Funded by
the European Union

European Research Council
Established by the European Commission

Coding against correlated synchronization errors

João Ribeiro

Inst. Telecomunicações + Técnico-ULisboa

Based on joint work with

Yuan-Pon Chen
UIUC

Roni Con
Technion

Olgica Milenkovic
UIUC

Jin Sima
UIUC/Purdue

Synchronization errors

Erasures

1 0 1 0 1 0 1 0

↓

1 0 ? 0 1 ? ? 0

Deletions

1 0 1 0 1 0 1 0

↓

1 0 0 1 0

Synchronization errors

Erasures

1 0 1 0 1 0 1 0

↓

1 0 ? 0 1 ? ? 0

Deletions

1 0 1 0 1 0 1 0

↓

1 0 0 1 0

Synchronization errors

Erasures

1 0 1 0 1 0 1 0

↓

1 0 ? 0 1 ? ? 0

Deletions

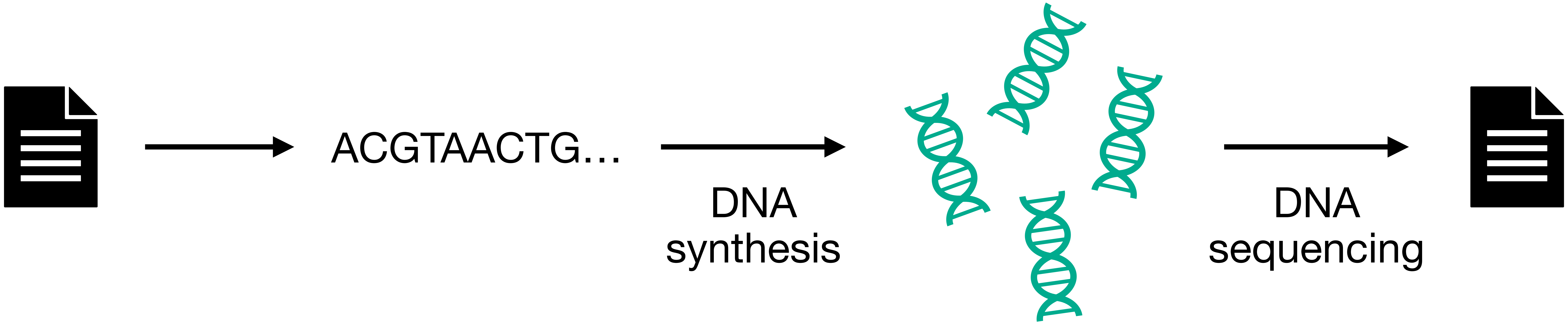
1 0 1 0 1 0 1 0

↓

1 0 0 1 0

Why deletions and insertions?

- Simple types of synchronization errors.
- Most existing techniques fail.
- DNA-based data storage systems.



Usually… i.i.d. errors

- Most works have focused on understanding channels with **iid** insertions and deletions.

