Random Oracle Combiners

Merkle-Damgärd Style

Yevgeniy Dodis, Eli Goldin, Peter Hall

NYU

(Hash) Combiners

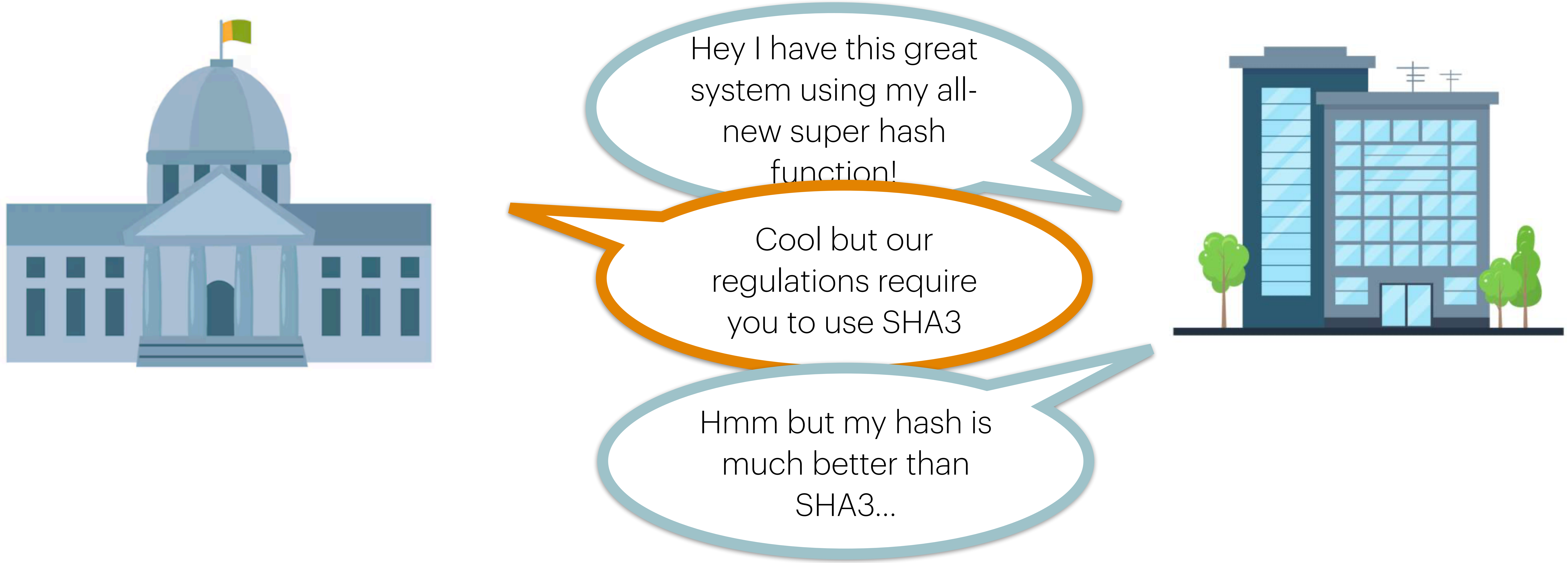
Hey I have this great system using my all-new super hash function!

(Hash) Combiners

Hey I have this great system using my all-new super hash function!

Cool but our regulations require you to use SHA3

(Hash) Combiners



(Hash) Combiners

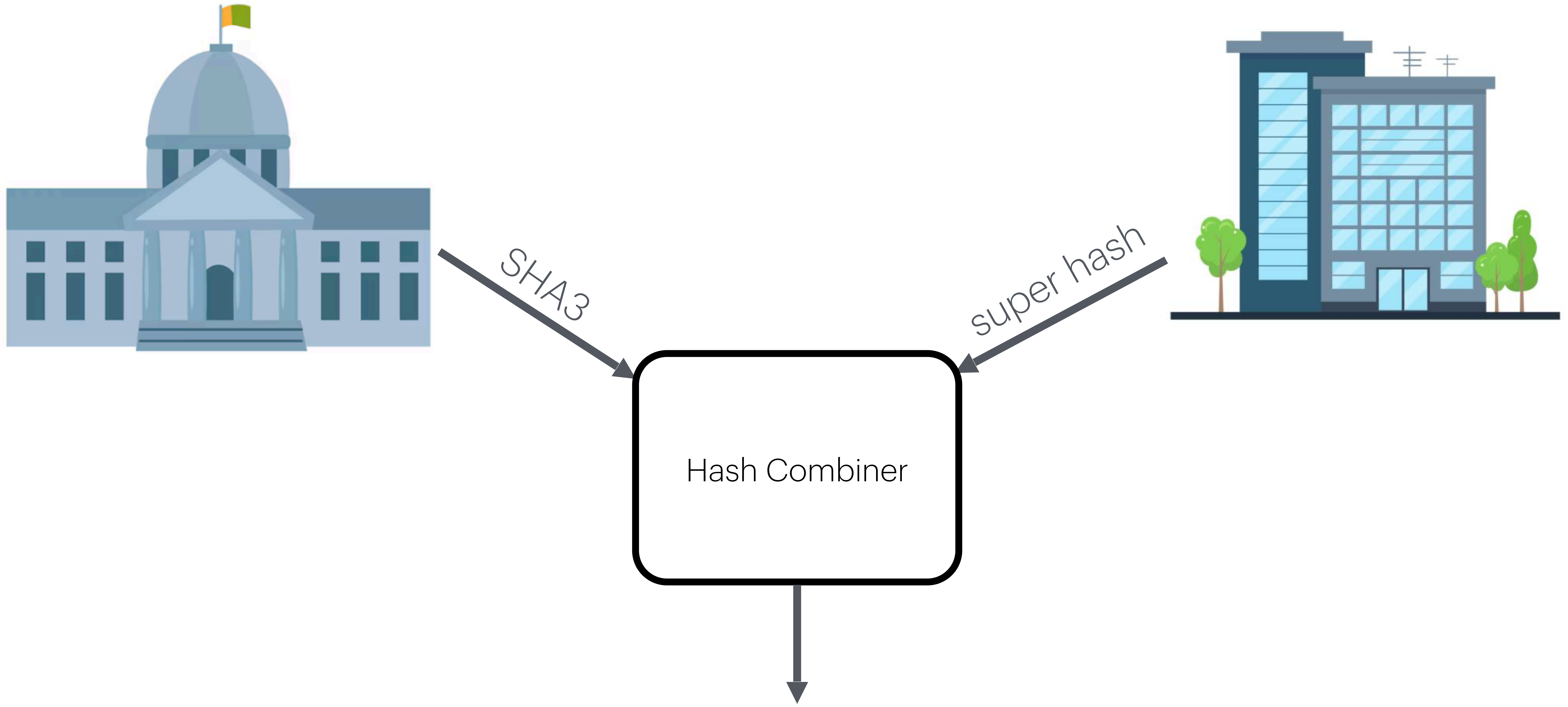
Hey I have this great system using my all-new super hash function!

Cool but our regulations require you to use SHA3

Hmm but my hash is much better than SHA3...

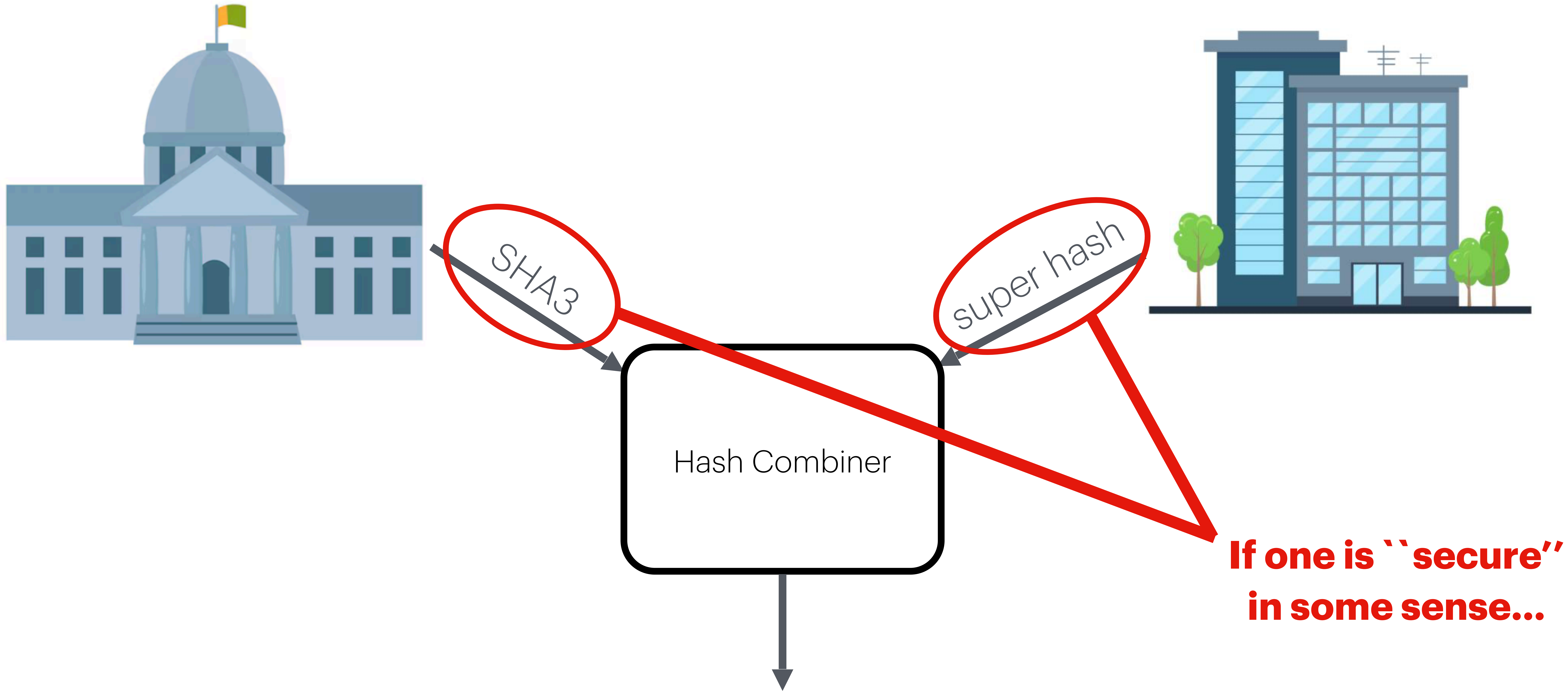
How can they both be happy??

(Hash) Combiners



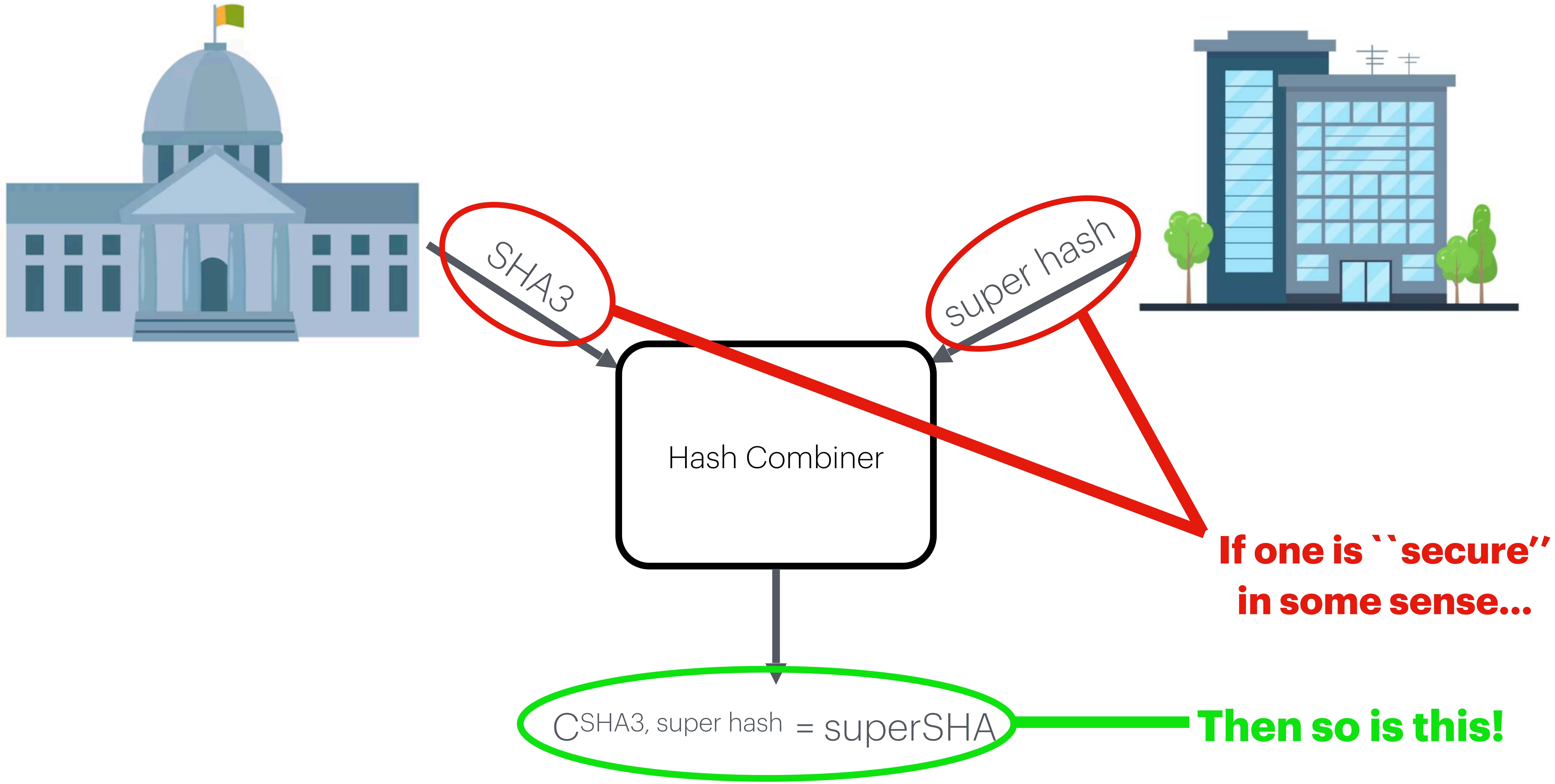
$C^{\text{SHA3, super hash}} = \text{superSHA}$

(Hash) Combiners



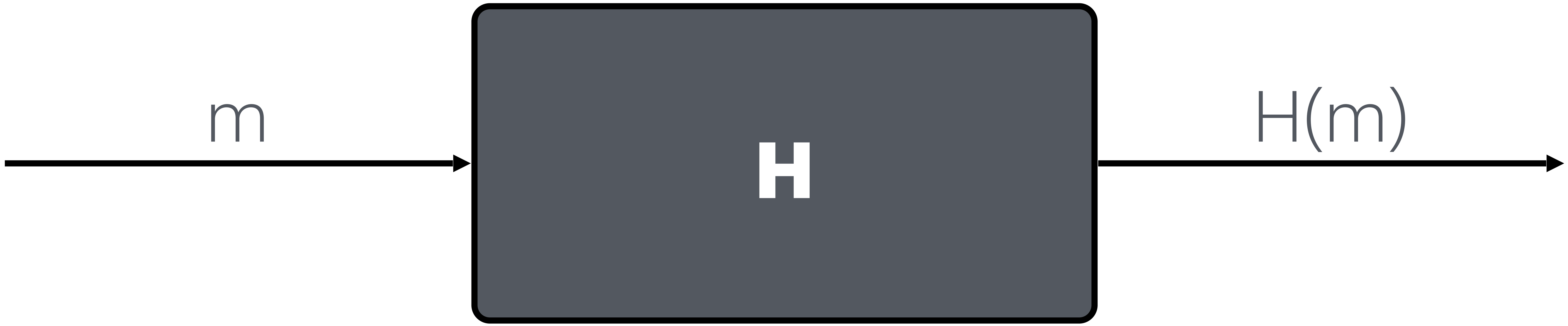
$$C^{\text{SHA3, super hash}} = \text{superSHA}$$

