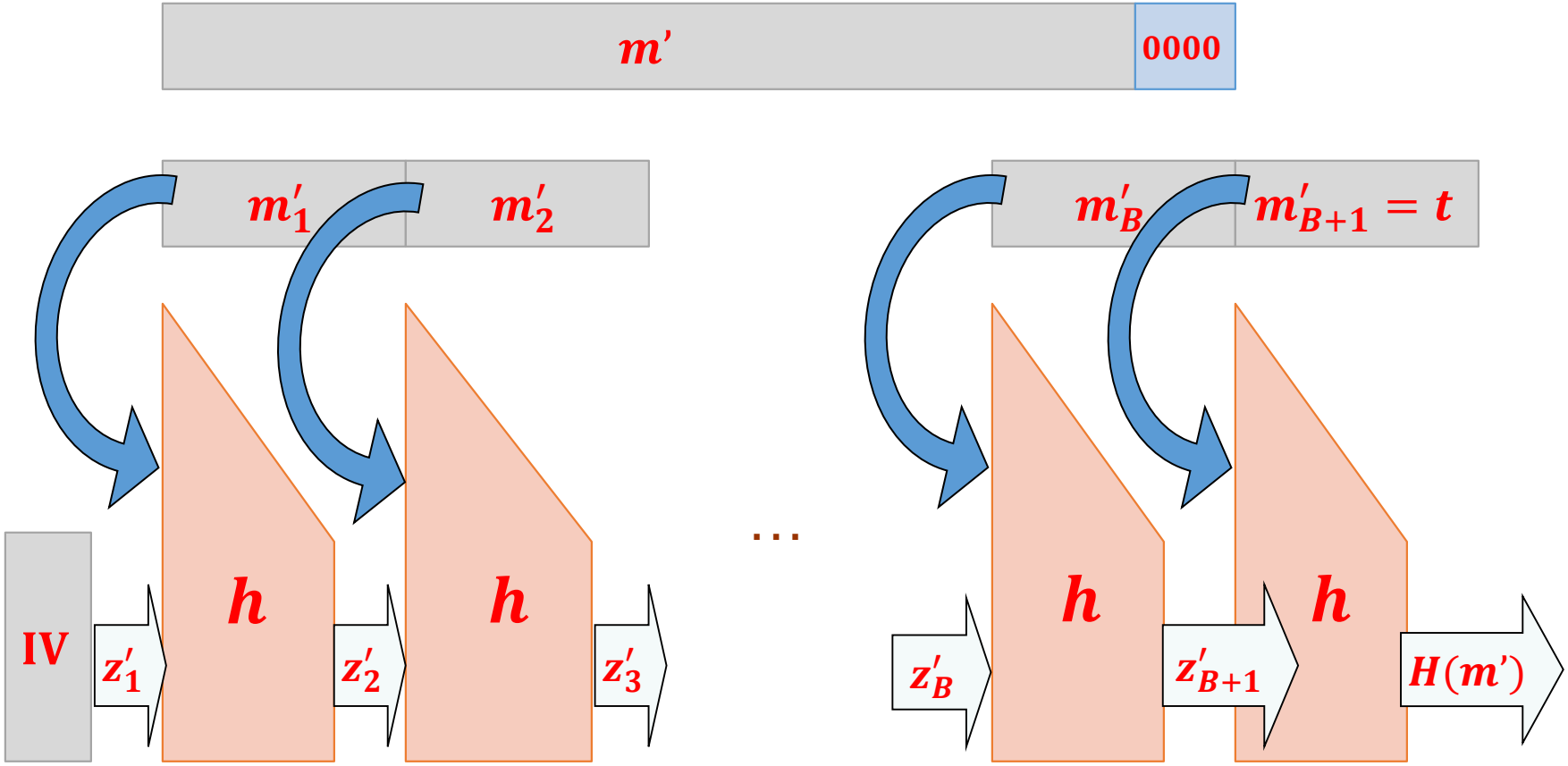
$|m| = |m'|$