Usually… i.i.d. errors

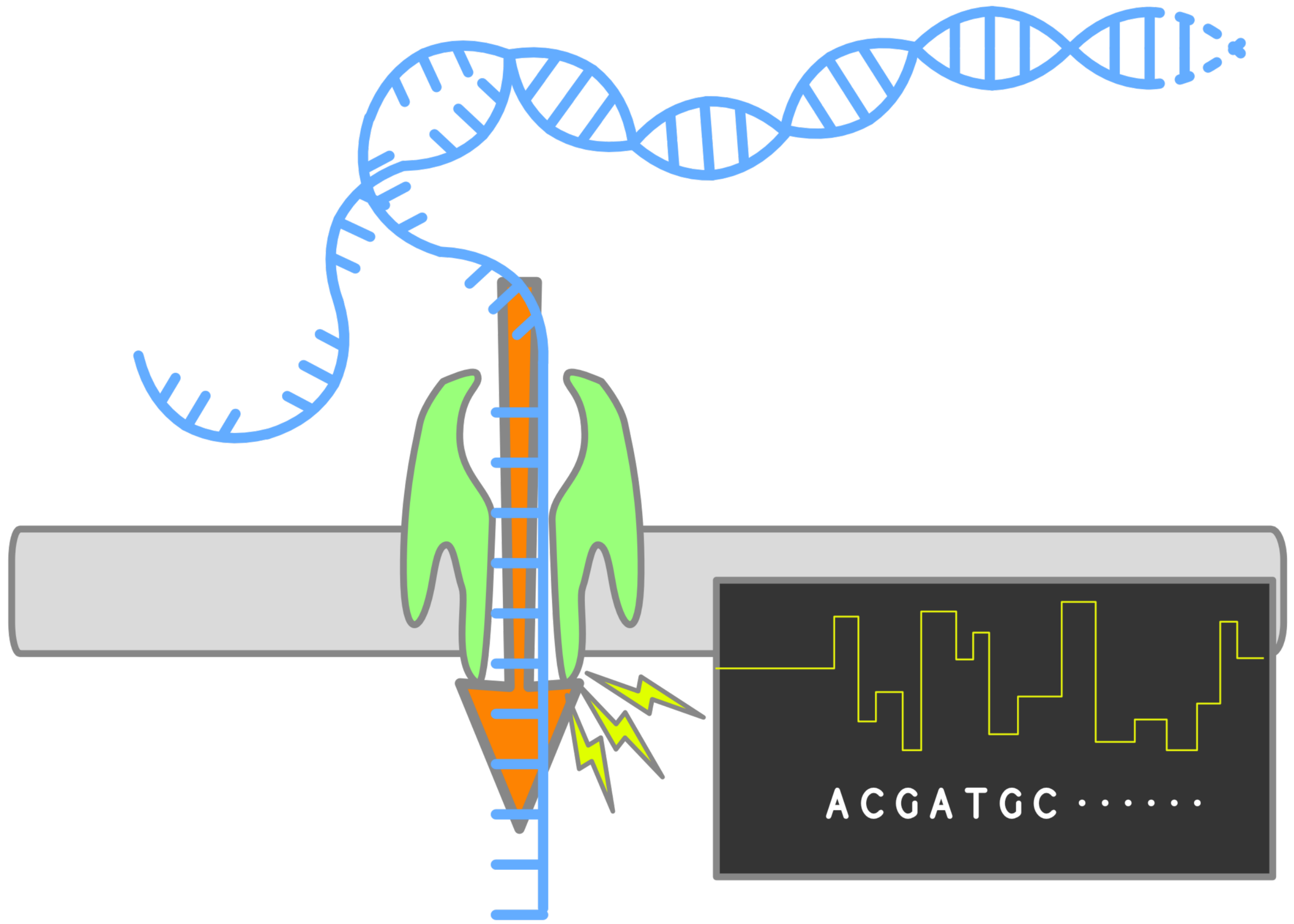
- Most works have focused on understanding channels with **iid** insertions and deletions.

- **But the world is not iid! Error distribution can be affected by the environment and structure of input.**
  This was pointed out already in early works on DNA storage.

Concrete examples

In DNA-based data storage with nanopore-based sequencing, deletions happen more often in or around long runs.

[Yazdi, Gabrys, Milenkovic 2017], [Weindel, Gimpel, Grass, Heckel 2023]



Credit: DataBase Center for Life Science (DBCLS), https://commons.wikimedia.org/wiki/File:202001_nanopore_sequencing.svg

Concrete examples

In DNA-based data storage with nanopore-based sequencing, deletions happen more often in or around long runs.

[Yazdi, Gabrys, Milenkovic 2017], [Weindel, Gimpel, Grass, Heckel 2023]

We can refine standard theoretical error models to capture these effects:

- Long runs experience higher error rates;
- Bits following long runs experience higher error rates.

Concrete examples

In DNA-based data storage with nanopore-based sequencing, deletions happen more often in or around long runs.

[Yazdi, Gabrys, Milenkovic 2017], [Weindel, Gimpel, Grass, Heckel 2023]

We can refine standard theoretical error models to capture these effects:

- **Long runs experience higher error rates;**
- Bits following long runs experience higher error rates.

Runlength-dependent deletion probabilities

Channel parameter: $d : \mathbb{N} \to [0,1]$

Behavior: bit in run of length ℓ is independently deleted with probability $d(\ell)$

0 0 1 1 1 1 0 0 0

$d(2)$ $d(4)$ $d(3)$

Concrete examples

In DNA-based data storage with nanopore-based sequencing, deletions happen more often in or around long runs.