(Hash) Combiners



Hash Transforms

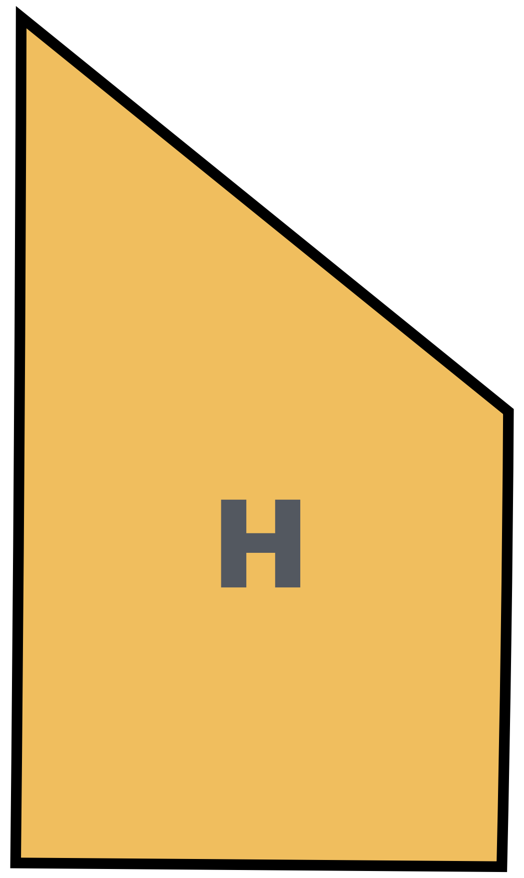
Monolithic Hash Function



Hash Transforms

The Merkle-Damgård Approach

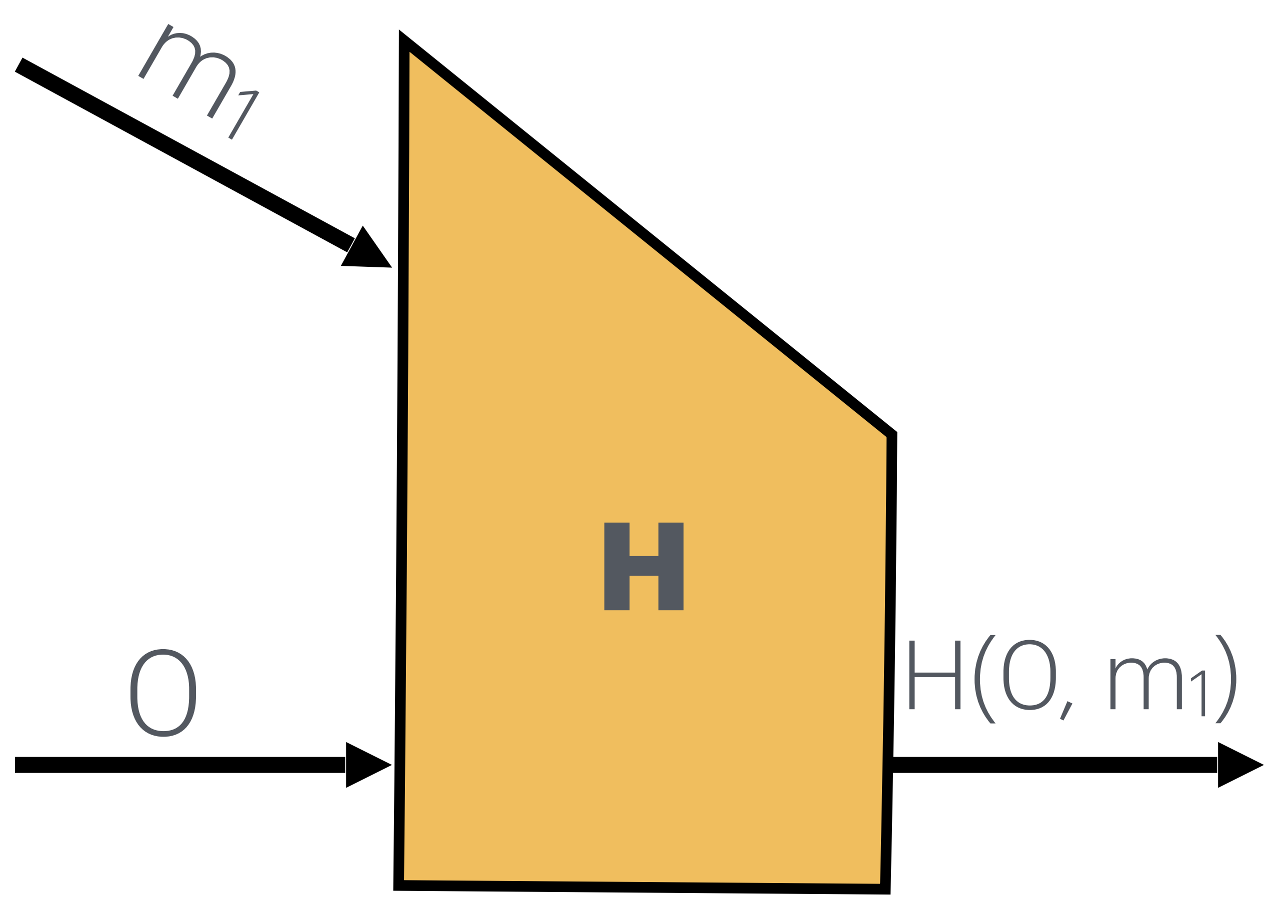
$(m = m_1 m_2 m_3 \ldots m_{n-1} m_n)$



Hash Transforms

The Merkle-Damgård Approach

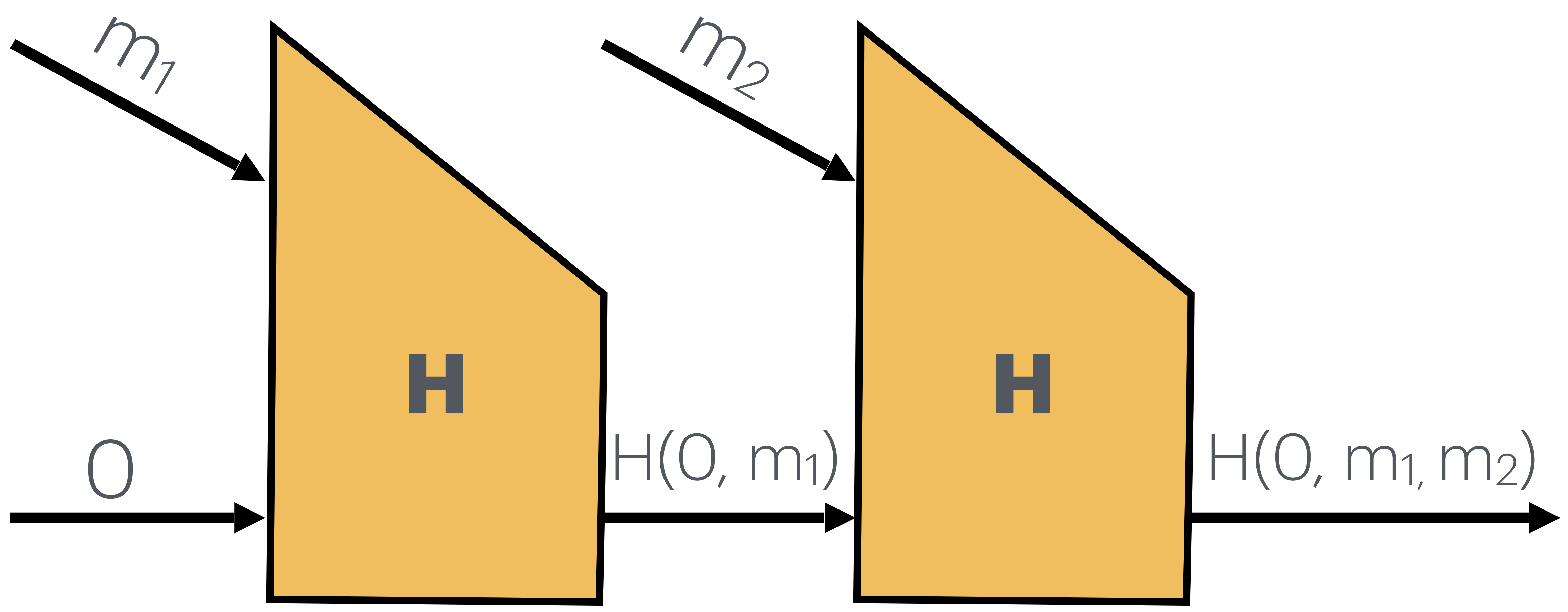
$(m = m_1 m_2 m_3 \ldots m_{n-1} m_n)$



Hash Transforms

The Merkle-Damgård Approach

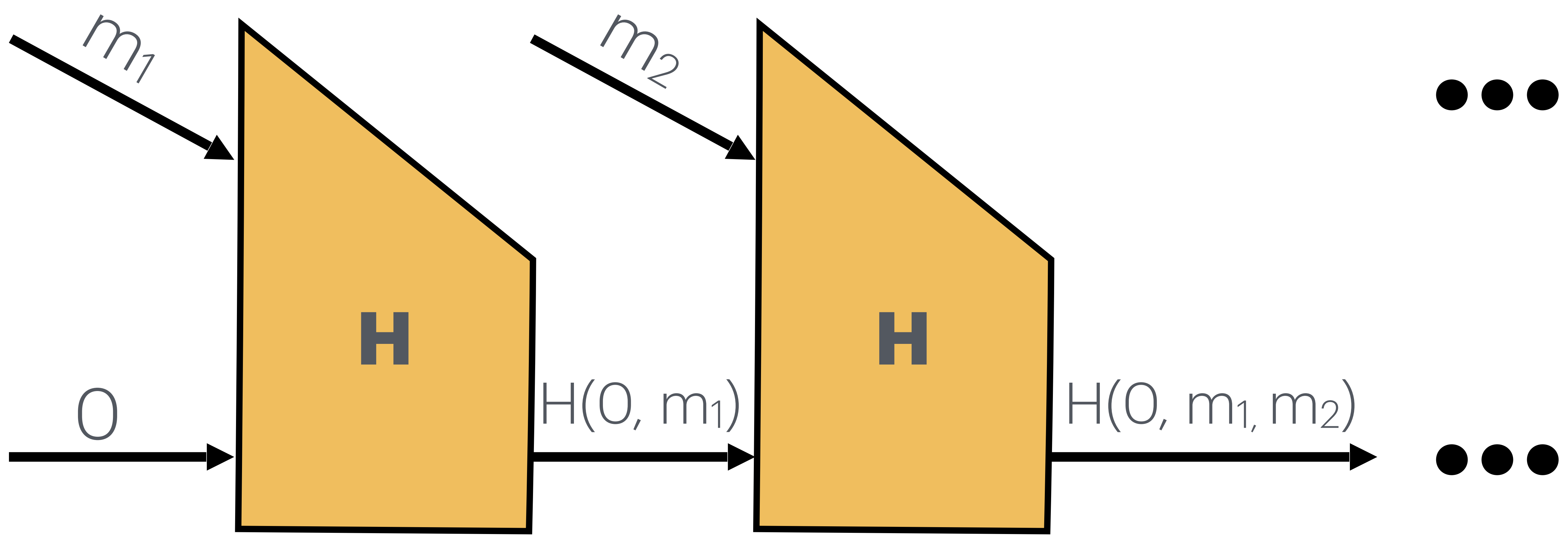
$(m = m_1 m_2 m_3 \ldots m_{n-1} m_n)$



Hash Transforms

The Merkle-Damgård Approach

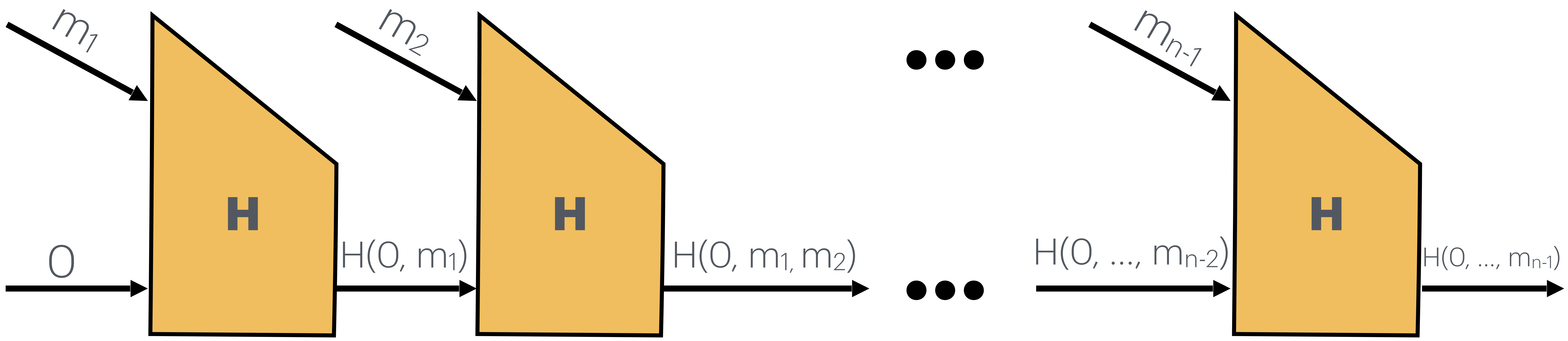
$(m = m_1 m_2 m_3 \ldots m_{n-1} m_n)$



Hash Transforms

The Merkle-Damgård Approach

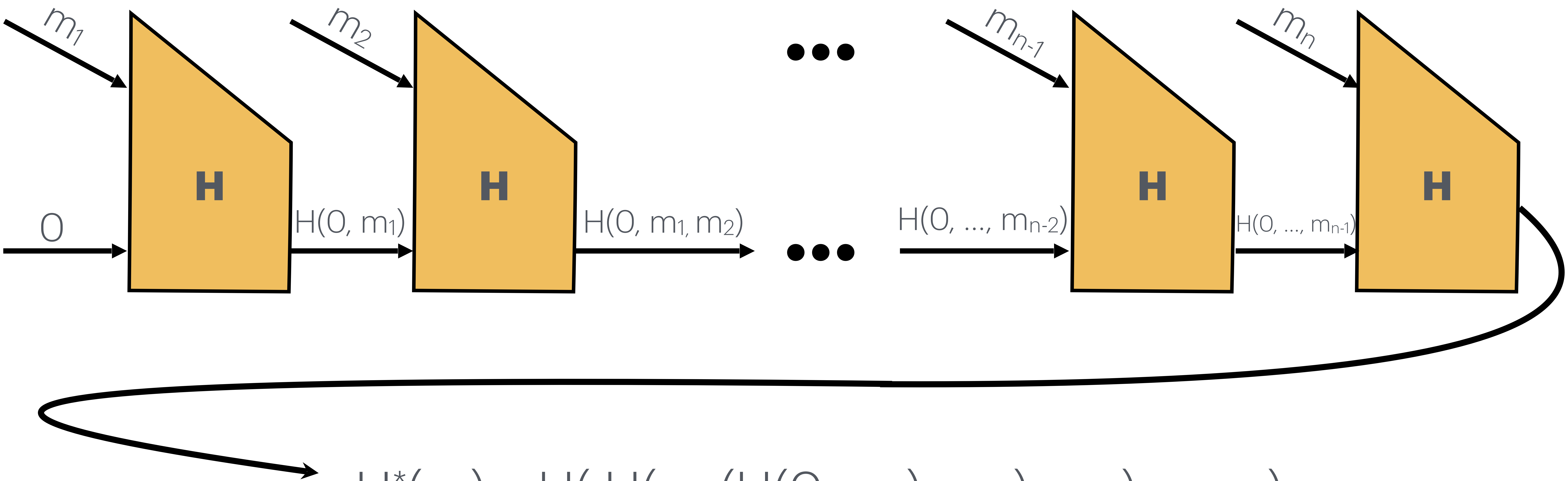
$(m = m_1 m_2 m_3 \ldots m_{n-1} m_n)$



Hash Transforms

The Merkle-Damgård Approach

$(m = m_1 m_2 m_3 \ldots m_{n-1} m_n)$

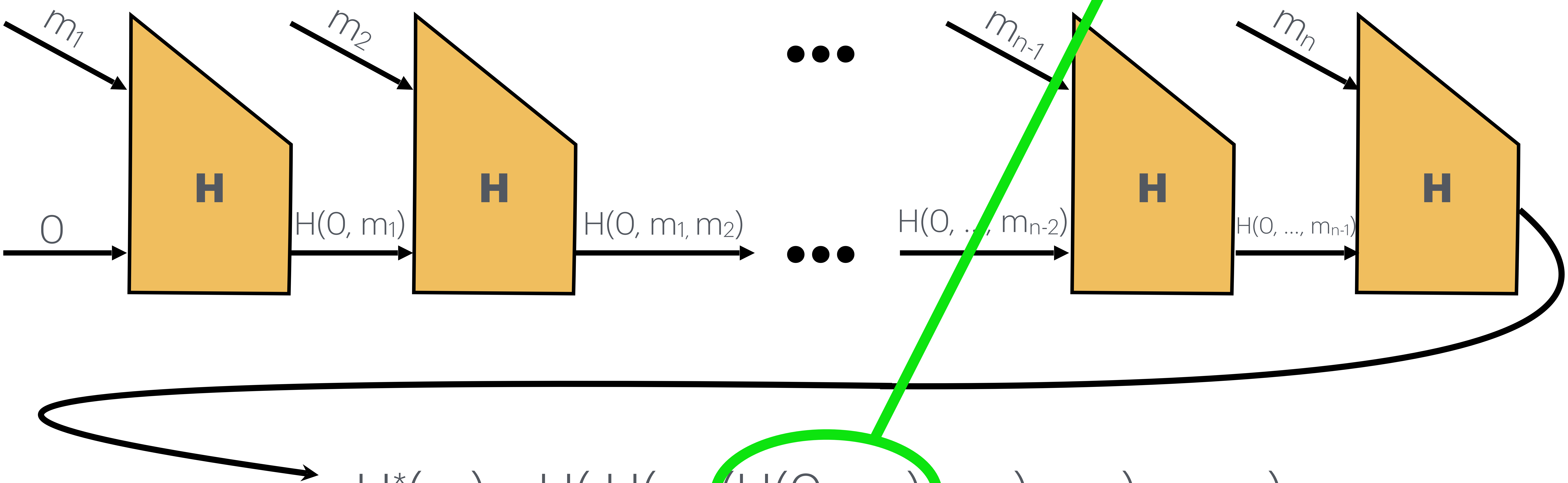


$H^*(m) = H(\,H(\ldots(H(0, m_1), m_2), m_3) \ldots m_n)$

Hash Transforms

The Merkle-Damgård Approach

Advantage: underlying hash H can be much less compressing than overall hash compression!



$$H^*(m) = H(\, H(\, \ldots (H(0, m_1), m_2), m_3) \ldots m_n)$$

Background

Concatenation Barrier

- Collision Resistance barrier

$$C^{h,\, g}(m) = h(m) \,||\, g(m)$$

- Similar for one-wayness, other constructions for pseudorandomness, MAC, etc.

Concatenation Barrier

- Collision Resistance barrier

$$C^{h,\,g}(m) = h(m) \,||\, g(m)$$

- Similar for one-wayness, other constructions for pseudorandomness, MAC, etc.

- [Pietrzak07, 08]: Length doubling is *optimal* for collision-resistance

Concatenation Barrier

- Collision Resistance barrier

$$C^{h,\,g}(m) = h(m) \,||\, g(m)$$

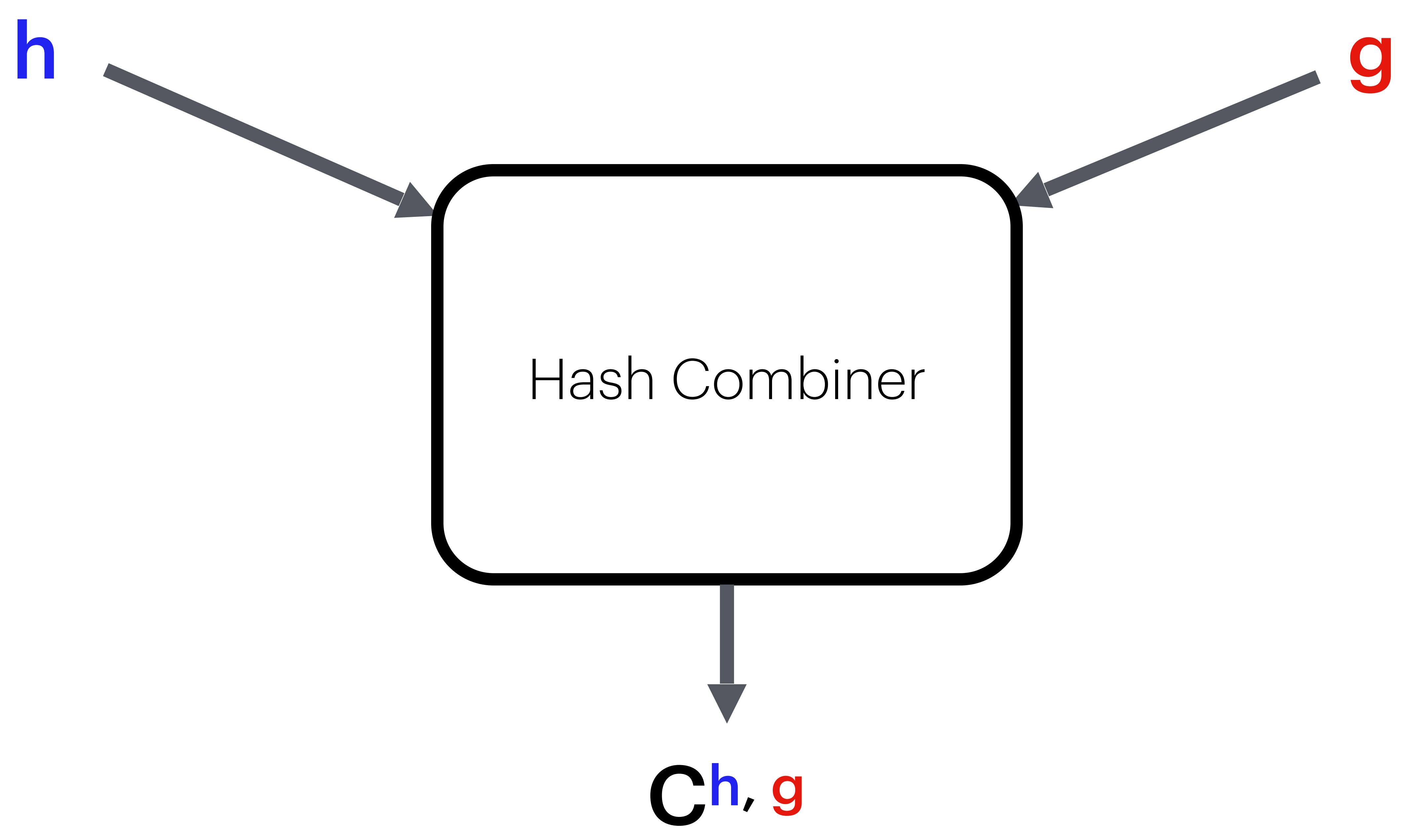
- Similar for one-wayness, other constructions for pseudorandomness, MAC, etc.