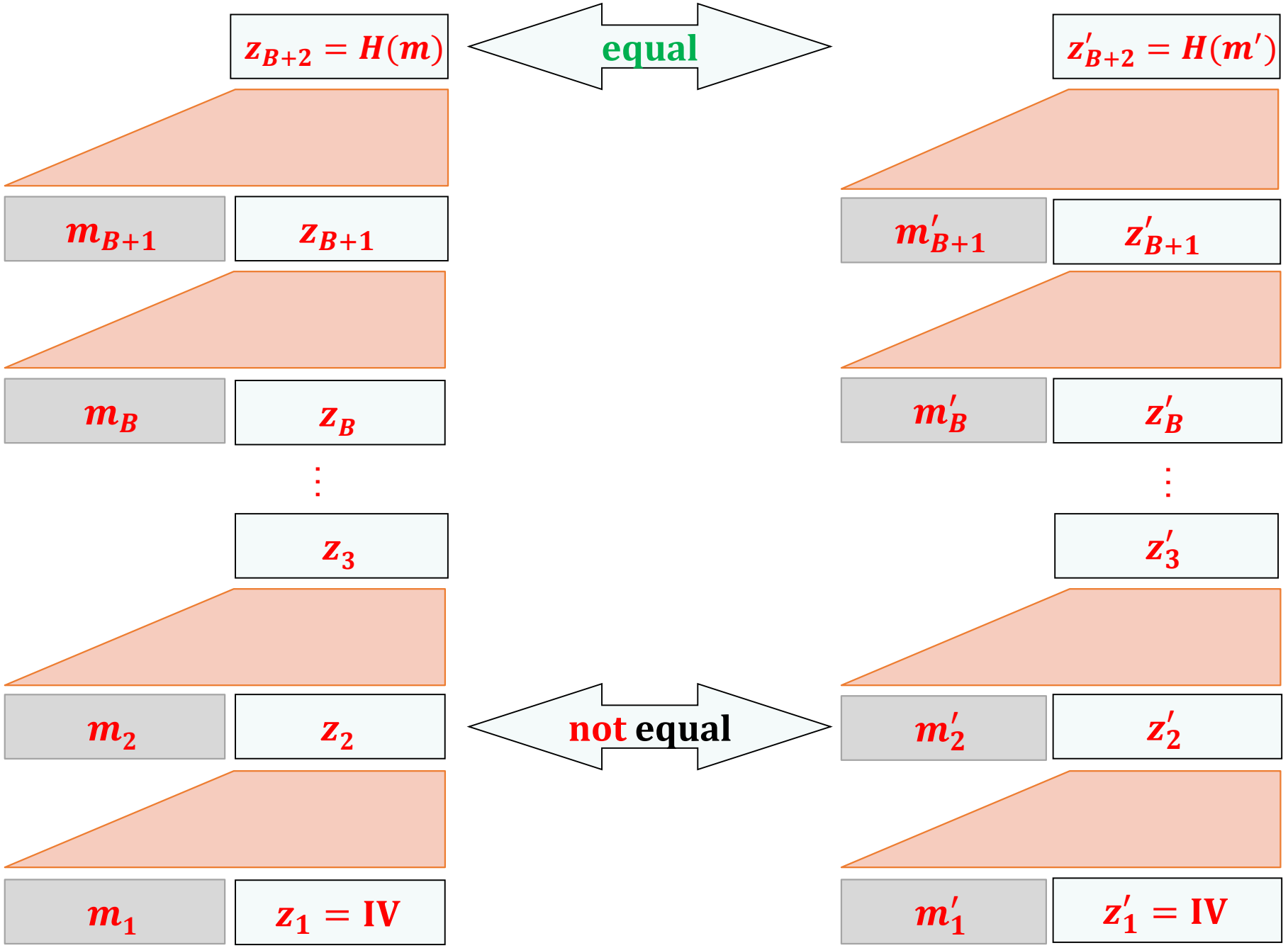
Some notation:

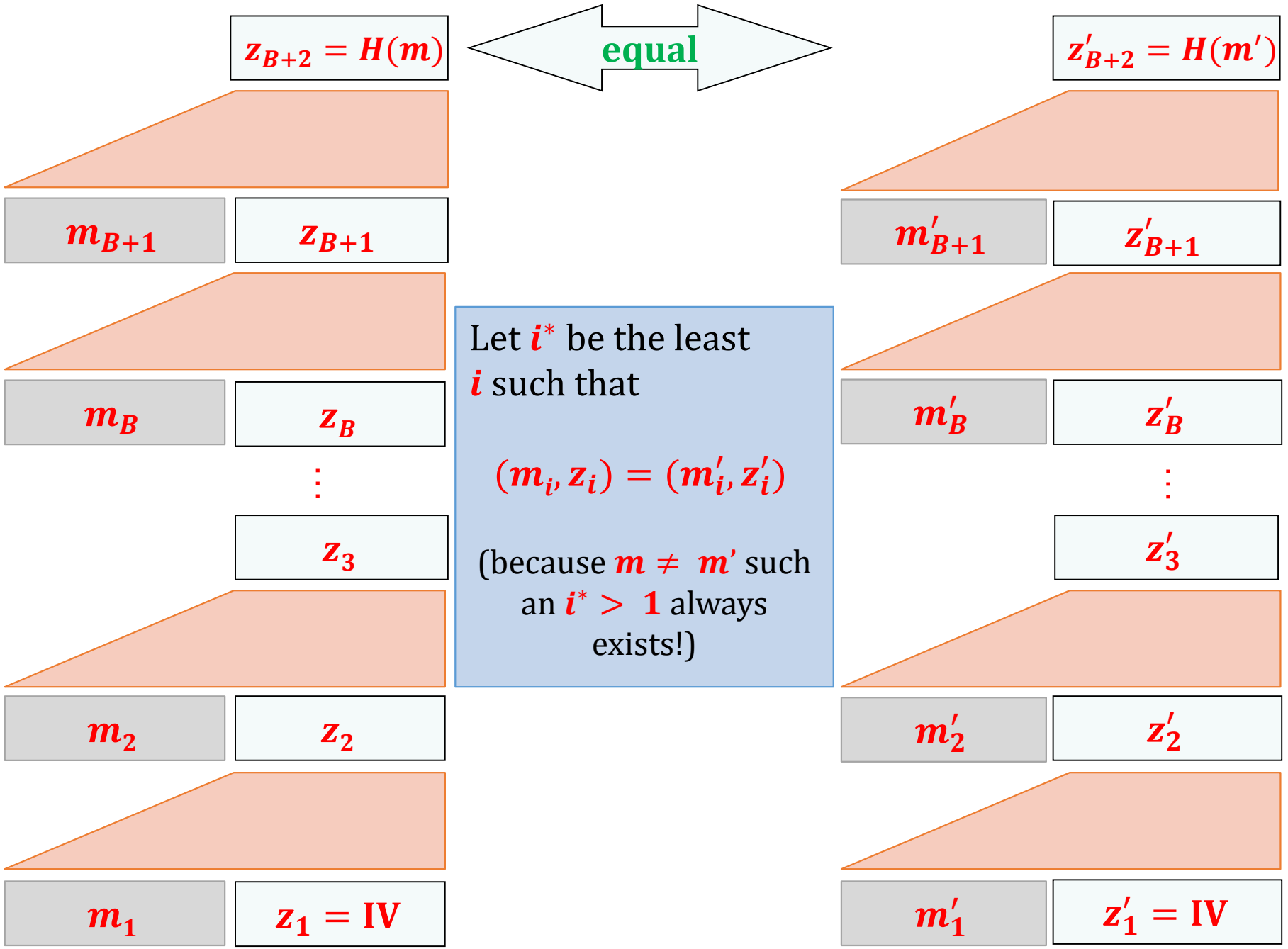


$|m| = |m'|$

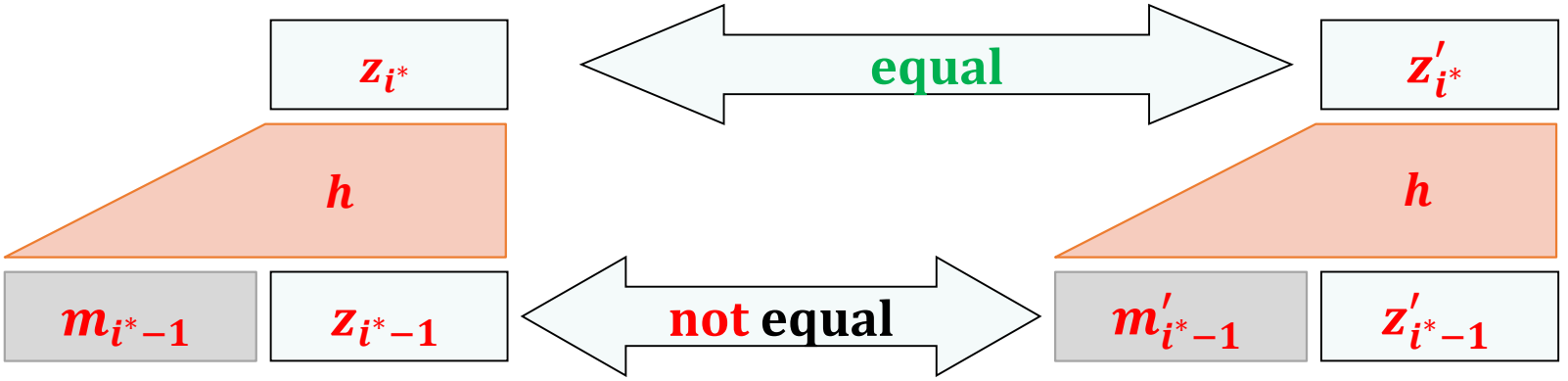
For m':