[Yazdi, Gabrys, Milenkovic 2017], [Weindel, Gimpel, Grass, Heckel 2023]

We can refine standard theoretical error models to capture these effects:

- **Long runs experience higher error rates;**
- **Bits following long runs experience higher error rates.** **discussed later!**

Runlength-dependent deletion probabilities

Channel parameter: $d : \mathbb{N} \to [0,1]$

Behavior: bit in run of length ℓ is independently deleted with probability $d(\ell)$

0 0 1 1 1 1 0 0 0

$d(2)$ $d(4)$ $d(3)$

Probabilistic channels with contextual synchronization errors

Probabilistic channels with contextual synchronization errors

- We'd like to understand the coding capacity of these channels.

Probabilistic channels with contextual synchronization errors

- We'd like to understand the coding capacity of these channels.
- Often it's useful to work instead with the information capacity $\liminf_{n\to\infty} \frac{1}{n} \sup_{X^n} I(X^n; Y_{X^n})$

Probabilistic channels with contextual synchronization errors

- We'd like to understand the coding capacity of these channels.
- Often it's useful to work instead with the information capacity $\liminf_{n\to\infty} \frac{1}{n} \sup_{X^n} I(X^n; Y_{X^n})$
- But it's not clear whether *coding capacity = information capacity*!

Probabilistic channels with contextual synchronization errors

- We'd like to understand the coding capacity of these channels.
- Often it's useful to work instead with the information capacity $\liminf_{n\to\infty} \frac{1}{n} \sup_{X^n} I(X^n; Y_{X^n})$
- But it's not clear whether *coding capacity = information capacity*!

Dobrushin 1967: true for i.i.d. synchronization errors.

Probabilistic channels with contextual synchronization errors

- We'd like to understand the coding capacity of these channels.
- Often it's useful to work instead with the information capacity $\liminf_{n\to\infty} \frac{1}{n} \sup_{X^n} I(X^n; Y_{X^n})$
- But it's not clear whether *coding capacity* = *information capacity*!

Dobrushin 1967: true for i.i.d. synchronization errors.

Mao-Diggavi-Kannan 2018: true for "finite-window" contextual synchronization errors.