- [Pietrzak07, 08]: Length doubling is ***optimal*** for collision-resistance
- [Mittelbach 13]: Cryptophia’s short combiners, way around this assuming random oracles

Random Oracle Combiners

[*D*F*GH*P, CRYPTO23]

- Introduced notion of Random Oracle (RO) Combiners
- *Idea*: If one of the hashes is RO, then the combiner C is ***essentially*** BIG-RO



Indifferentiability Framework

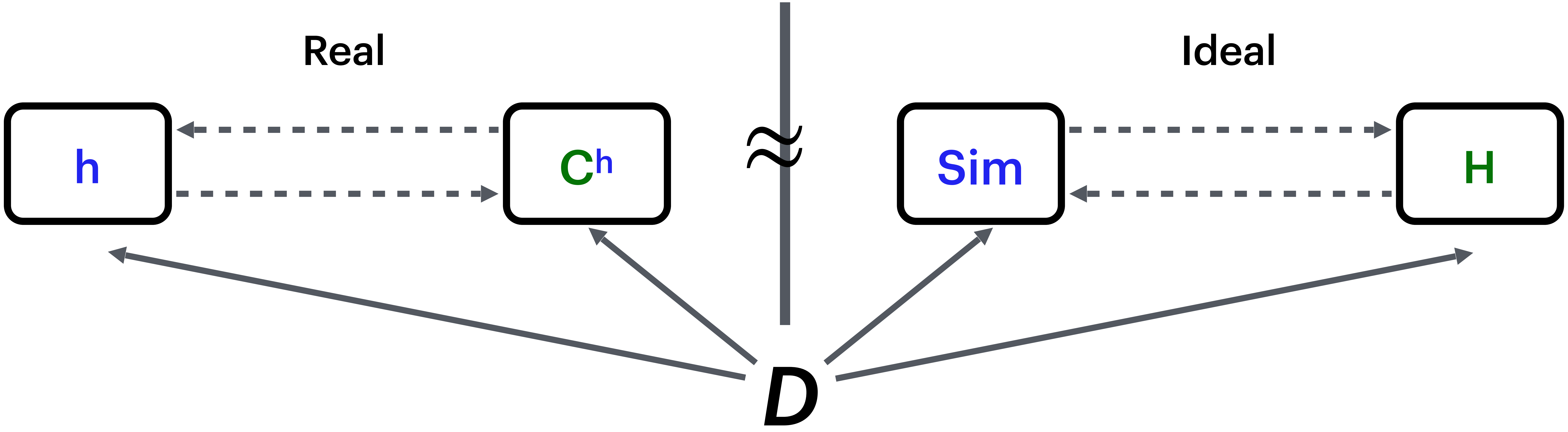
[MRH04, CDMP05]

- Real-ideal world framework, where we want indistinguishability between
    - interacting with RO h and the combiner C
    - Interacting with a simulator Sim and RO H

Indifferentiability Framework

[MRH04, CDMP05]

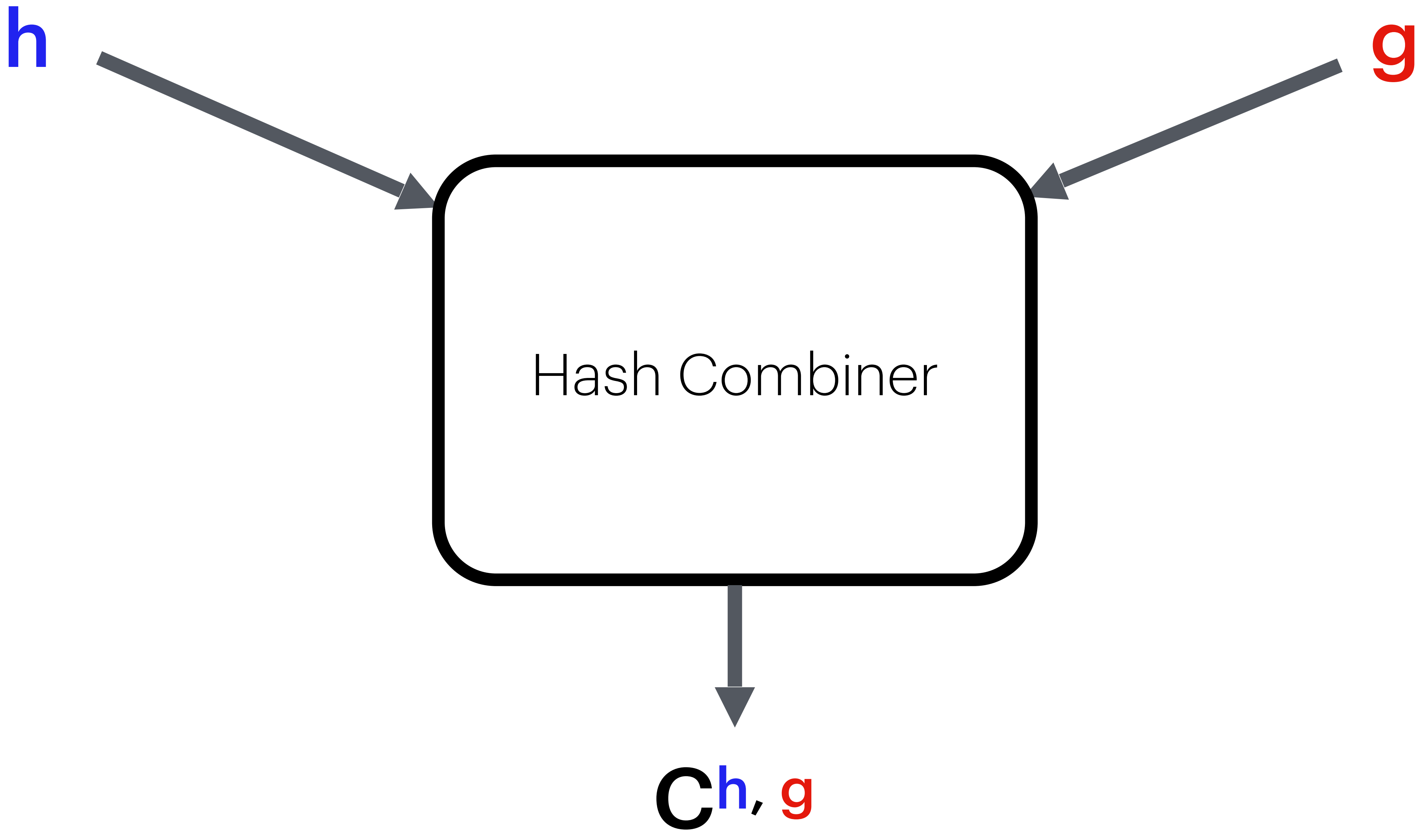
- Real-ideal world framework, where we want indistinguishability between
  - interacting with RO *h* and the combiner *C*
  - Interacting with a simulator Sim and RO H



Random Oracle Combiners

[*D*F*GH*P, CRYPTO23]

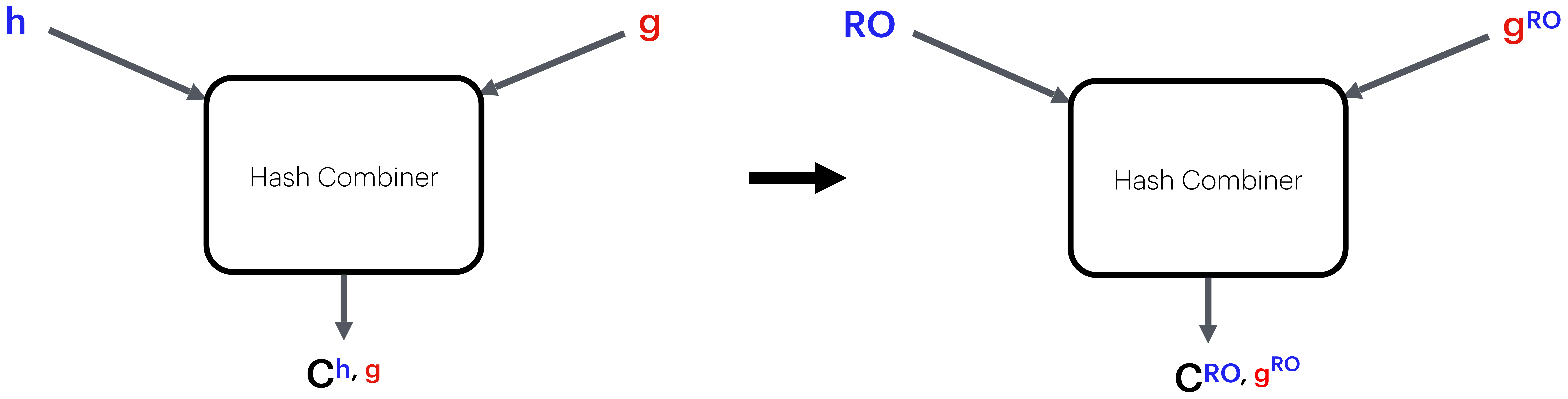
- Introduced notion of Random Oracle (RO) Combiners
- *Idea*: If one of the hashes is RO, then the combiner is ***essentially*** a RO



Random Oracle Combiners

[*D*F*GH*P, CRYPTO23]

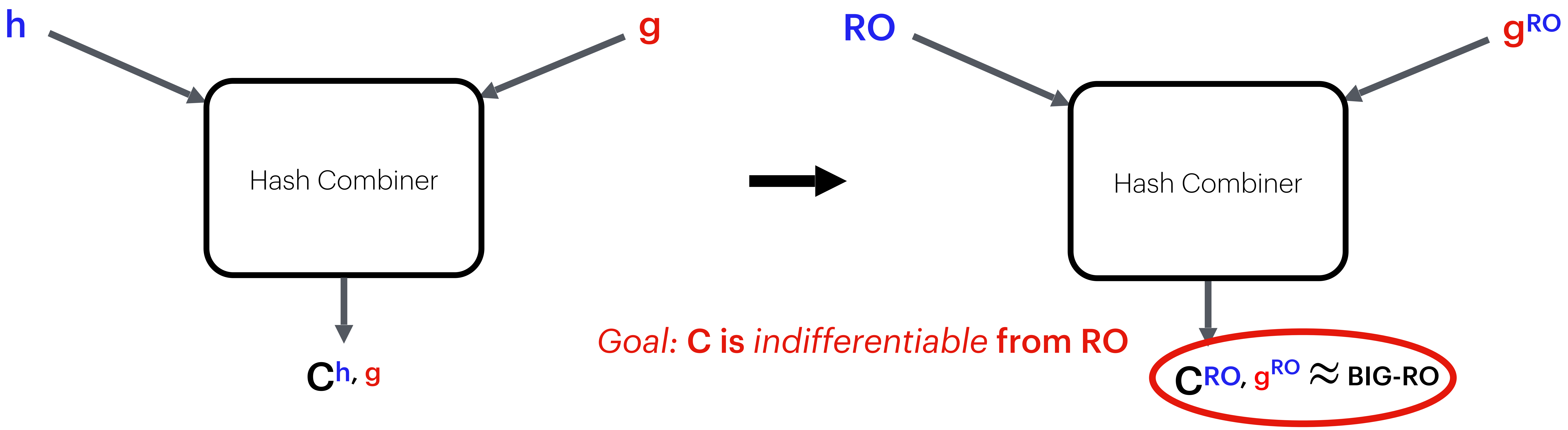
- Introduced notion of Random Oracle (RO) Combiners
- *Idea*: If one of the hashes is RO, then the combiner C is ***essentially*** BIG-RO



Random Oracle Combiners

[*D*F*GH*P, CRYPTO23]

- Introduced notion of Random Oracle (RO) Combiners
- *Idea*: If one of the hashes is RO, then the combiner C is ***essentially*** BIG-RO



Random Oracle Combiners

[*D*F*GH*P, CRYPTO23]

- Introduced notion of Random Oracle (RO) Combiners
- *Idea*: If one of the hashes is RO, then the combiner C is ***essentially*** BIG-RO

Random Oracle Combiners

[*D*F*GH*P, CRYPTO23]

- Introduced notion of Random Oracle (RO) Combiners
- *Idea*: If one of the hashes is RO, then the combiner C is ***essentially*** BIG-RO

Goal: **C is *indifferentiable* from RO**

Note: **C must be salted**

Hash Combiners on Long Inputs

[DFG+23] gave basically optimal construction for hashes with good compression

$$C^{h,g}_{Z_1,Z_2}(m) = h(m, Z_1) \oplus g(m, Z_2)$$

Downside: for applications, does not give evidence when h (and g) Merkle-Damgård

—> *despite* the fact that the above is indif. from RO *and* M-D is indif. from RO

—> composition of the two only works for single-stage games, which RO Comb. is not

—> in particular, g can depend arbitrarily on underlying compression of h

Hash Combiners on Long Inputs

[DFG+23] gave basically optimal construction for hashes with good compression

$$C^{h,g}_{Z_1,Z_2}(m) = h(m, Z_1) \oplus g(m, Z_2)$$

Desired goals for Merkle-Damgård combiner:

1. **Assumes only that one of *h* or *g* is a RO of mild compression**
2. **Only calls M-D transformation of h^*, g^* on inputs**
3. **Supports arbitrarily long input messages, but message output is still less than 2 times output of $h=h^*$.**

Our Results

Main Result

We construct the following RO Combiner:

$$C^{h,g}_{Z_1,Z_2}(m) = h^*(m, Z_1) \oplus g^*(m, Z_2)$$

Parameters (previous):

- **Compression of individual hash:** $n + \delta$
- **Compression of** ***overall*** **hashes h, g:** $n + \delta$
- **Length of salts Z_1, Z_2:** $|M| + \lambda$

Parameters (us):

- **Compression of individual hash:** $n + \delta$
- **Compression of** ***overall*** **hashes h^*, g^*:** $\delta \cdot \ell$
- **Length of salts Z_1, Z_2:** $|M| + \lambda$

What Fails with Merkle-Damgård?

- In monolithic, proof critically used the fact that Z is longer than messages to show that $g(m', Z_2)$ cannot compute $h(m, Z_1)$ for any messages *m*.
  - This way, Sim can answer $h(m', Z_1) = H(m') \oplus g(m', Z_2)$ without any "recursion"
- Unfortunately for M-D, it is easy to construct (m, m′) and define g^h such that $(g^h)^*(m', Z_2)$ can compute completions of $h^*(m, Z_1)$.

What Fails with Merkle-Damgård?

- Unfortunately for M-D, it is easy to construct (m, m′) and define g^h such that $(g^h)^*(m', Z_2)$ can compute completions of $h^*(m, Z_1)$.

What Fails with Merkle-Damgård?

- Unfortunately for M-D, it is easy to construct (m, m′) and define g^h such that $(g^h)^*(m', Z_2)$ can compute completions of $h^*(m, Z_1)$.

- Consider $h : \{0,1\}^{2n} \to \{0,1\}^n$. Then, for a message m, consider:

$$m' := \left(h^*\left(m, Z_1^{(1)}, Z_1^{(2)}, \ldots, Z_1^{(k-1)}\right), Z_1^{(k)}\right)$$

Where k is the last block of Z_1 and $Z_1^{(i)}$ is the i-th block of Z_1.

What Fails with Merkle-Damgård?

- Unfortunately for M-D, it is easy to construct (m, m') and define g^h such that $(g^h)^*(m', Z_2)$ can compute completions of $h^*(m, Z_1)$.

- Consider $h : \{0,1\}^{2n} \to \{0,1\}^n$. Then, for a message m, consider:

$$m' := \left(h^*\left(m, Z_1^{(1)}, Z_1^{(2)}, \ldots, Z_1^{(k-1)}\right), Z_1^{(k)}\right)$$

Where k is the last block of Z_1 and $Z_1^{(i)}$ is the i-th block of Z_1. Then, easy to define g^h so that:

$$(g^h)^*(m', Z_2) = h(m') = h\left(\left(h^*\left(m, Z_1^{(1)}, Z_1^{(2)}, \ldots, Z_1^{(k-1)}\right), Z_1^{(k)}\right)\right) = h^*(m, Z_1)$$

What Fails with Merkle-Damgård?

- Unfortunately for M-D, it is easy to construct (m, m′) and define g^h such that $(g^h)^*(m', Z_2)$ can compute completions of $h^*(m, Z_1)$.

- Consider $h : \{0,1\}^{2n} \to \{0,1\}^n$. Then, for a message m, consider:

$$m' := \left(h^*\left(m, Z_1^{(1)}, Z_1^{(2)}, \ldots, Z_1^{(k-1)}\right), Z_1^{(k)} \right)$$

Where k is the last block of Z_1 and $Z_1^{(i)}$ is the i-th block of Z_1. Then, easy to define g^h so that:

$$(g^h)^*(m', Z_2) \;= h(m') = h\left(\left(h^*\left(m, Z_1^{(1)}, Z_1^{(2)}, \ldots, Z_1^{(k-1)}\right), Z_1^{(k)} \right) \right) = h^*(m, Z_1)$$

Lesson: intuitions for monolithic may not carry over to Merkle-Damgård case

Proof

Proof

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathsf{Sim}^*(m, Z_1) \oplus (g^{\mathsf{Sim}})^*(m, Z_2) = H(m) \iff \mathsf{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$$

- Sim must answer all queries $(a, b) \rightarrow c$ at random while also satisfying above "global" question
- Challenges:

  1. Need to spot all such constraints
  2. Need the last value of *h* satisfying the equation to be "free"
  3. Runtime of simulator must be polynomial given distinguisher *D* is query bounded

Proof

Need to build Sim such that $\text{Sim}^*(m, Z_1) \oplus (g^{\text{Sim}})^*(m, Z_2) = H(m)$

$$\mathsf{Sim}^*(m, Z_1) \oplus (g^{\mathsf{Sim}})^*(m, Z_2) = H(m) \iff \mathsf{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$$

- Sim must answer all queries $(a, b) \rightarrow c$ at random while also satisfying above "global" question
- Challenges:

   1. Need to spot all such constraints
   2. Need the last value of *h* satisfying the equation to be "free"
   3. Runtime of simulator must be polynomial given distinguisher *D* is query bounded

In essence the trickiest, will imply 1+2 if done right…

Proof

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m) \iff \mathrm{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathrm{Sim}})^*(m, Z_2)$$

Sim:

Query	1st input a	2nd input b	Output c
$(0, m_1)$	0	m_1	r_1
(r_1, m_2)	r_1	m_2	r_2
$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	

Proof

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m) \iff \mathrm{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathrm{Sim}})^*(m, Z_2)$$

Sim:

Query	1st input a	2nd input b	Output c
$(0, m_1)$	0	m_1	r_1
(r_1, m_2)	r_1	m_2	r_2
$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	

Got a message of length *l+k*…
have to be consistent here

Proof

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathsf{Sim}^*(m, Z_1) \oplus (g^{\mathsf{Sim}})^*(m, Z_2) = H(m) \iff \mathsf{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$$

Sim:

Query	1st input a	2nd input b	Output c
$(0, m_1)$	0	m_1	r_1
(r_1, m_2)	r_1	m_2	r_2
$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	$H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$

Proof

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathsf{Sim}^*(m, Z_1) \oplus (g^{\mathsf{Sim}})^*(m, Z_2) = H(m) \iff \mathsf{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$$

Sim:

Query	1st input a	2nd input b	Output c
$(0, m_1)$	0	m_1	r_1
(r_1, m_2)	r_1	m_2	r_2
$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	$H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$

Okay, what did g^* call here?
I need to fill in those as well

The worry

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m) \iff \mathrm{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathrm{Sim}})^*(m, Z_2)$$

Sim:

Query	1st input a	2nd input b	Output c
$(0, m_1)$	0	m_1	r_1
(r_1, m_2)	r_1	m_2	r_2
$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	$H(m) \oplus (g^{\mathrm{Sim}})^*(m, Z_2)$

g-Query	1st input a	2nd input b	Output c
(a_1, b_1)	a_1	b_1	r'_1
(a_2, b_2)	a_2	b_2	r'_3
...	...	...	...