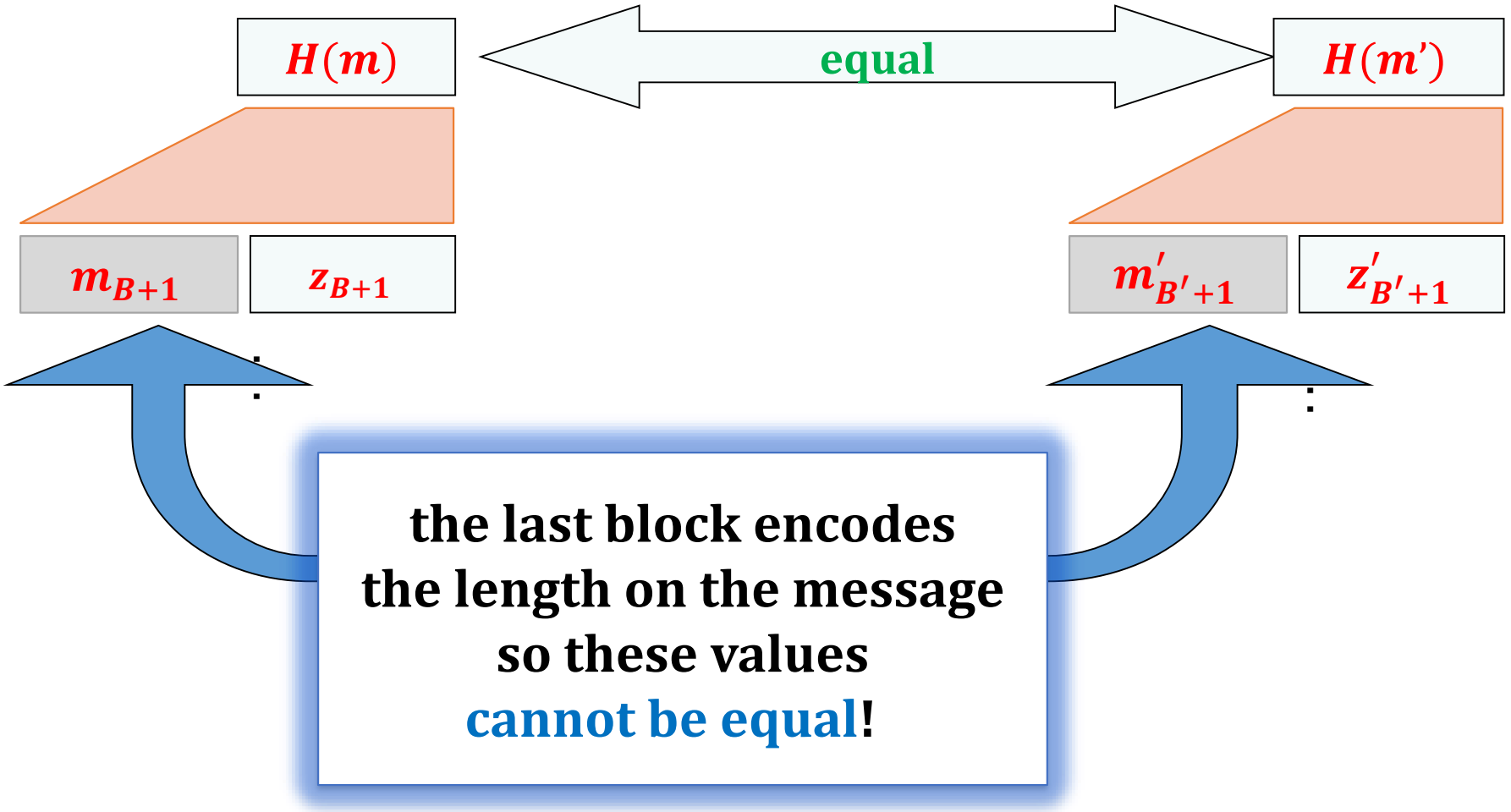

$z_{B+2} = H(m)$
equal
$z'_{B+2} = H(m')$
m_{B+1}
z_{B+1}
m'_{B+1}
z'_{B+1}
m_B
z_B
m'_B
z'_B
z_3
z'_3
m_2
z_2
not equal
m'_2
z'_2
m_1
$z_1 = \mathrm{IV}$
m'_1
$z'_1 = \mathrm{IV}$


$z_{B+2} = H(m)$
equal
$z'_{B+2} = H(m')$
m_{B+1}
z_{B+1}
m'_{B+1}
z'_{B+1}
m_B
z_B
m'_B
z'_B
z_3
z'_3
m_2
z_2
m'_2
z'_2
m_1
$z_1 = \mathrm{IV}$
m'_1
$z'_1 = \mathrm{IV}$
Let i^* be the least i such that
$(m_i, z_i) = (m'_i, z'_i)$