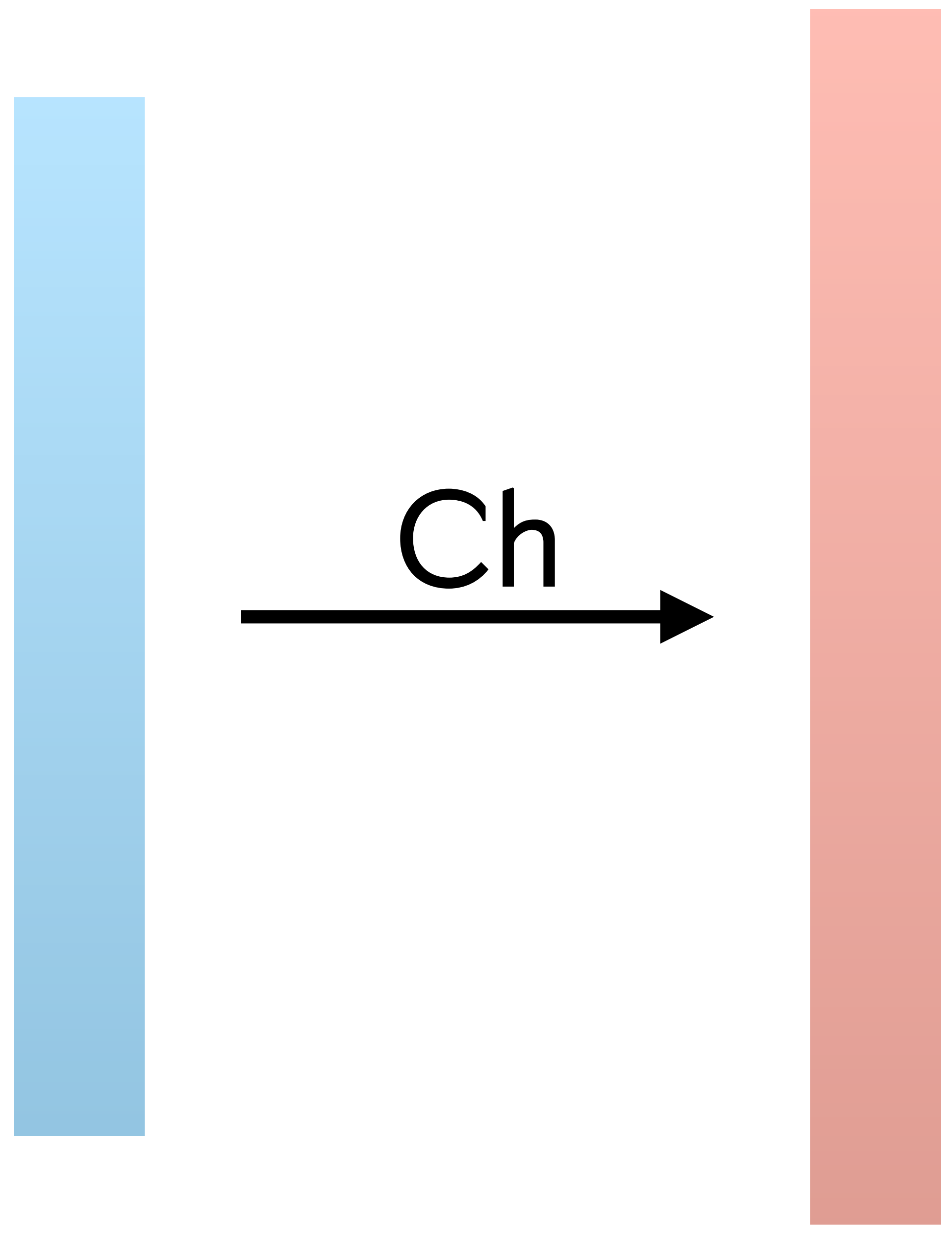
Given $x \in \{0,1\}^n$, channel outputs $Y_1 \| Y_2 \| \cdots \| Y_n$ with $Y_i \in \{0,1\}^*$ dependent on $x_i, x_{i-1}, \ldots, x_{i-w}$ for some constant w.

Probabilistic channels with contextual synchronization errors

Ignoring some things, suppose the channel Ch satisfies the following:

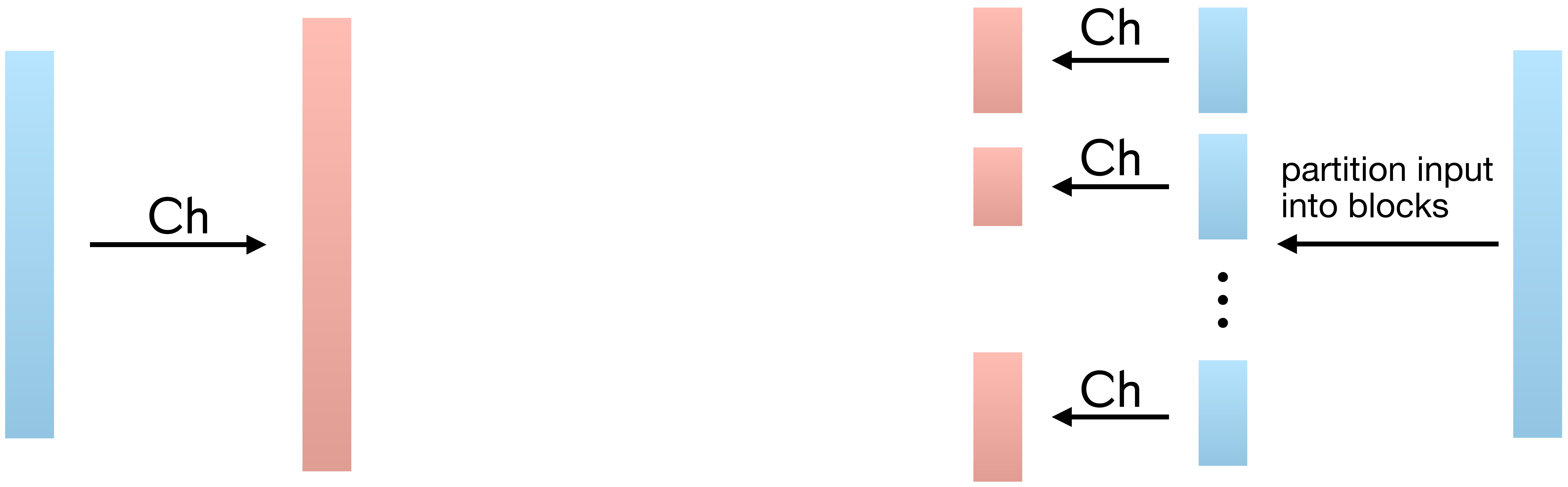
Probabilistic channels with contextual synchronization errors