The worry

Need to build Sim such that $\text{Sim}^*(m, Z_1) \oplus (g^{\text{Sim}})^*(m, Z_2) = H(m)$

$$\text{Sim}^*(m, Z_1) \oplus (g^{\text{Sim}})^*(m, Z_2) = H(m) \iff \text{Sim}^*(m, Z_1) = H(m) \oplus (g^{\text{Sim}})^*(m, Z_2)$$

Sim:

Query	1st input a	2nd input b	Output c	g-Query	1st input a	2nd input b	Output c
$(0, m_1)$	0	m_1	r_1	(a_1, b_1)	a_1	b_1	r'_1
(r_1, m_2)	r_1	m_2	r_2	...	...	...	...
$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3				
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	$H(m) \oplus (g^{\text{Sim}})^*(m, Z_2)$				

Hm...this also calls
something of length *l+k*...
have to recurse again

The worry

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathsf{Sim}^*(m, Z_1) \oplus (g^{\mathsf{Sim}})^*(m, Z_2) = H(m) \iff \mathsf{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$$

Sim:

Query	1st input a	2nd input b	Output c	g-Query	1st input a	2nd input b	Output c
$(0, m_1)$	0	m_1	r_1	(a_1, b_1)	a_1	b_1	r'_1
(r_1, m_2)	r_1	m_2	r_2	…	…	…	…
$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3				
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	$H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$				

Hm...this also calls something of length *l+k*... have to recurse again

The worry

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathsf{Sim}^*(m, Z_1) \oplus (g^{\mathsf{Sim}})^*(m, Z_2) = H(m) \iff \mathsf{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$$

Sim:

Query	1st input a	2nd input b	Output c	g-Query	1st input a	2nd input b	Output c
$(0, m_1)$	0	m_1	r_1	(a_1, b_1)	a_1	b_1	r'_1
(r_1, m_2)	r_1	m_2	r_2	…	…	…	…
$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3				
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	$H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$				

Hm…this also calls something of length *l+k*… have to recurse again

The worry

Need to build Sim such that $\mathrm{Sim}^*(m, Z_1) \oplus (g^{\mathrm{Sim}})^*(m, Z_2) = H(m)$

$$\mathsf{Sim}^*(m, Z_1) \oplus (g^{\mathsf{Sim}})^*(m, Z_2) = H(m) \iff \mathsf{Sim}^*(m, Z_1) = H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$$

Sim:

Need to argue no infinite (or just overwhelming) recursion!

$(r_2, Z_{1,1})$	r_2	$Z_{1,1}$	r_3
$(r_3, Z_{1,2})$	r_3	$Z_{1,2}$	$H(m) \oplus (g^{\mathsf{Sim}})^*(m, Z_2)$

Hm…this also calls
something of length *l+k*…
have to recurse again

Termination

- As said before, cannot argue directly that g never queries some (m', Z_1)
  - We will settle for weaker claim:

Termination

- As said before, cannot argue directly that g never queries some (m', Z_1)
  - We instead settle for a sufficient weaker claim:

Claim:

For *D* to find a message m such that $(g^h)^*(m, Z_2)$ queries some $h^*(m', Z_1)$, *D* must have queried $h^*(m')$ itself.

Termination

- As said before, cannot argue directly that g never queries some (m', Z_1)
  - We instead settle for a sufficient weaker claim:

Claim:

For you to find a message m such that $(g^h)^*(m, Z_2)$ queries some $h^*(m', Z_1)$, then you must have queried $h^*(m')$ yourself.

- If this is true, then #(queries made by recursive g calls) ≤ #(queries D made) — bounded!

Termination

Key

= direct Sim queries

= current recursive Sim queries

= permanent recursive Sim queries

Termination

Note: only queries of length up to ***l*** followed by some Z_1 blocks “matter"

Key

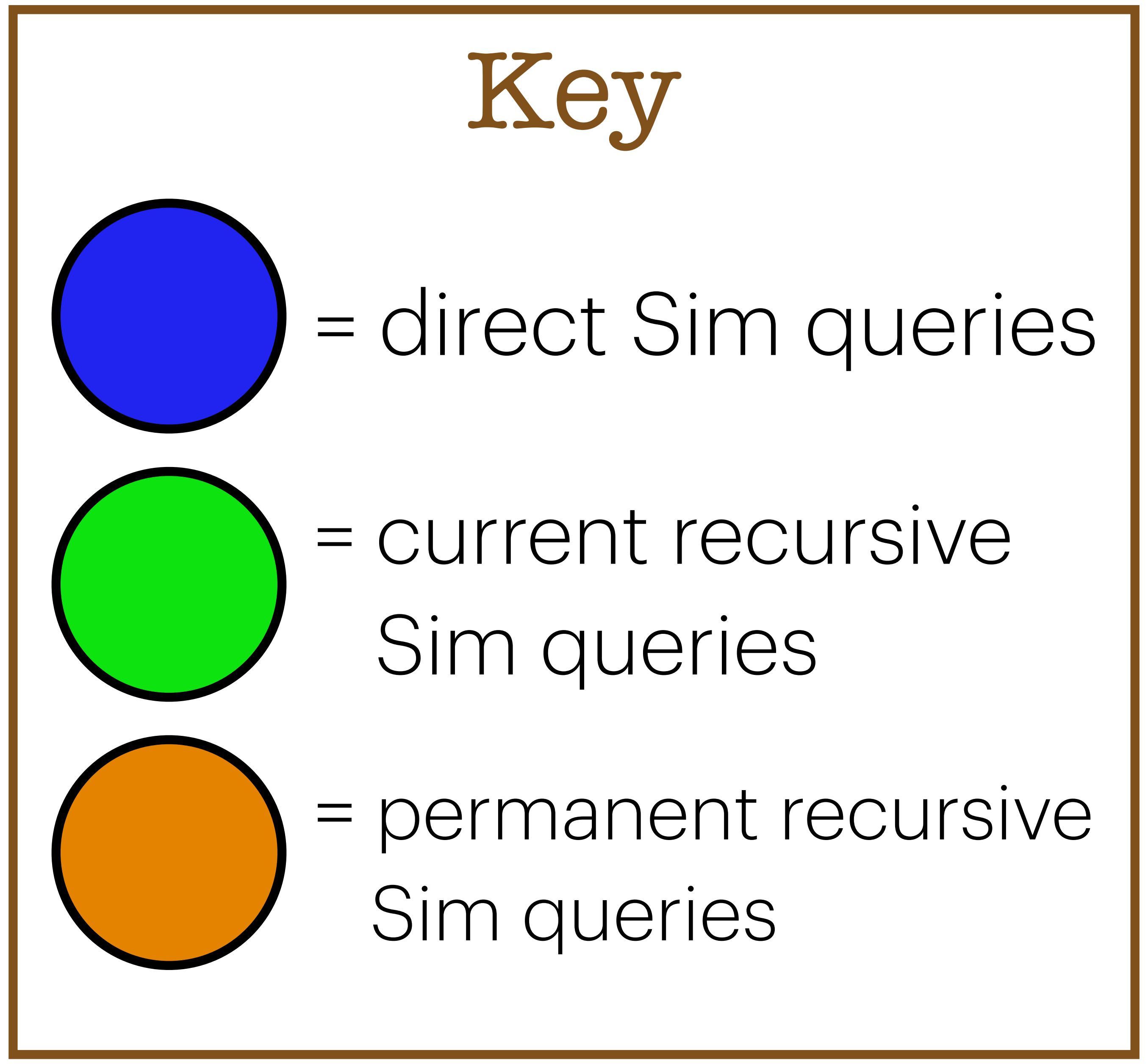
= direct Sim queries

= current recursive Sim queries

= permanent recursive Sim queries

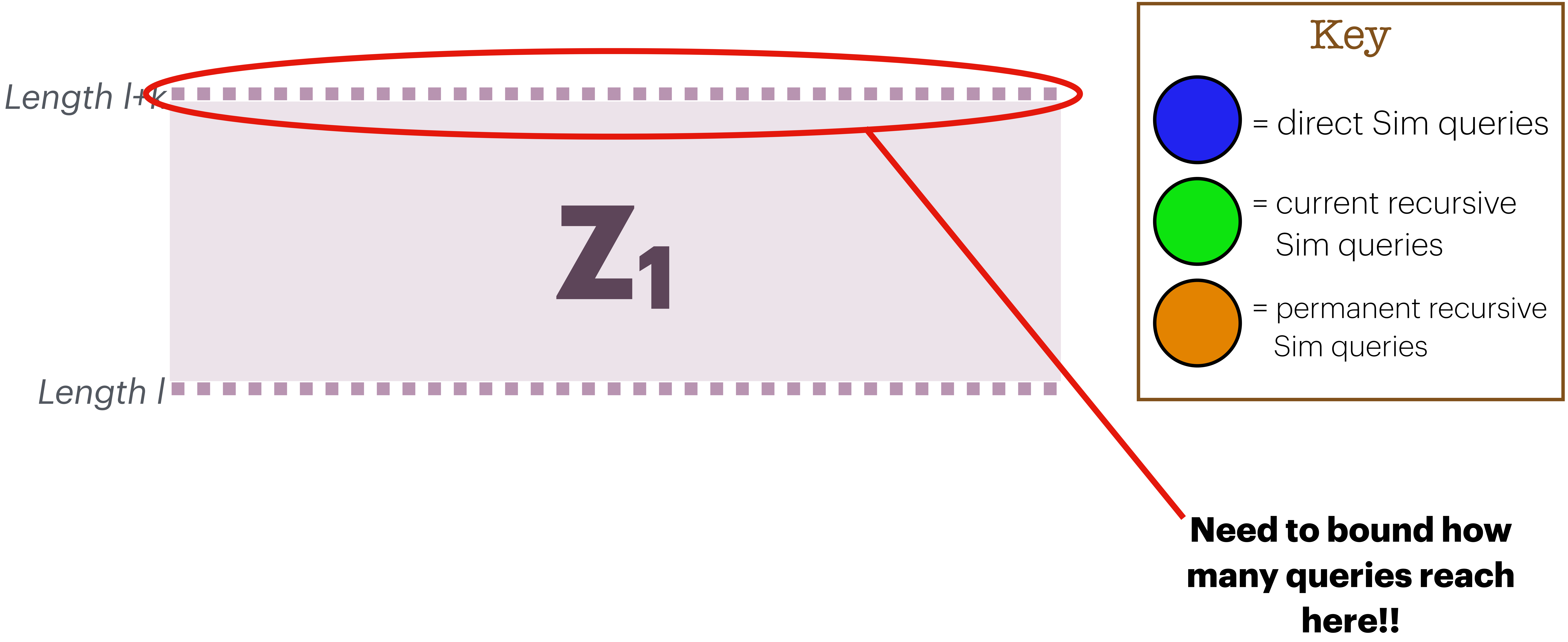
Termination

Note: only queries of length up to ***l*** followed by some Z_1 blocks "matter"



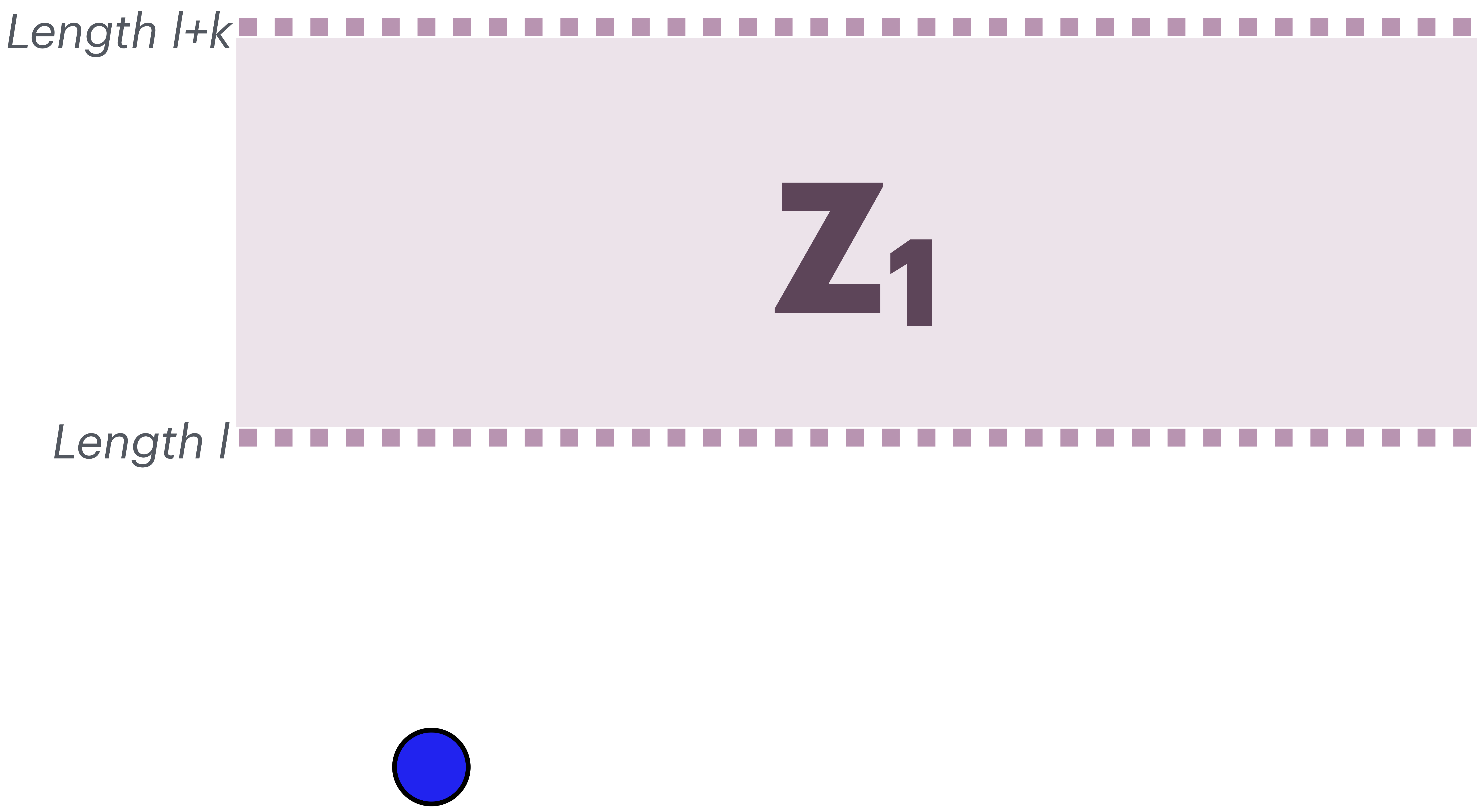
Termination

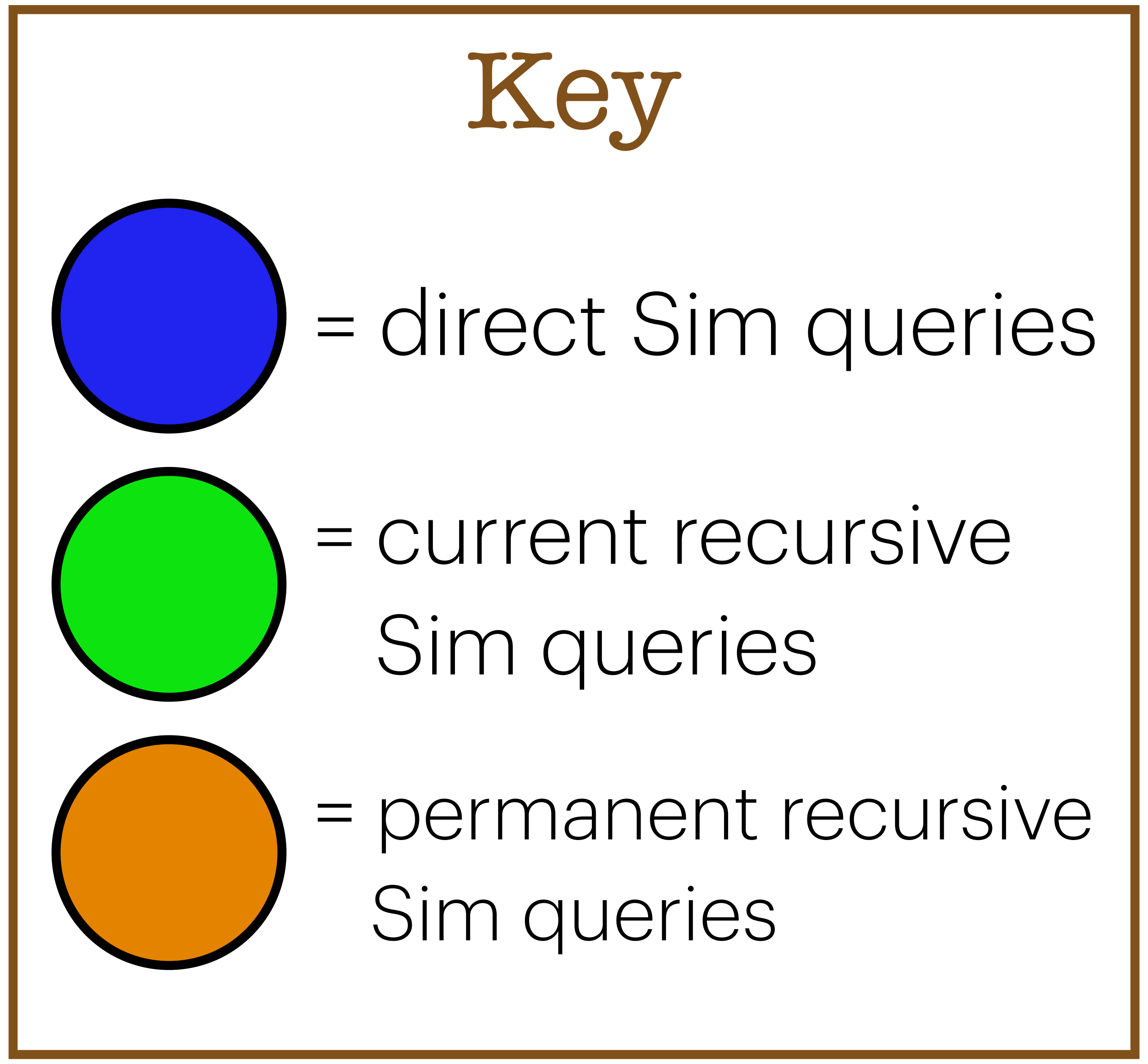
Note: only queries of length up to ***l*** followed by some Z_1 blocks "matter"