(because $m \neq m'$ such an $i^* > 1$ always exists!)

So, we have found a collision!



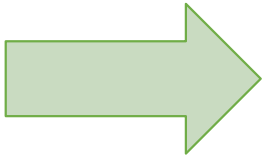
Case 2: $|m| \neq |m'|$



So, again we have found a collision!

Plan

1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction



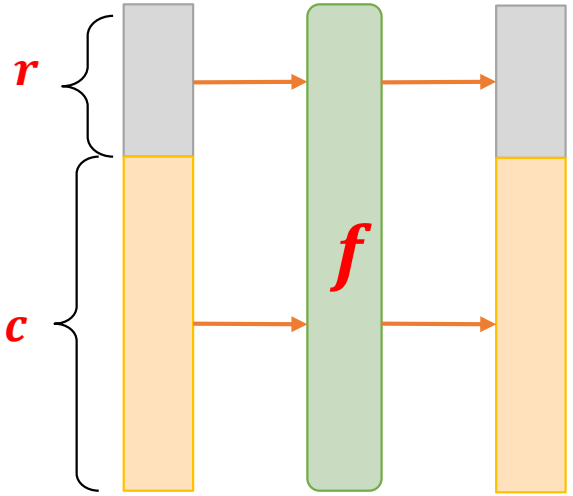
Sponge construction (used in Keccak)

main parameters:

c – "capacity"

r – "rate"

$b := c + r$ – "state width"



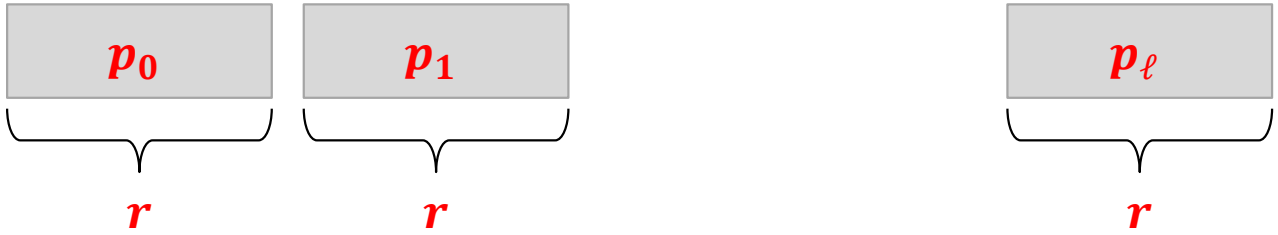
main ingredient:

a function $f: \{0,1\}^{r+c} \to \{0,1\}^{r+c}$

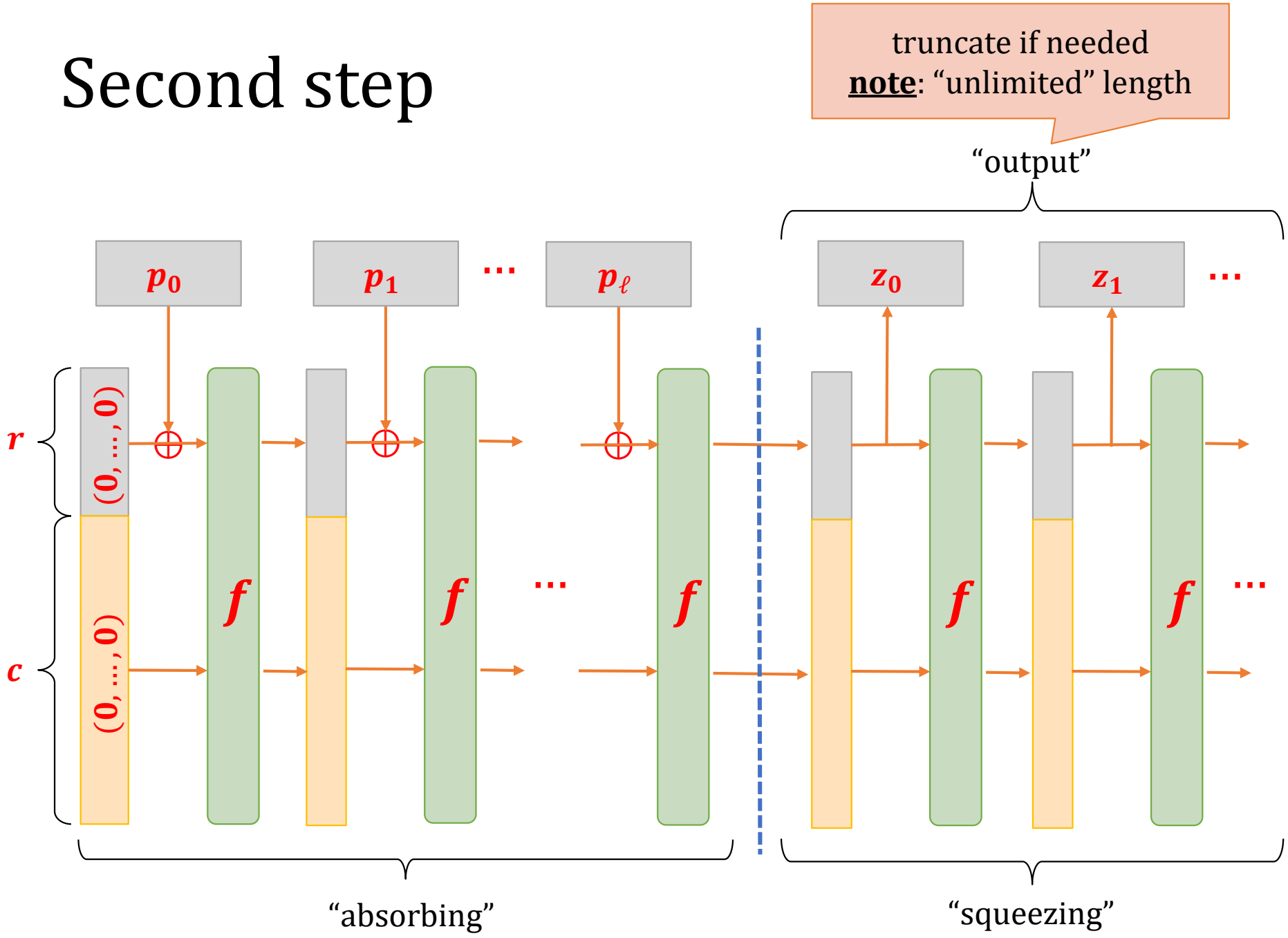
first step: input processing:

pad if needed with 10^*1

divide m as:



Second step


truncate if needed
note: “unlimited” length
“output”
p_0
p_1
p_ℓ
z_0
z_1
r
c
$(0,\dots,0)$
$(0,\dots,0)$
f
“absorbing”
“squeezing”