Ignoring some things, suppose the channel Ch satisfies the following:



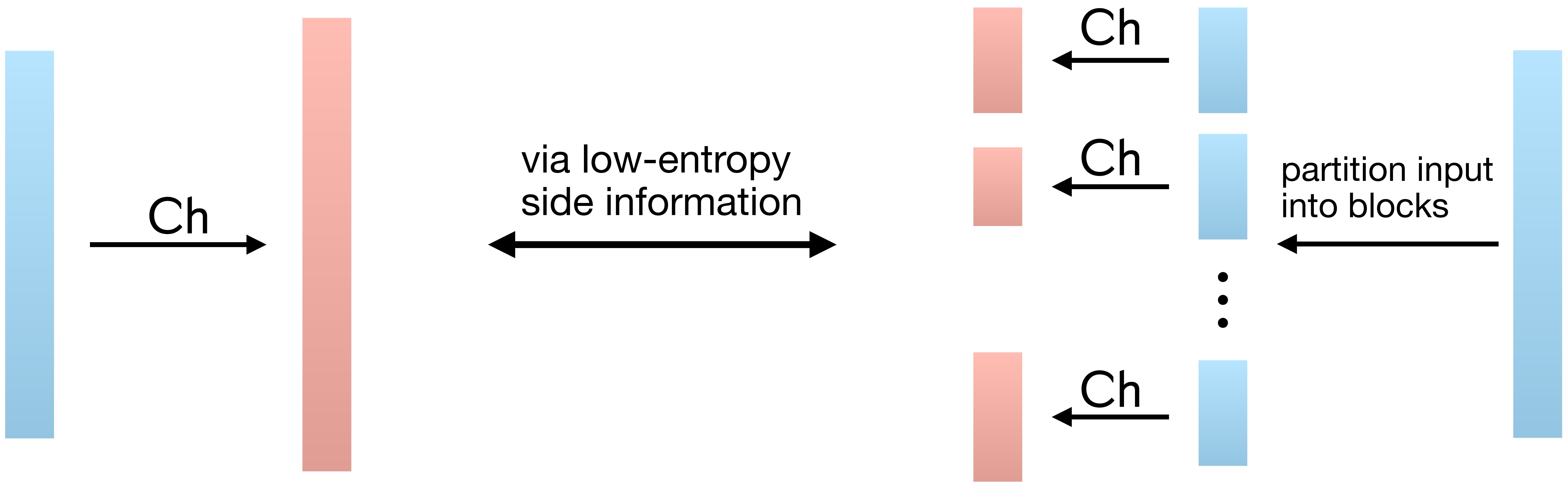
Probabilistic channels with contextual synchronization errors

Ignoring some things, suppose the channel Ch satisfies the following:



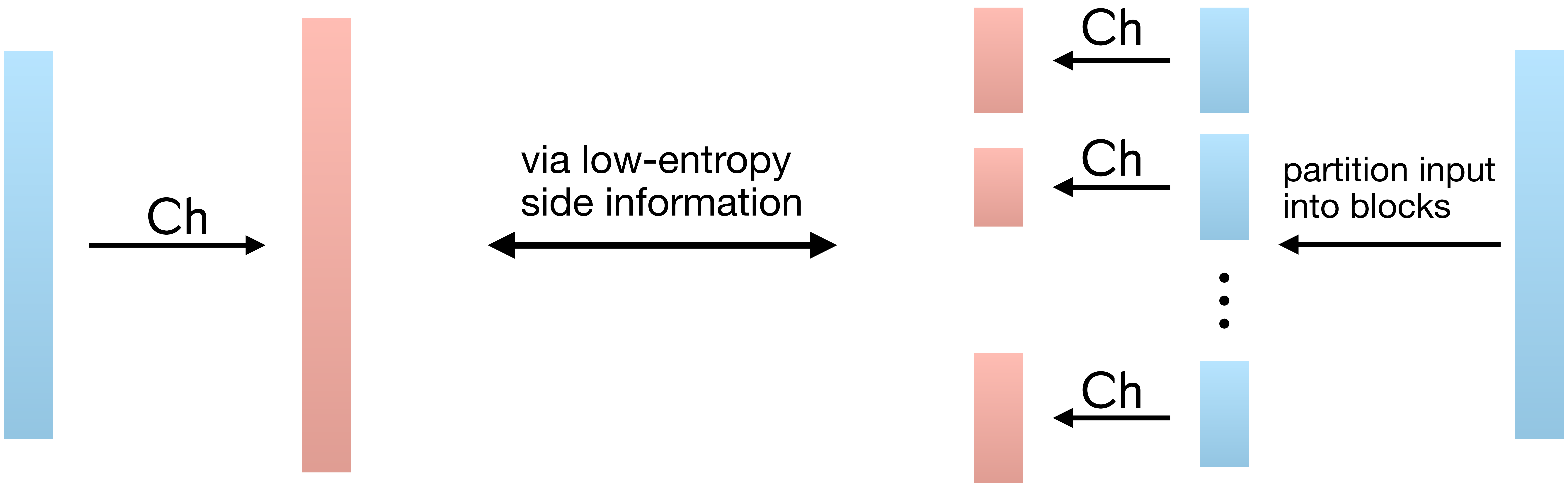
Probabilistic channels with contextual synchronization errors

Ignoring some things, suppose the channel Ch satisfies the following:



Probabilistic channels with contextual synchronization errors

Ignoring some things, suppose the channel Ch satisfies the following:



with Roni Con:

Under this assumption, coding capacity = information capacity, and capacity is achieved by Markov input processes.

Probabilistic channels with contextual synchronization errors

with Roni Con:

Under more general conditions, coding capacity = information capacity, and capacity is achieved by Markov input processes.

Probabilistic channels with contextual synchronization errors

with Roni Con:

Under more general conditions, coding capacity = information capacity, and capacity is achieved by Markov input processes.

- Applies to contextual errors with unbounded context window

Probabilistic channels with contextual synchronization errors

with Roni Con:

Under more general conditions, coding capacity = information capacity, and capacity is achieved by Markov input processes.