Termination

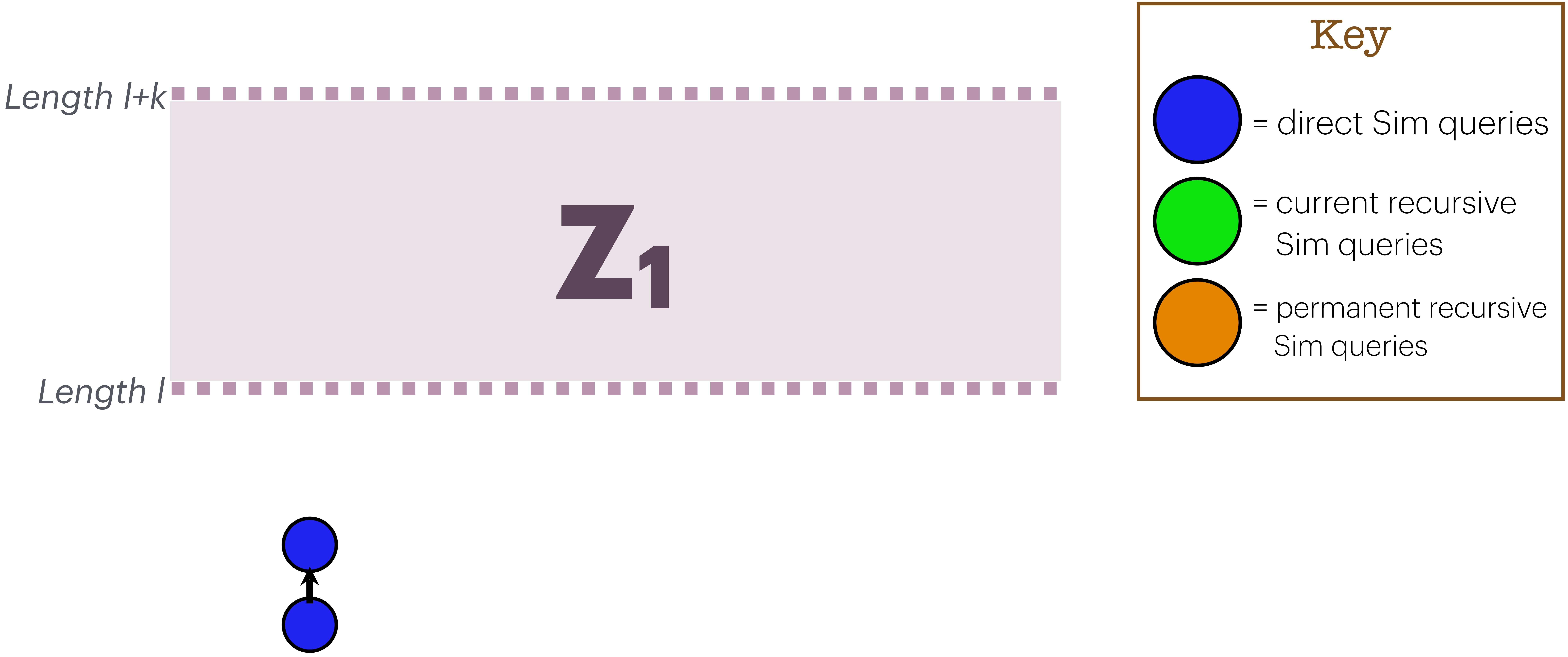
Note: only queries of length up to l followed by some Z_1 blocks "matter"





Termination

Note: only queries of length up to *l* followed by some Z_1 blocks "matter"

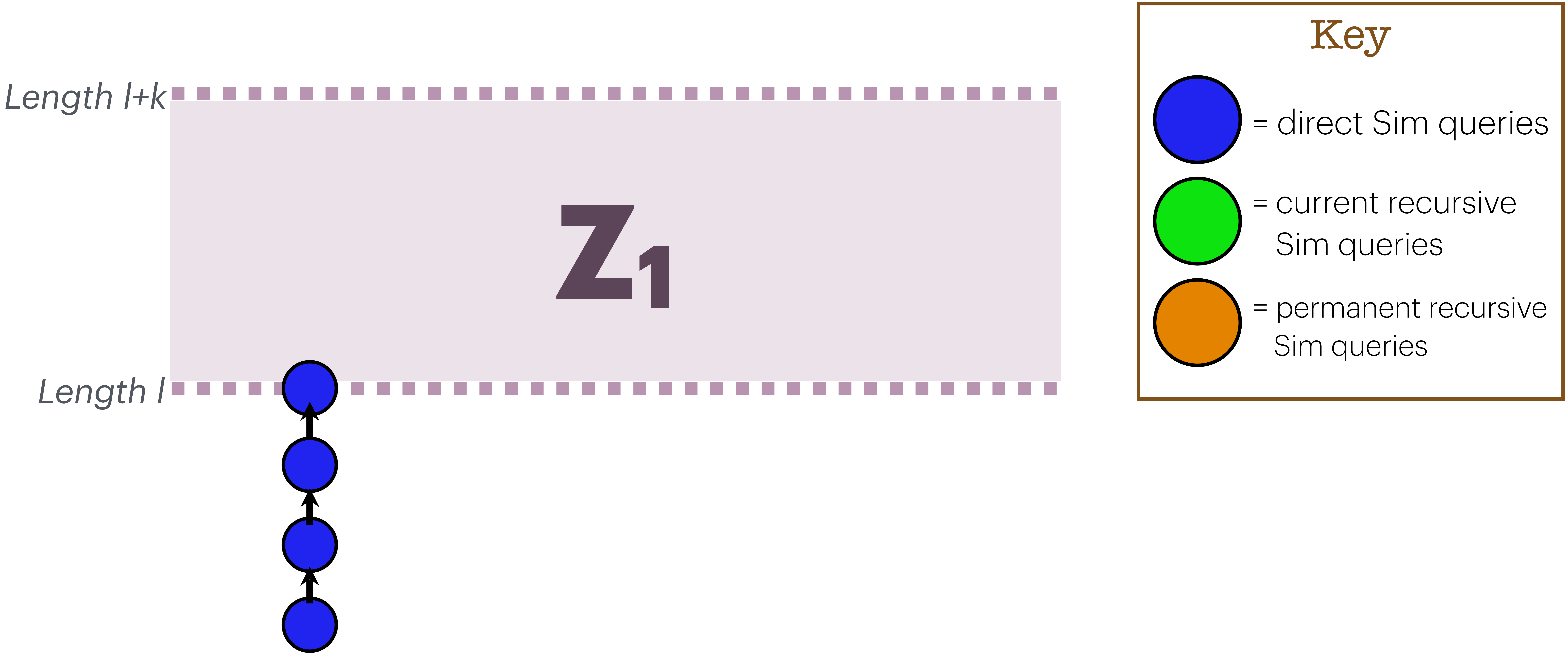


Termination

Note: only queries of length up to l followed by some Z_1 blocks "matter"

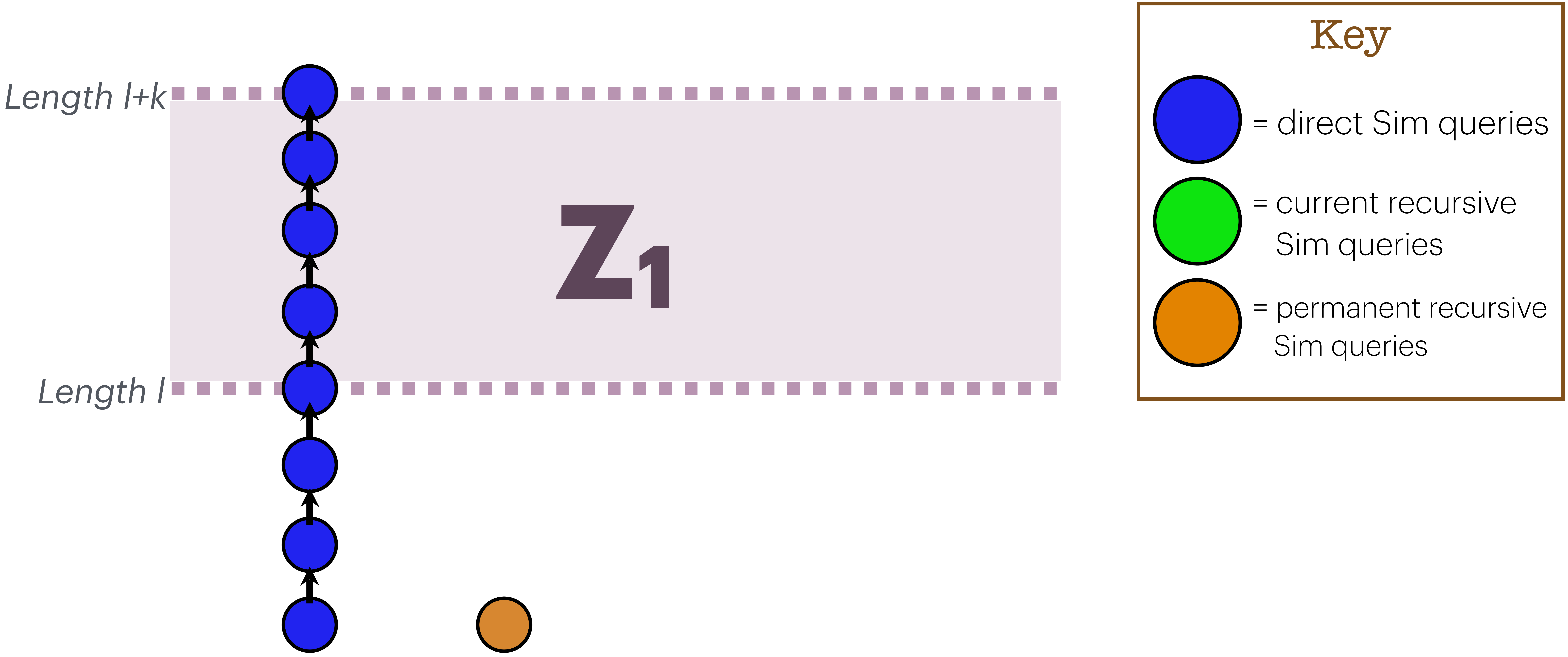
Termination

Note: only queries of length up to ***l*** followed by some Z_1 blocks "matter"



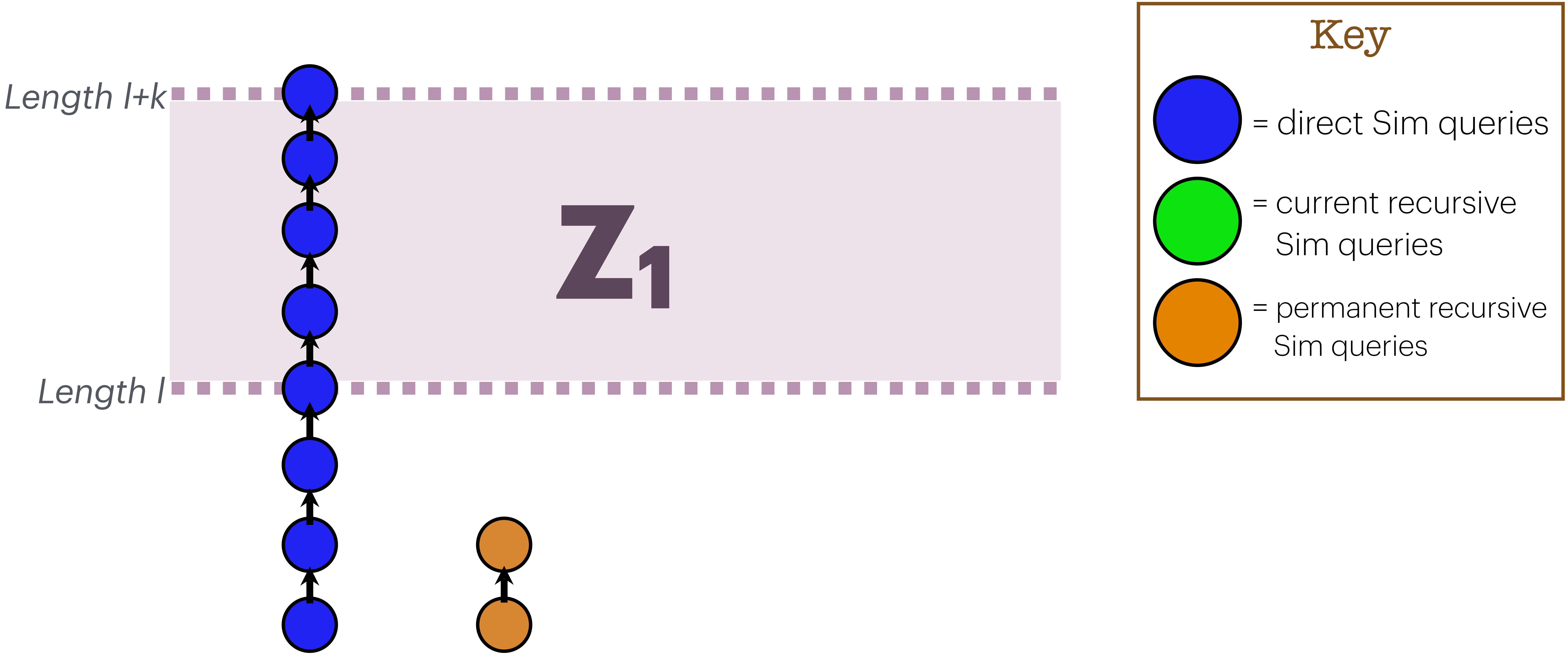
Termination

Note: only queries of length up to *l* followed by some Z_1 blocks "matter"



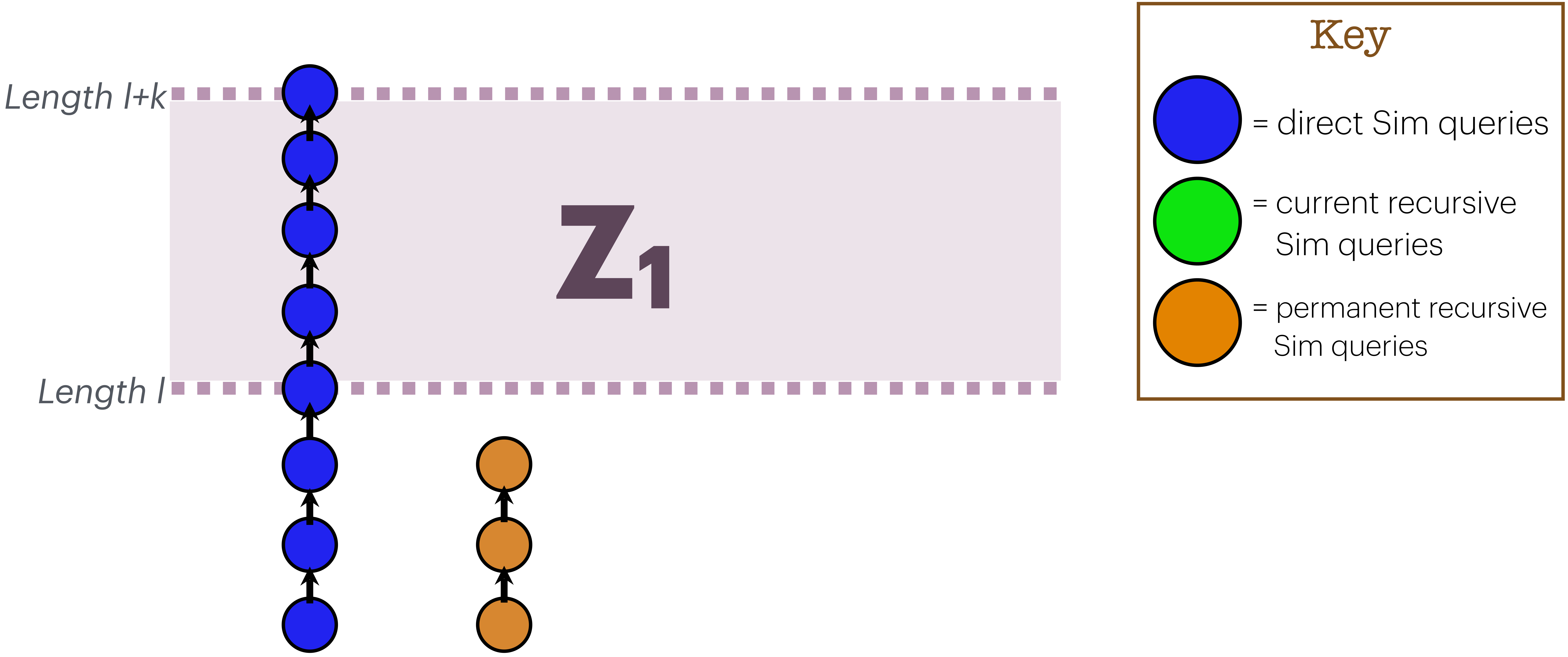
Termination

Note: only queries of length up to ***l*** followed by some Z_1 blocks "matter"



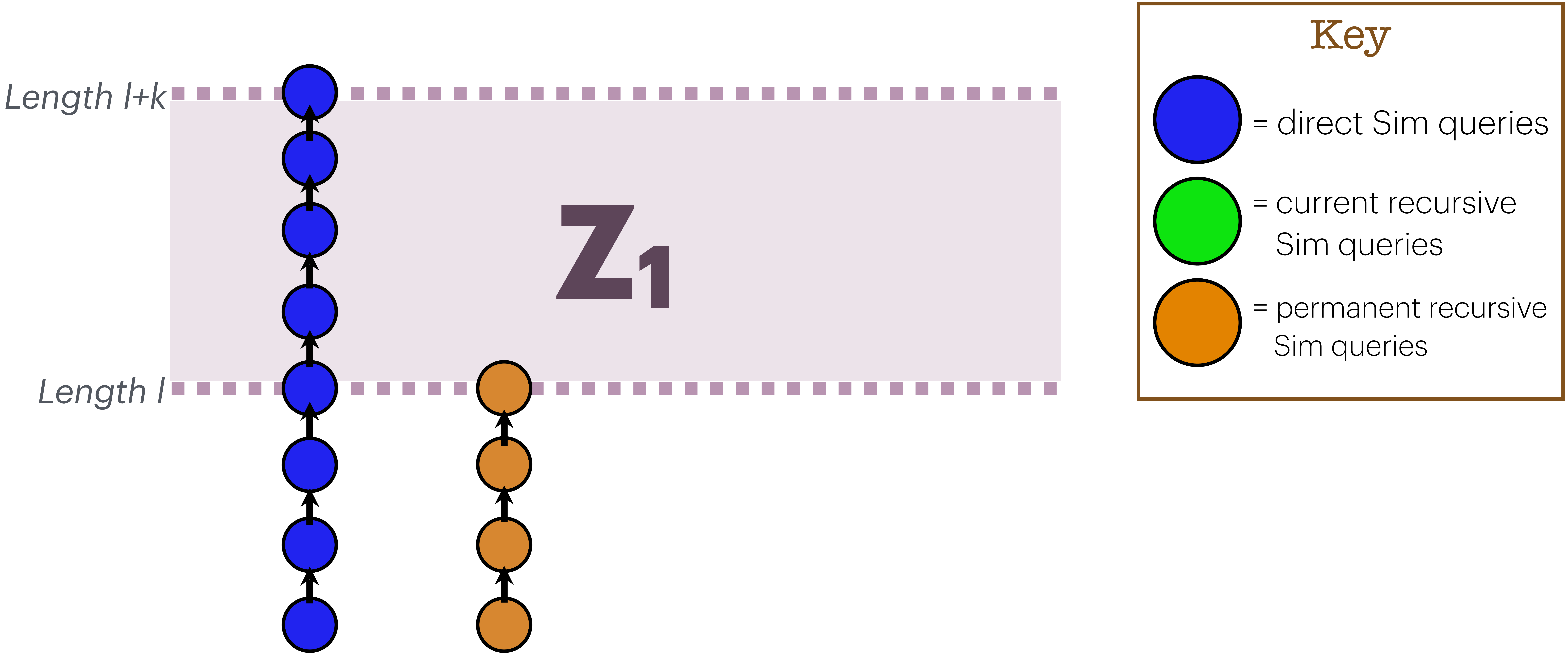
Termination

Note: only queries of length up to $\boldsymbol{l}$ followed by some Z_1 blocks "matter"



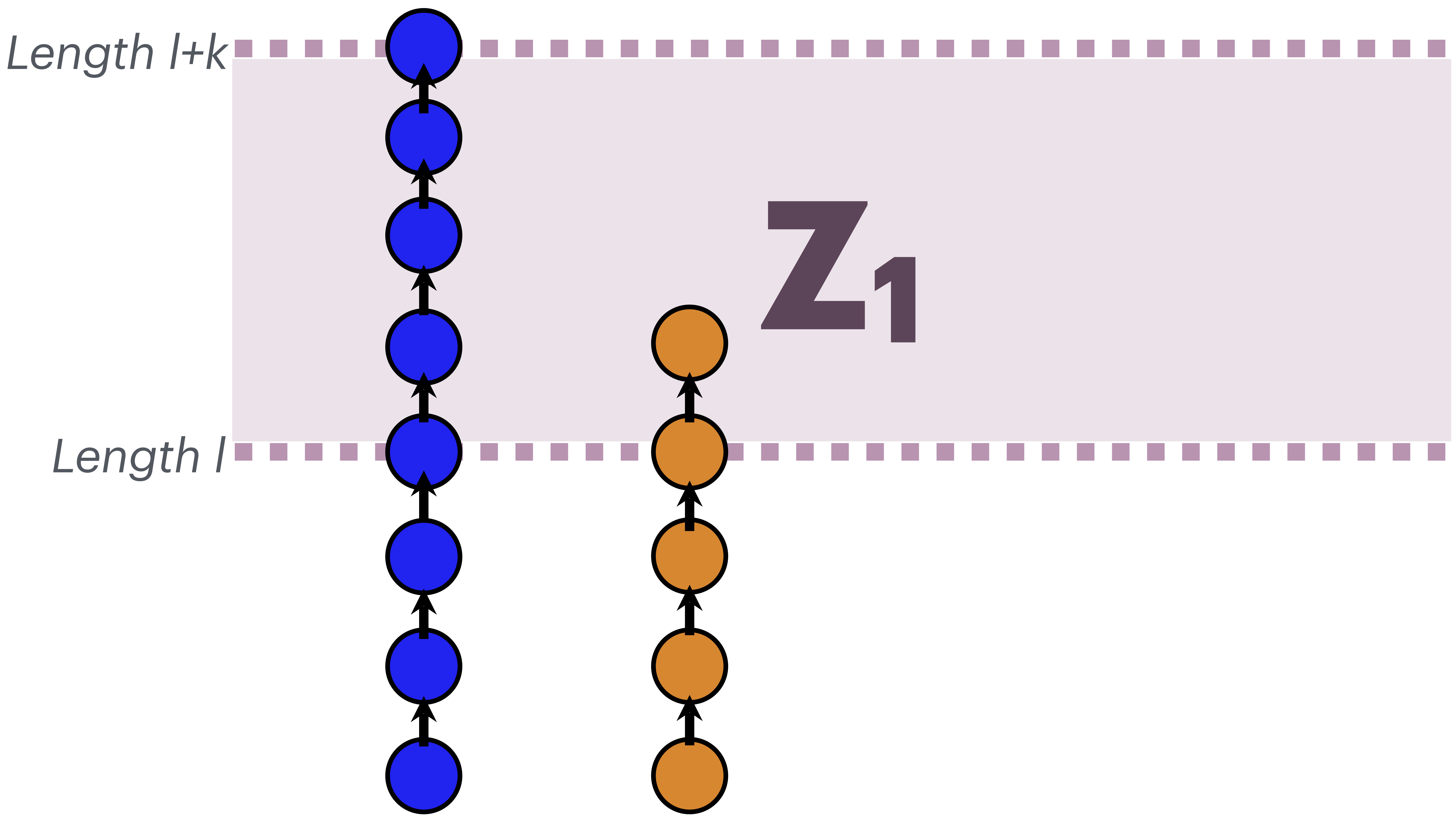
Termination

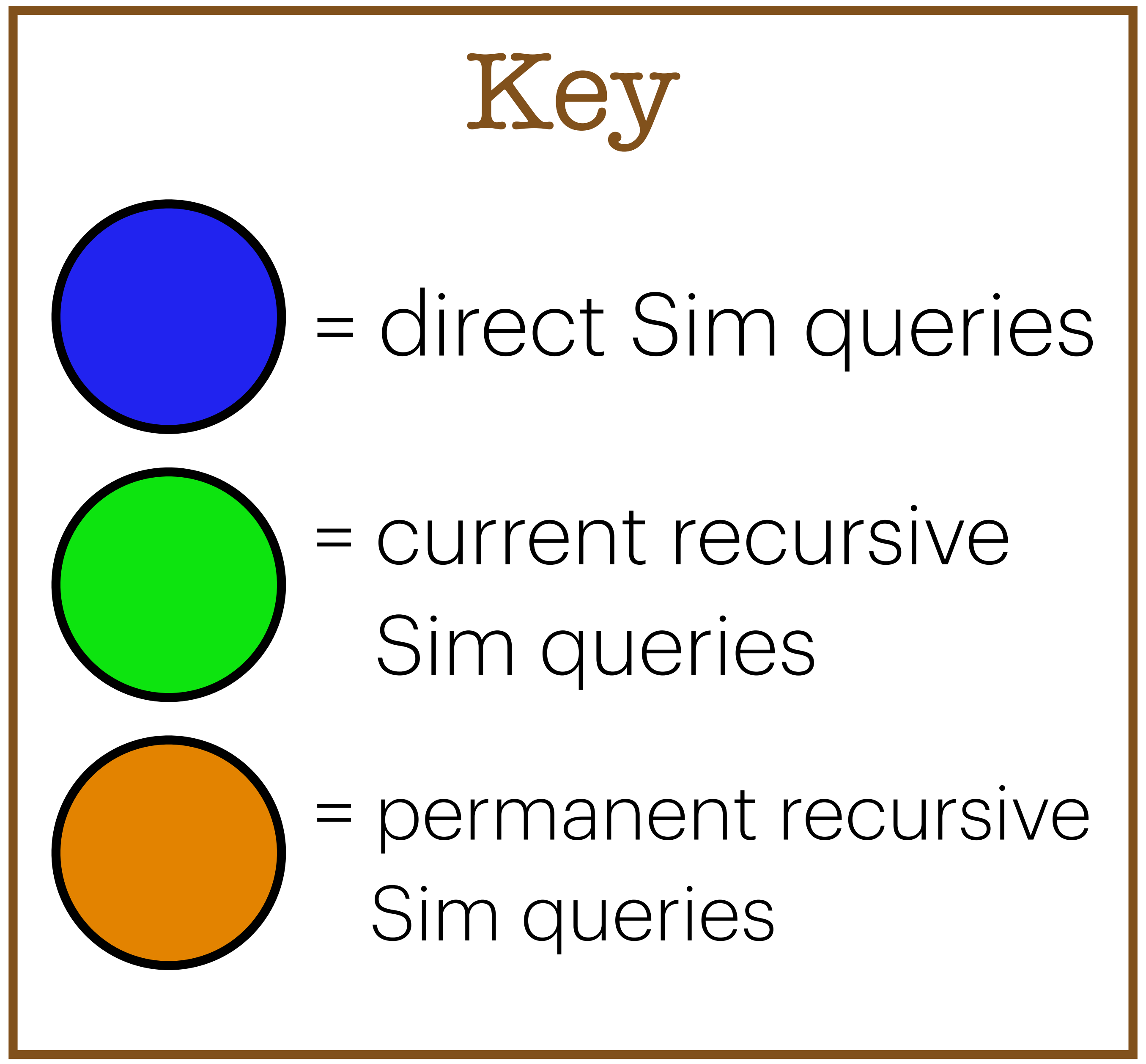
Note: only queries of length up to ***l*** followed by some Z_1 blocks "matter"



Termination

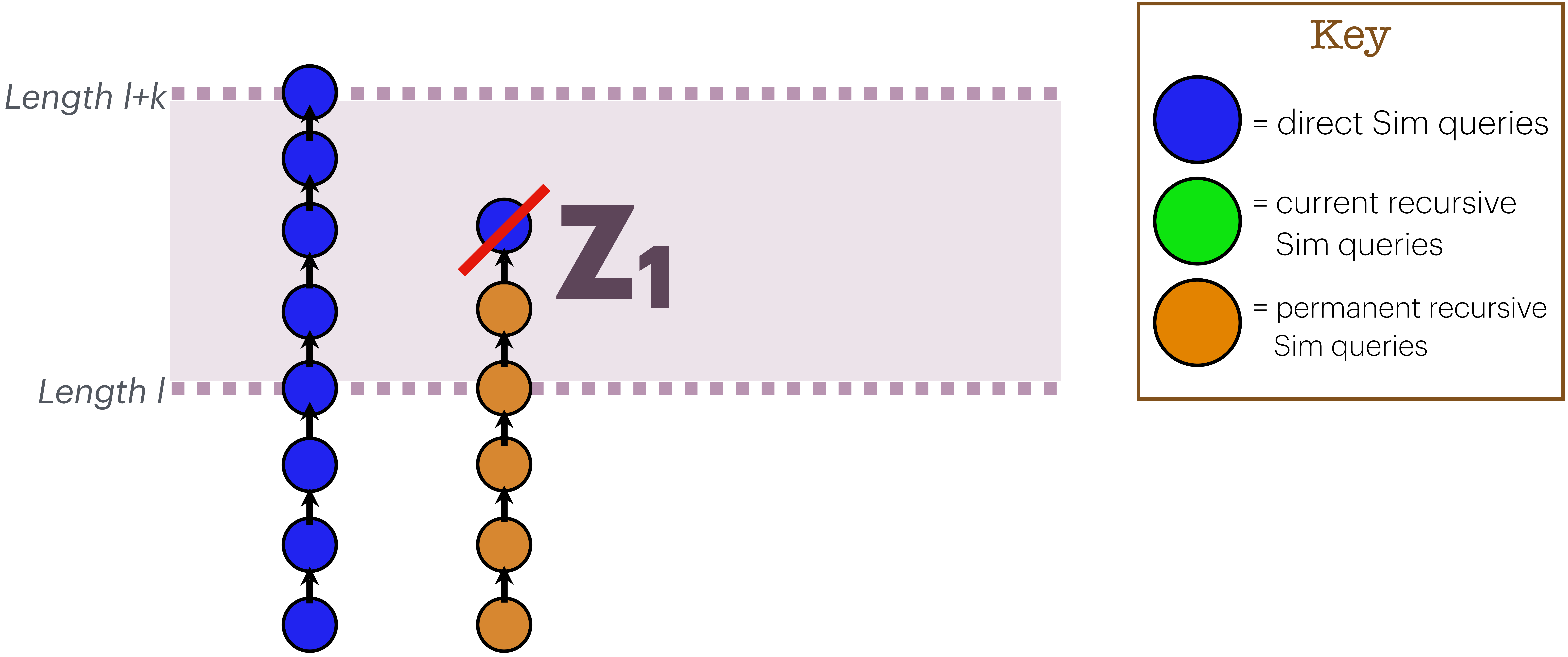
Note: only queries of length up to ***l*** followed by some Z_1 blocks "matter"





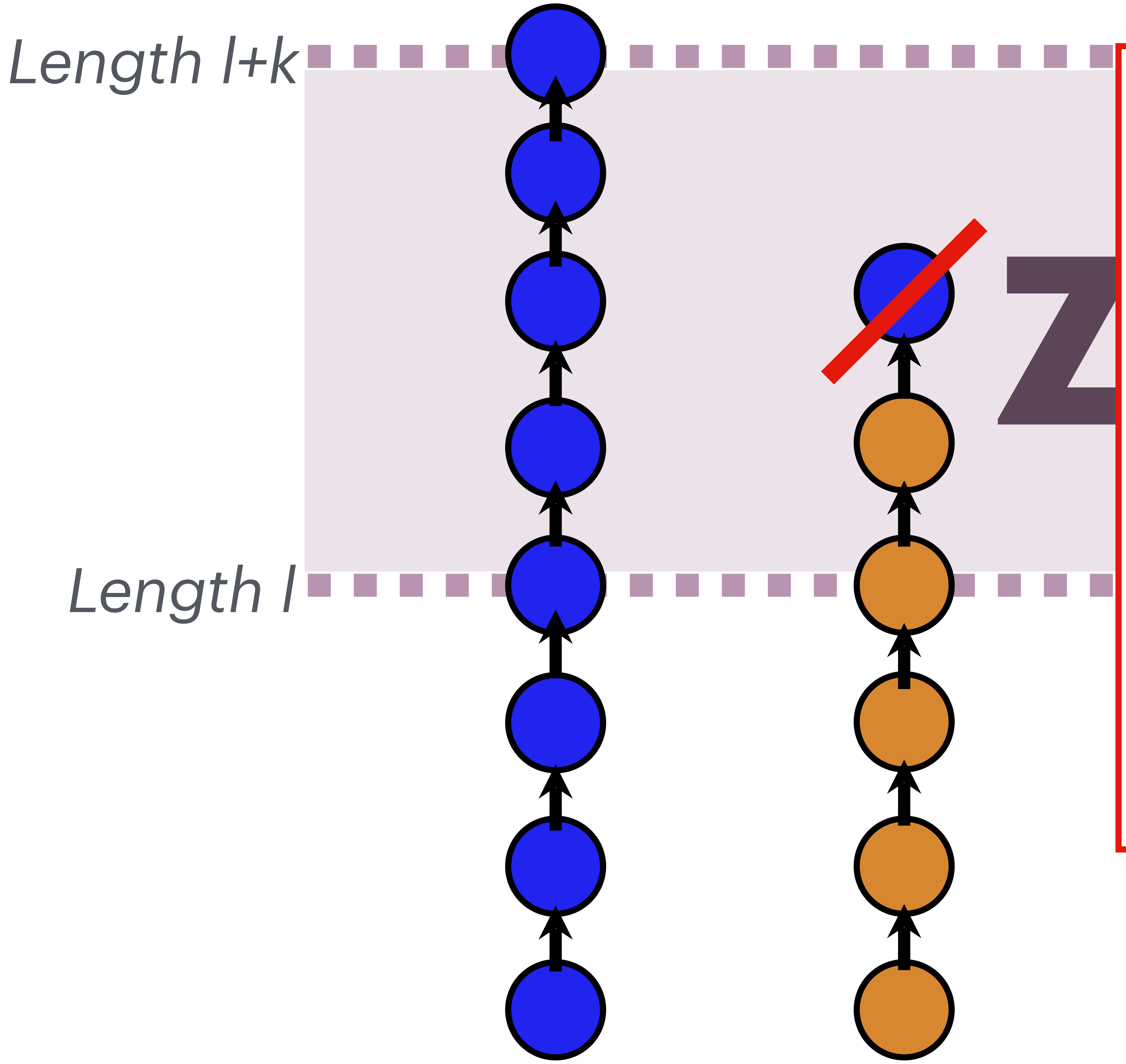
Termination

Property: blue query cannot follow orange query



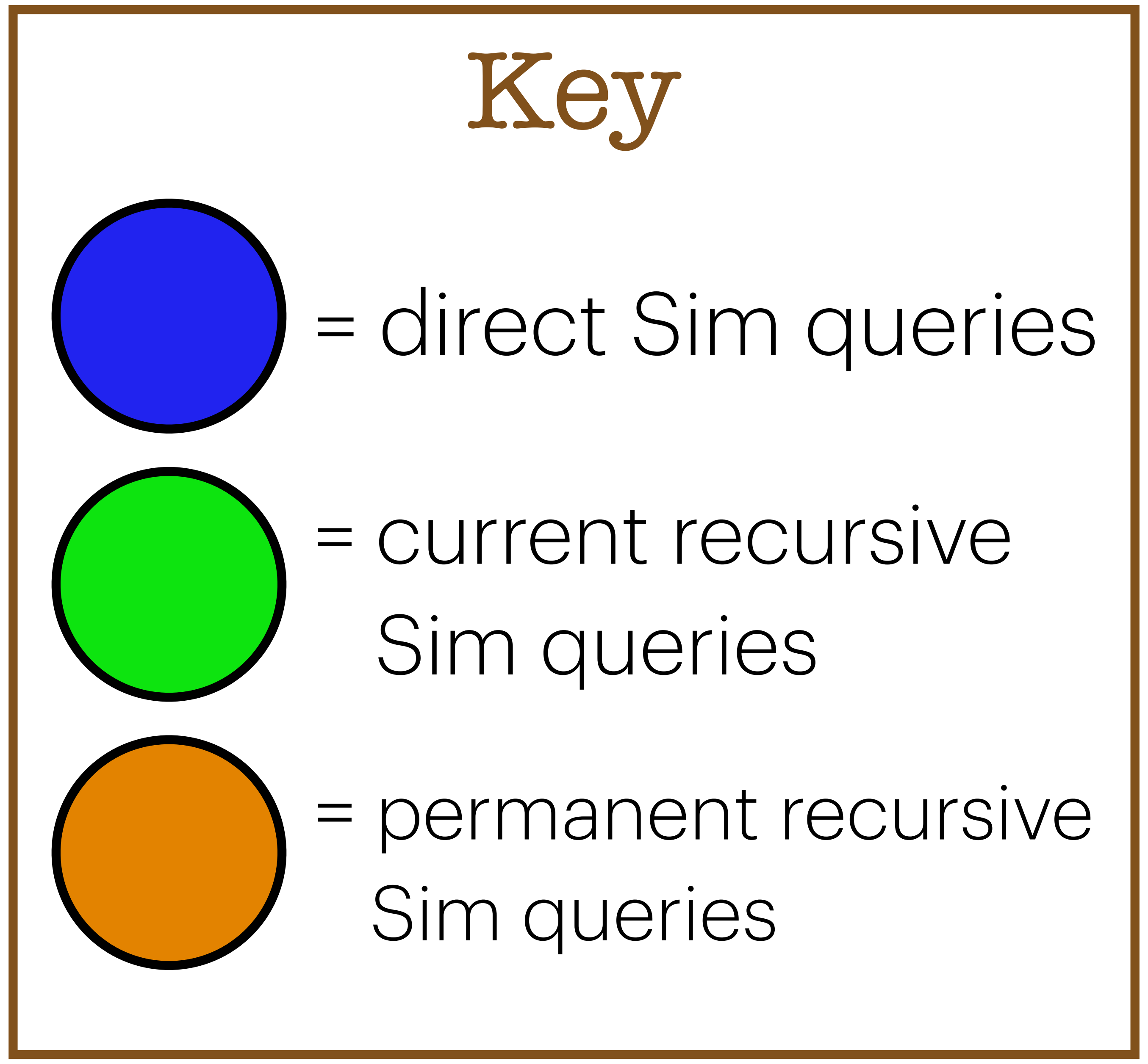
Termination

Property: blue query cannot follow orange query



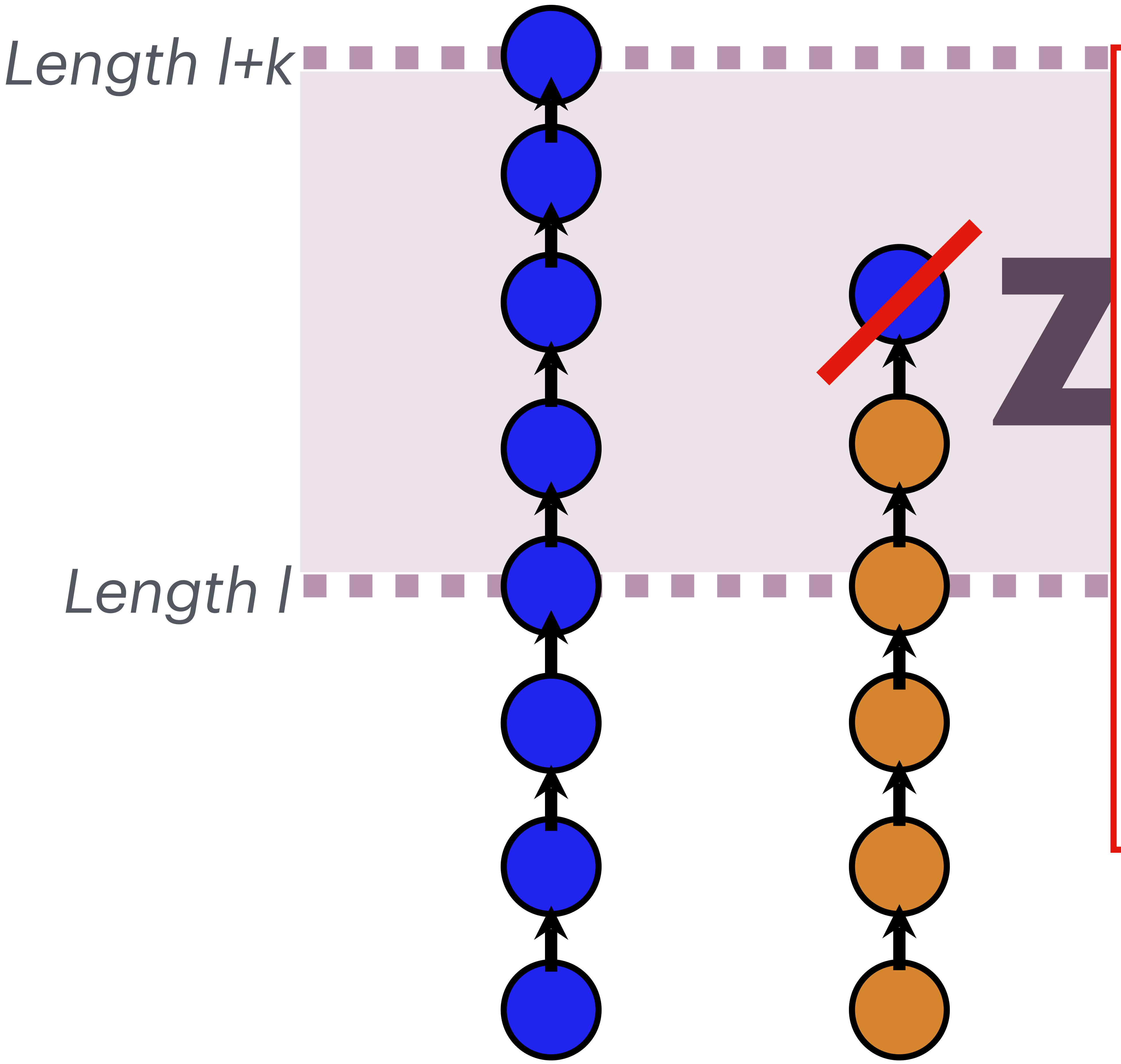
Why?

If it could, we could generate all blue queries without orange queries



Termination

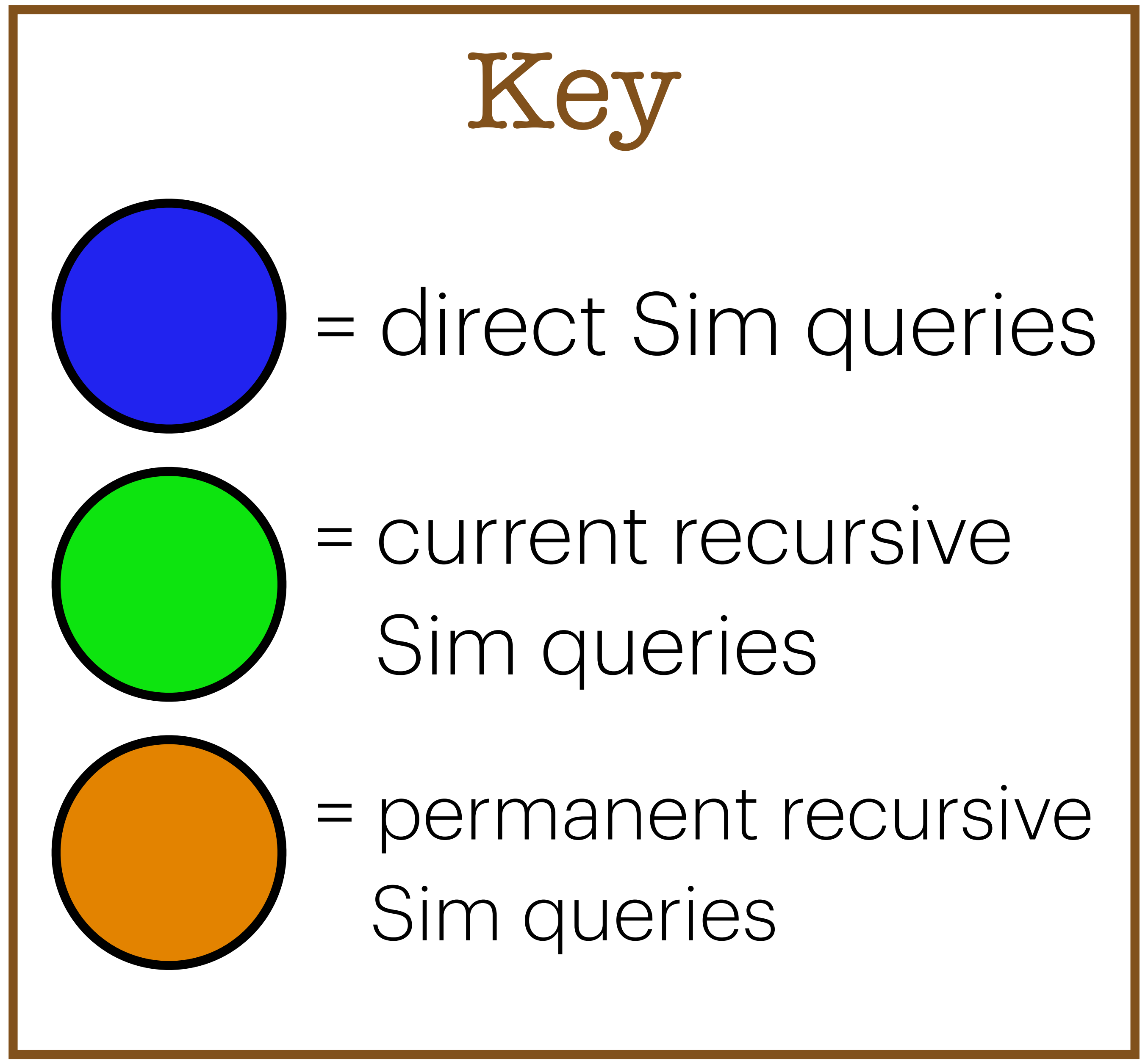
Property: blue query cannot follow orange query



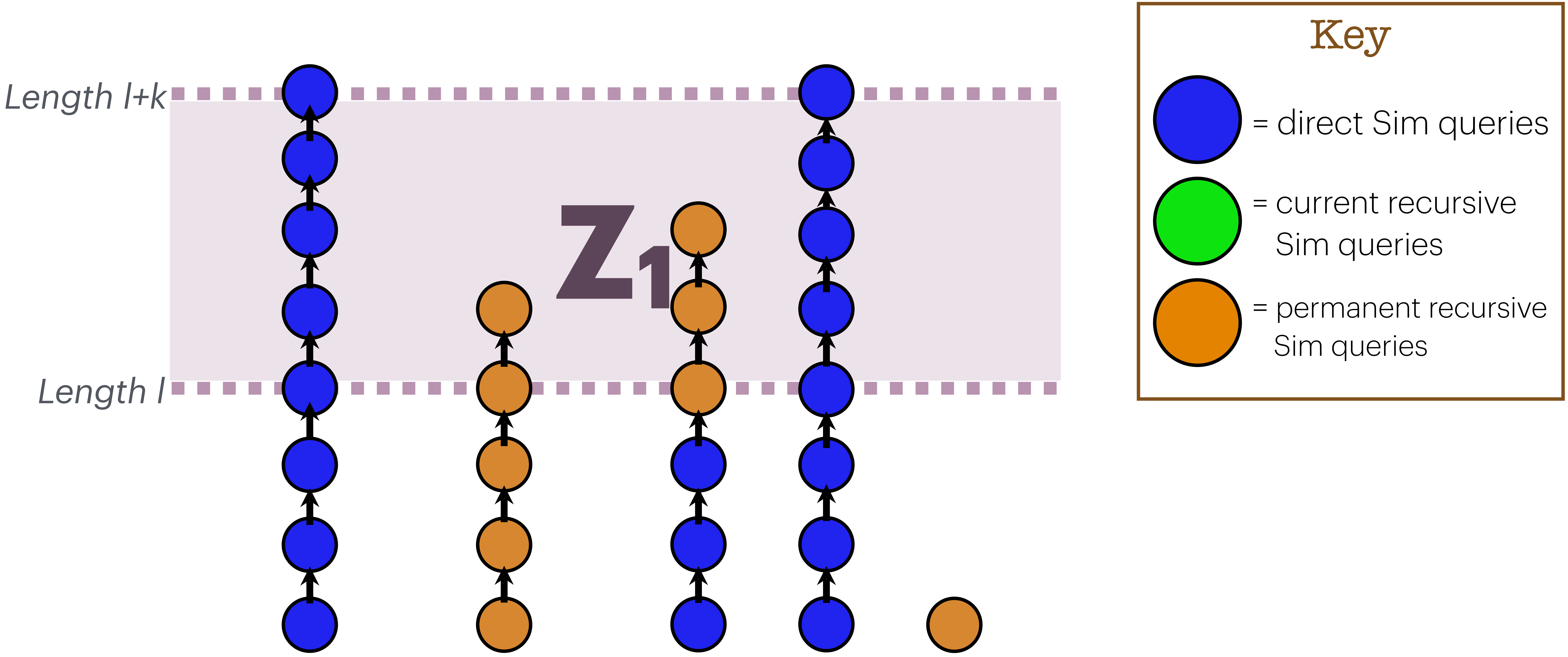
Why?

If it could, we could generate all blue queries without orange queries

Then, we could predict random oracle outputs

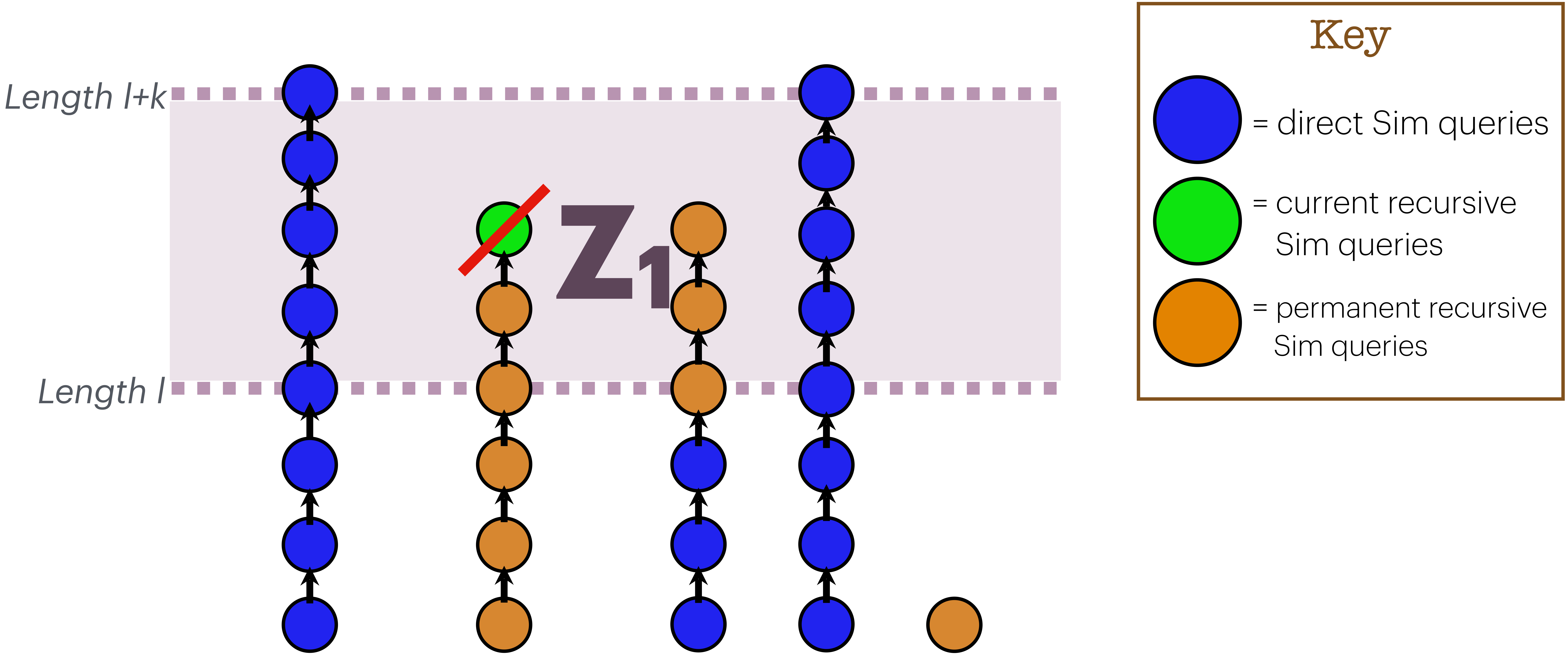


Termination



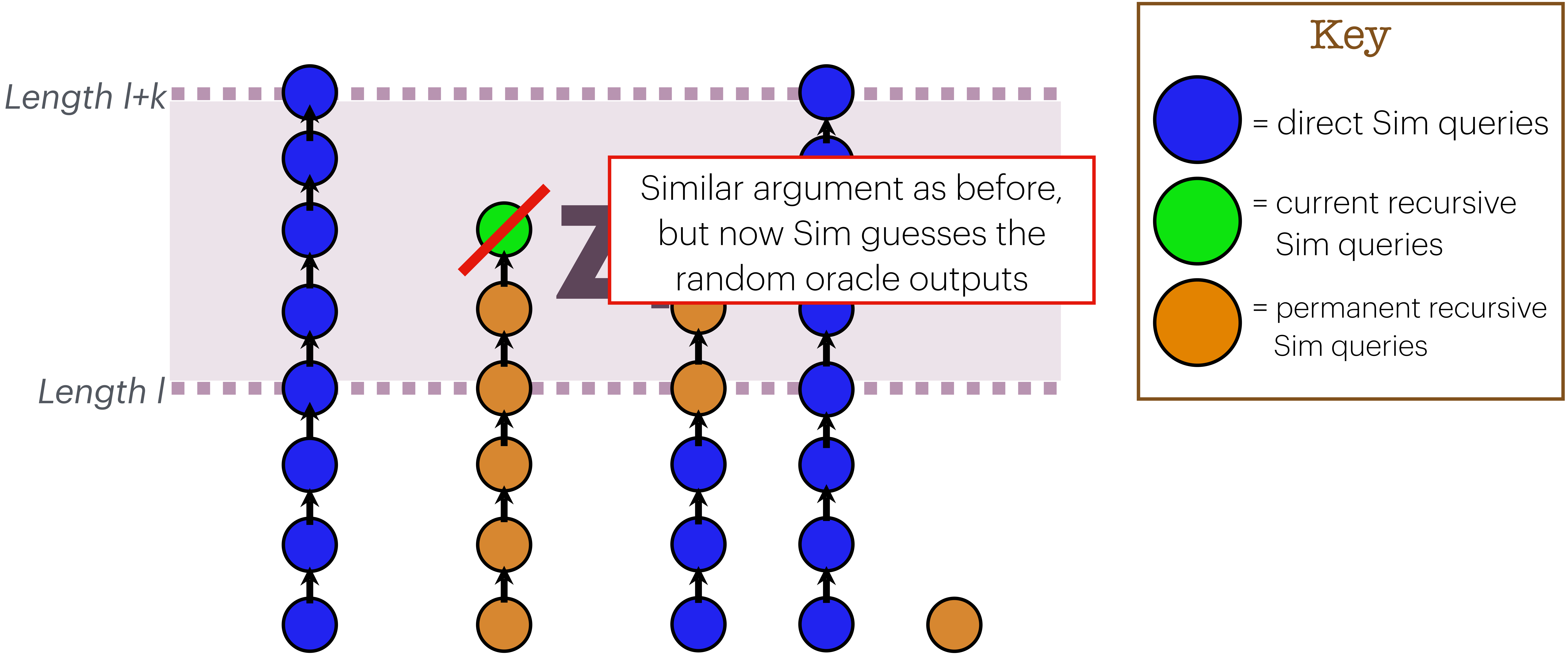
Termination

Property: green query cannot follow orange query



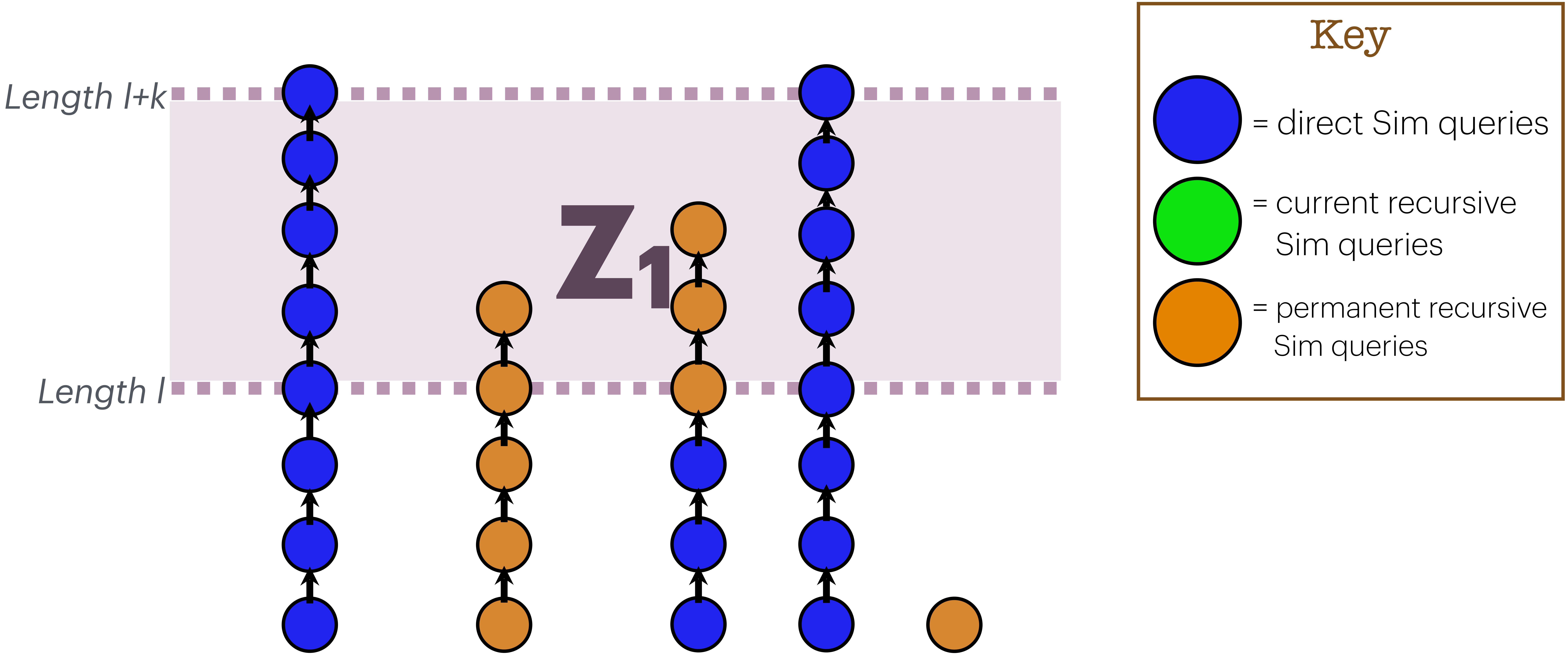
Termination

Property: green query cannot follow orange query



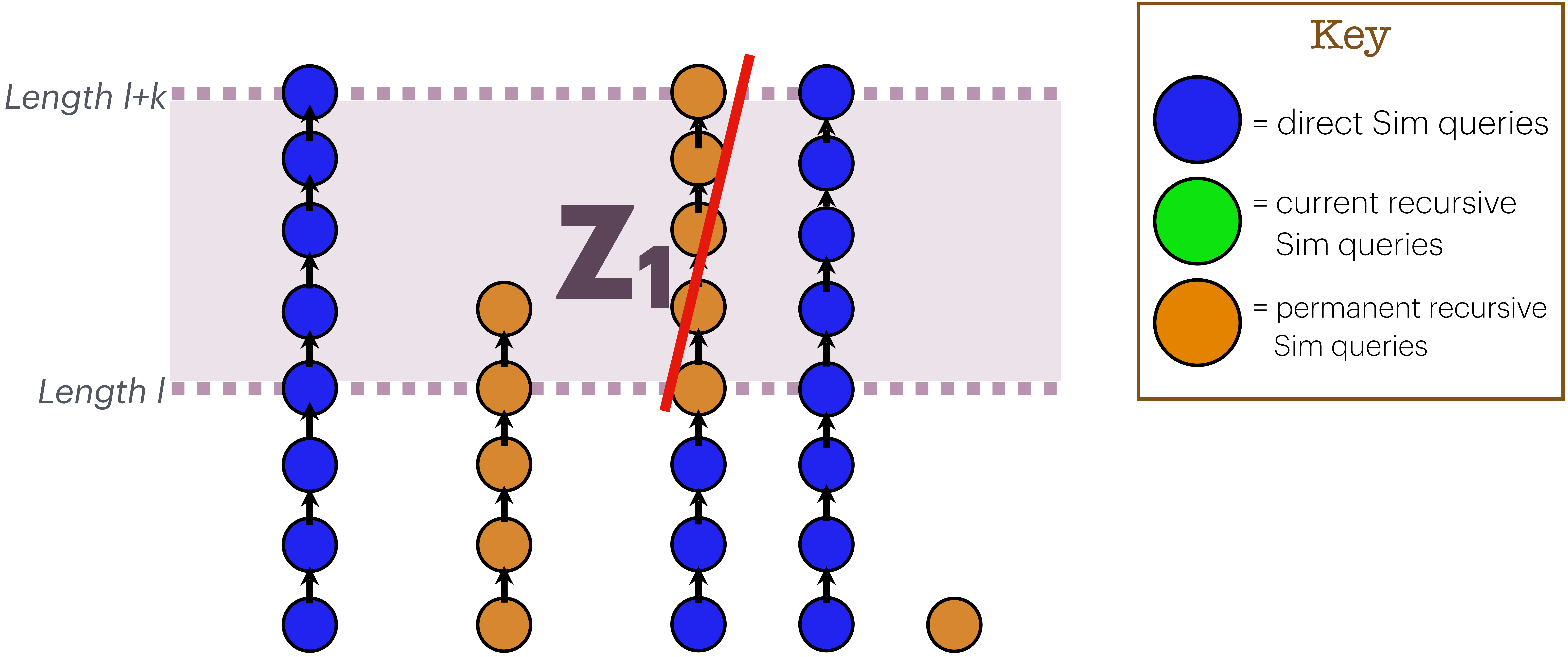
Similar argument as before, but now Sim guesses the random oracle outputs

Termination



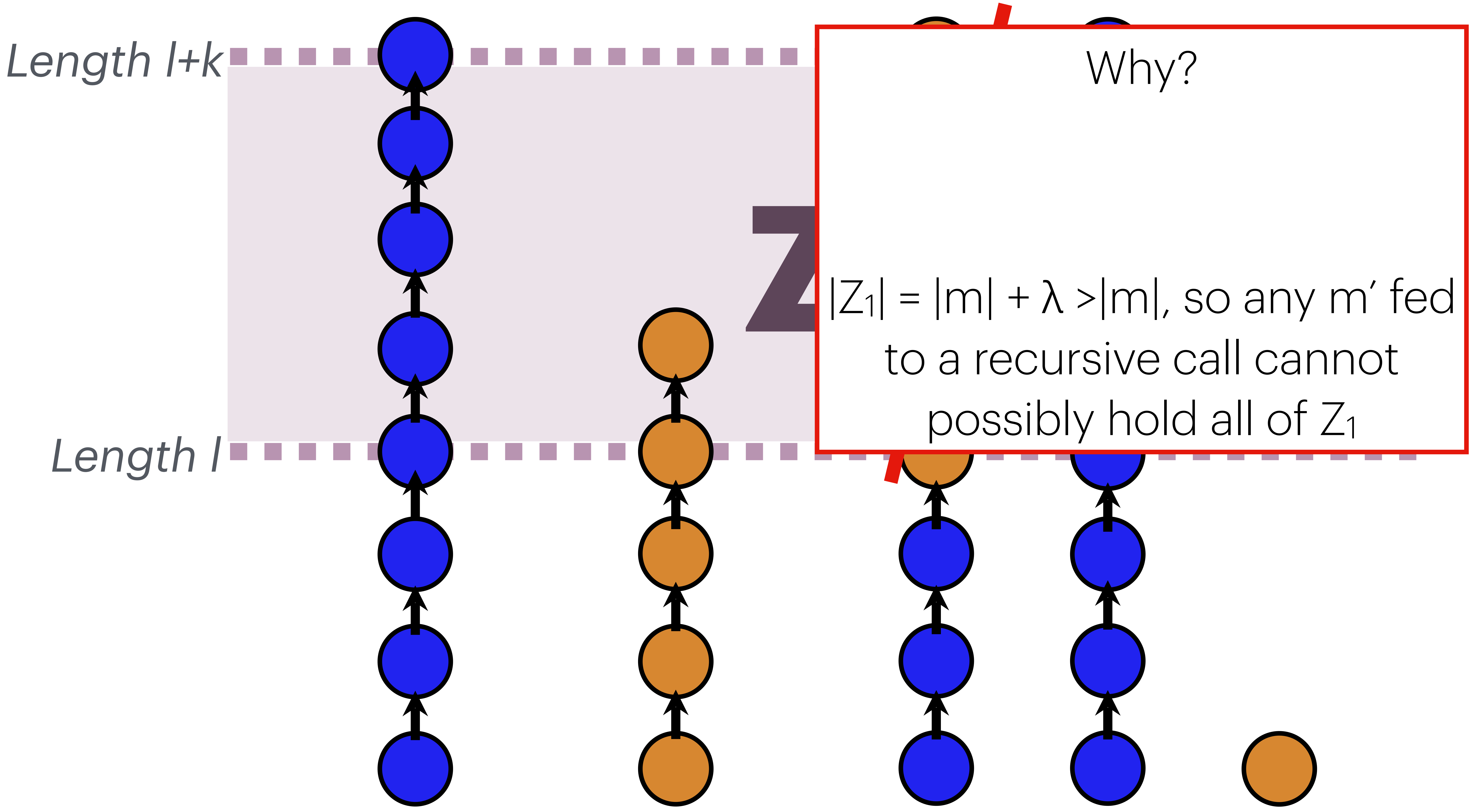
Termination

Property: orange query can't find all of Z_1



Termination

Property: orange query can't find all of Z_1



Why?

$|Z_1| = |m| + \lambda > |m|$, so any m′ fed to a recursive call cannot possibly hold all of Z_1

Key

= direct Sim queries

= current recursive Sim queries

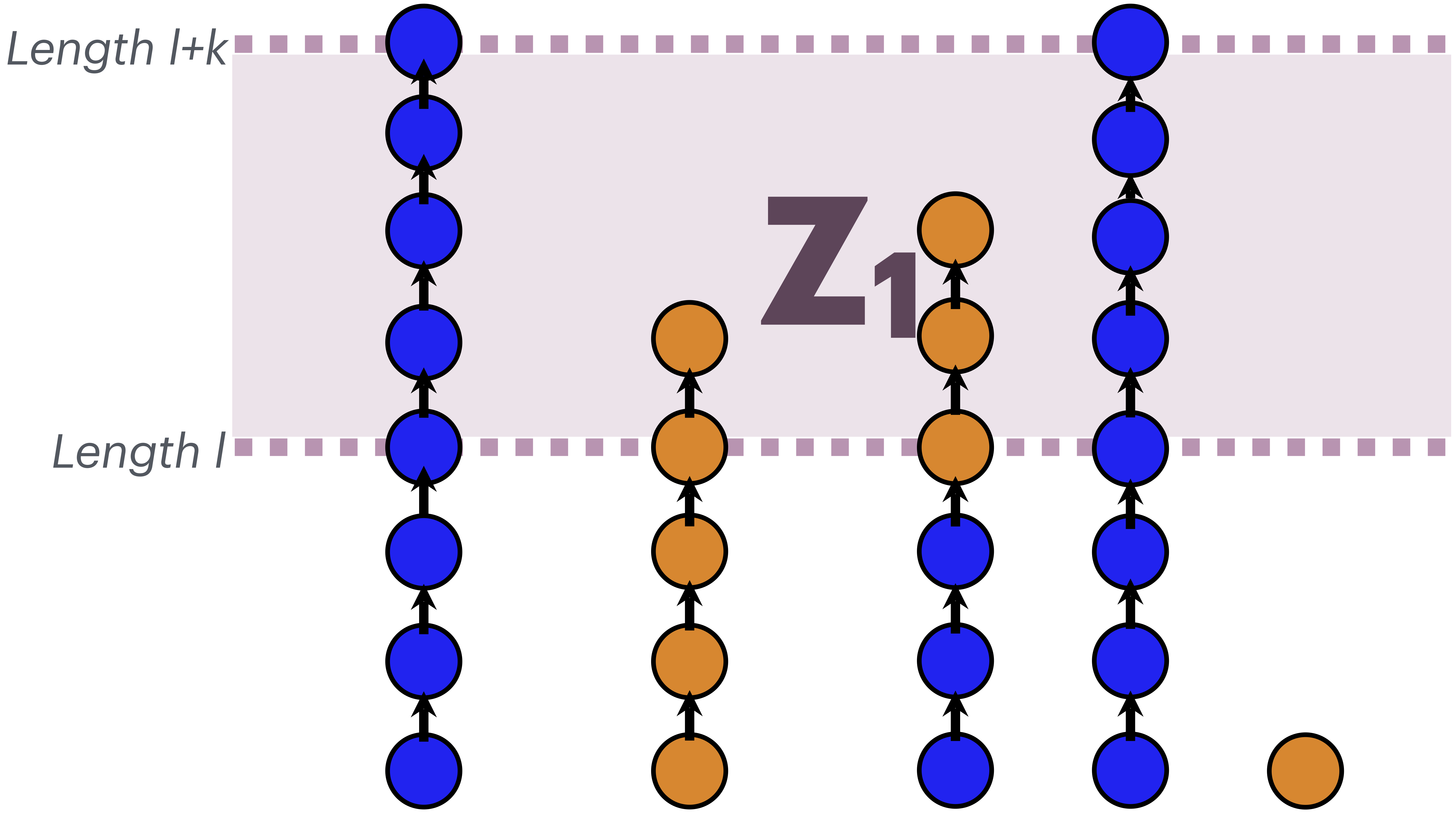
= permanent recursive Sim queries

Termination

Property 1: blue query cannot follow orange query

Property 2: green query cannot follow orange query

Property 3: orange query can't find all of Z_1

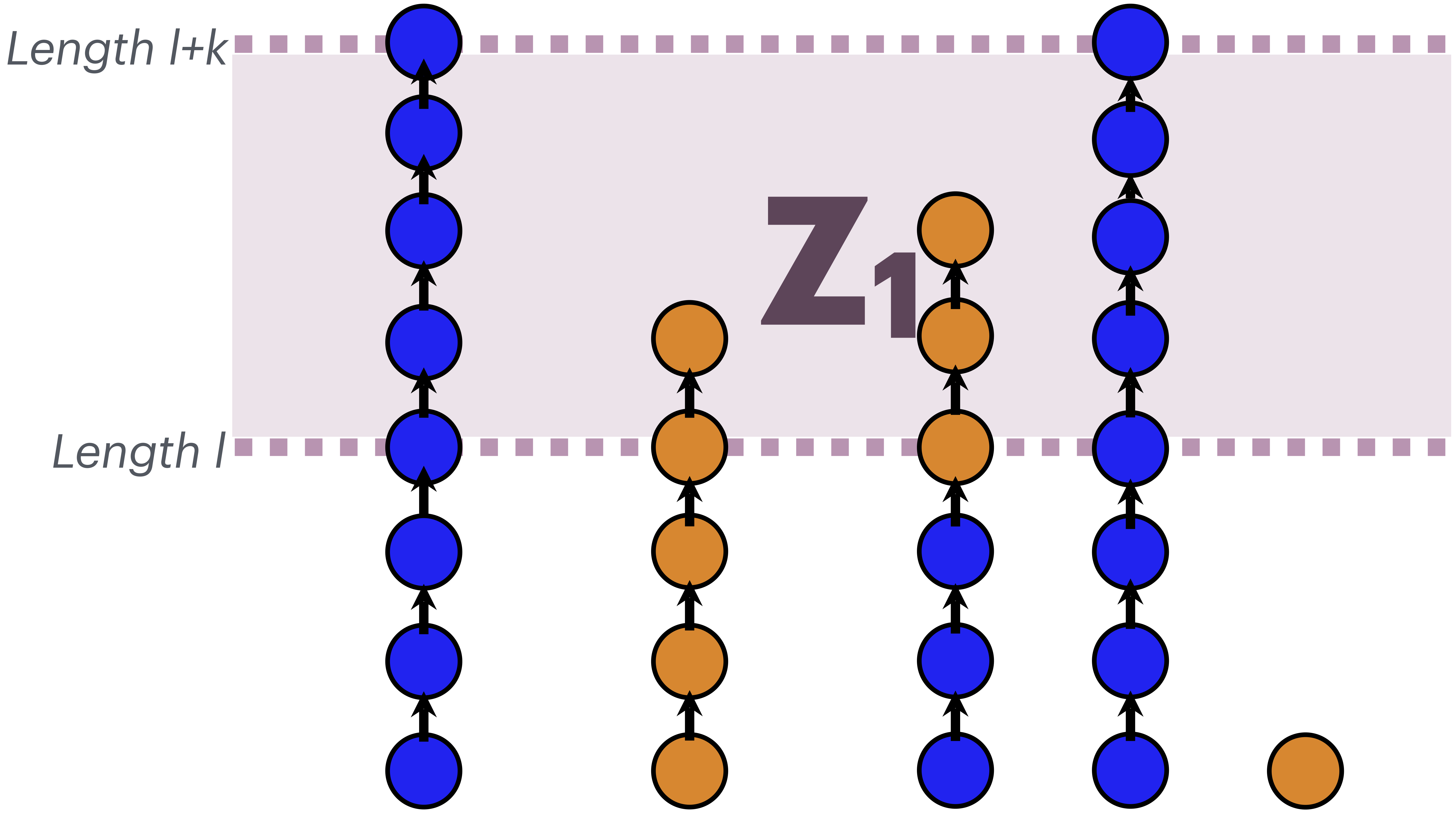


Termination

Property 1: blue query cannot follow orange query

Property 2: green query cannot follow orange query

Property 3: orange query can't find all of Z_1



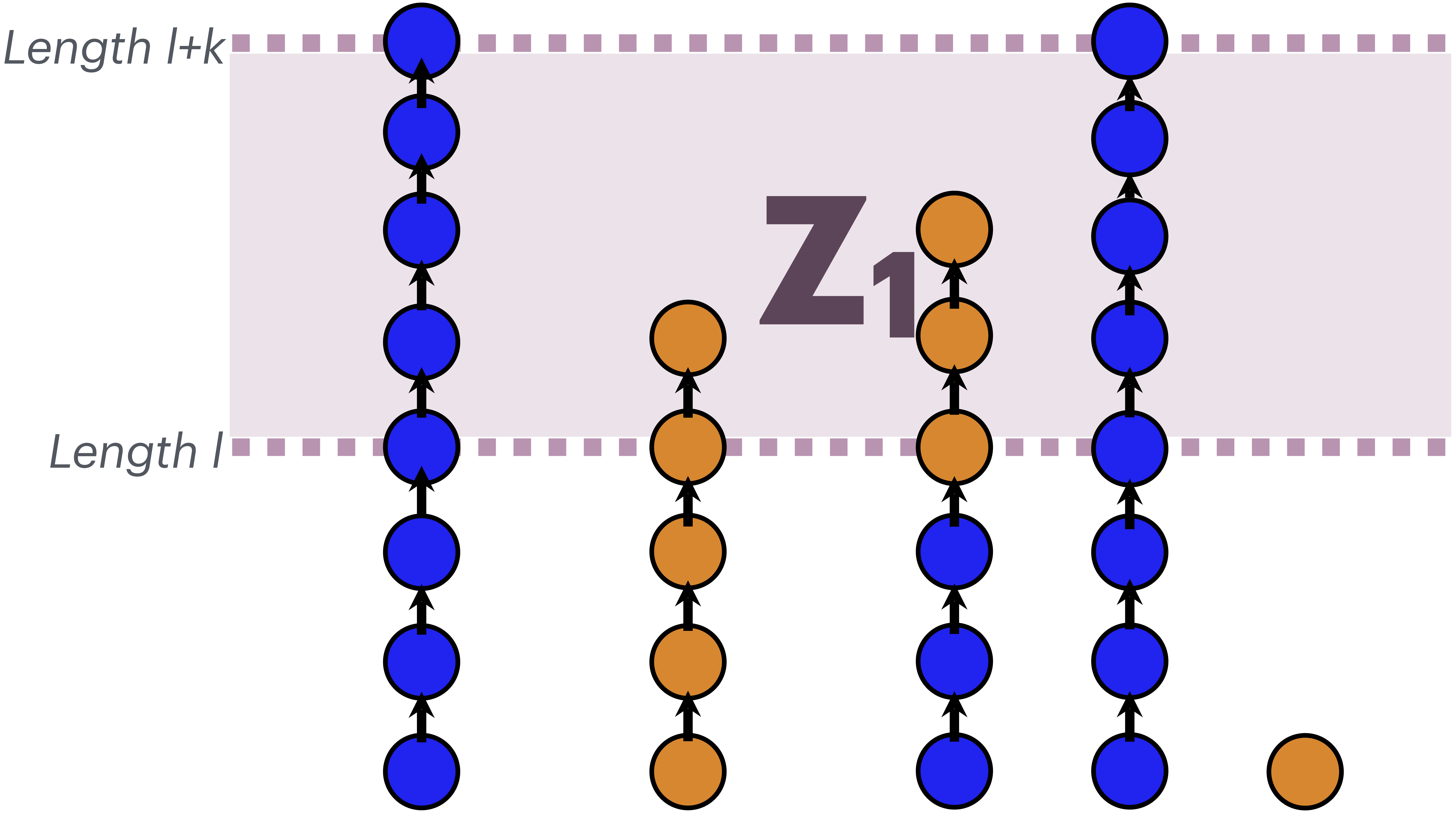
Putting these properties together, we see that orange queries can only complete blue paths (triggering recursion) that extend past ***l***=|m|, so there must be fewer recursive paths than blue ones!

Termination

Property 1: blue query cannot follow orange query

Property 2: green query cannot follow orange query

Property 3: orange query can't find all of Z_1



Putting these properties together, we see that orange queries can only complete blue paths (triggering recursion) that extend past ***l***=|m|, so there must be fewer recursive paths than blue ones!

Completes termination argument - hard part about showing Sim works in ideal

Conclusions

Conclusion and Open Questions

Conclusions

- We expand the practical utility of RO Combiners by constructing a RO Combiner for standard, Merkle-Damgård-based constructions

- We only rely on underlying compression functions being a RO using delicate termination argument

Conclusion and Open Questions

Conclusions

1. Can we construct a RO Combiner for Merkle-Damgård hashes with:

    A. Small $O(\lambda)$ salts Z_1, Z_2

    B. A constant number of calls to h_*, g_*?

4. Alternatively, can we show such a RO Combiner cannot exist? Tradeoffs?

Thank you!

2025/609