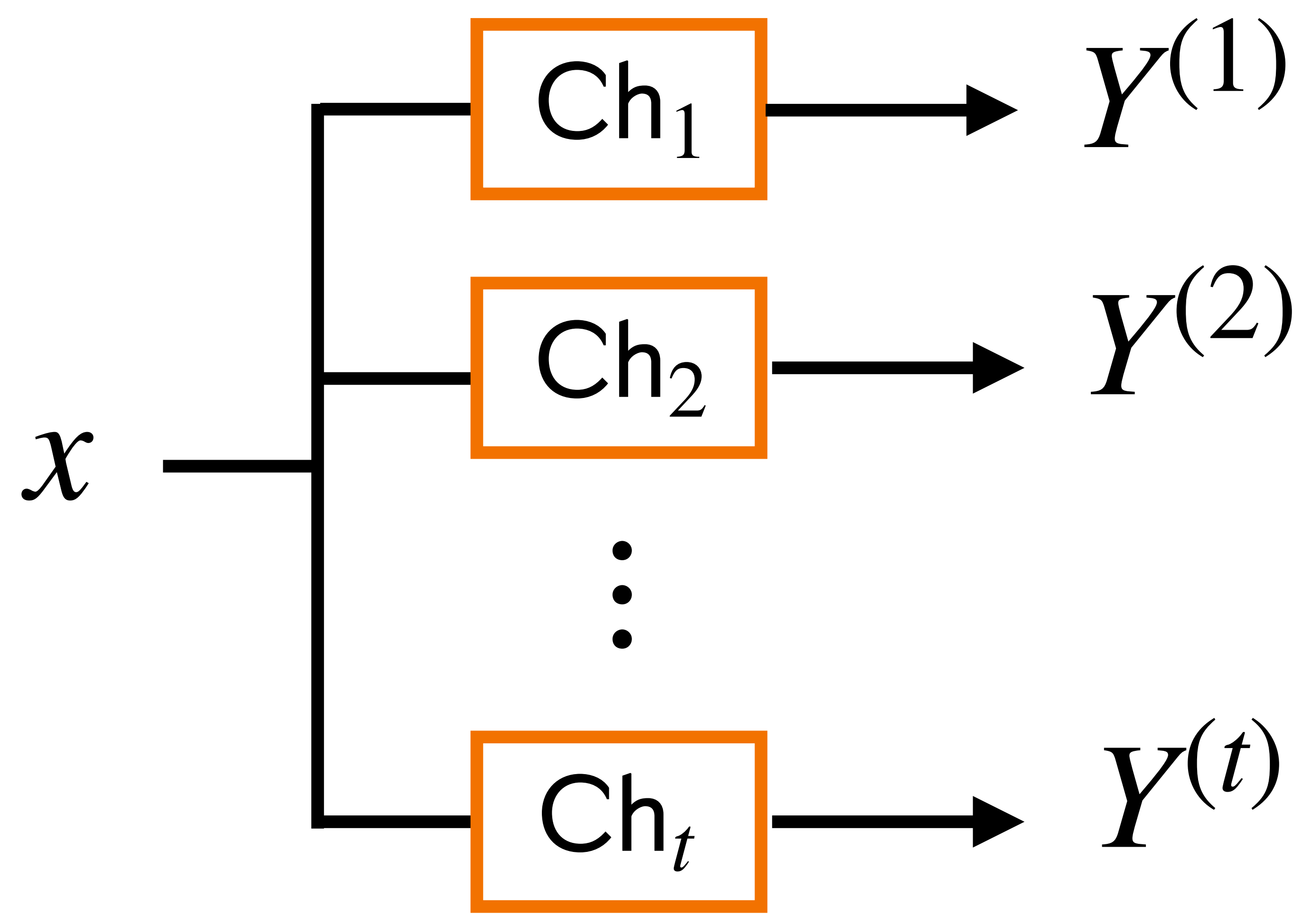
- Applies to contextual errors with unbounded context window
- Extends to multi-trace channels

Probabilistic channels with contextual synchronization errors

with Roni Con:

Under more general conditions, coding capacity = information capacity, and capacity is achieved by Markov input processes.

- Applies to contextual errors with unbounded context window
- Extends to multi-trace channels



From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process	$\Longrightarrow$	Capacity-achieving codes with many 1's in all sufficiently long substrings

From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process $\Longrightarrow$ Capacity-achieving codes with many 1's in all sufficiently long substrings

Can exploit this to obtain ***efficient*** capacity-achieving codes!

From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process $\Longrightarrow$ Capacity-achieving codes with many 1's in all sufficiently long substrings

Can exploit this to obtain ***efficient*** capacity-achieving codes!

Fix a channel with ***runlength-dependent*** deletions. Then, there exists a family of codes $(C_n)_{n\in\mathbb{N}}$ that:

- Achieves capacity on this channel;
- Is encodable in linear time and decodable in quasi-linear time in n;
- Has decoding error probability $2^{-\Omega(n)}$.

From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process $\Longrightarrow$ Capacity-achieving codes with many 1's in all sufficiently long substrings

Can exploit this to obtain ***efficient*** capacity-achieving codes!

Fix a channel with ***runlength-dependent*** deletions. Then, there exists a family of codes $(C_n)_{n\in\mathbb{N}}$ that:

- Achieves capacity on this channel;
- Is encodable in linear time and decodable in quasi-linear time in n;
- Has decoding error probability $2^{-\Omega(n)}$.

With more effort, can extend this result to the multi-trace setting

From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process	$\Longrightarrow$	Capacity-achieving codes with many 1's in all sufficiently long substrings

[Pernice, Li, Wootters 2022], originally for channels with iid deletions.

From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process	$\Longrightarrow$	Capacity-achieving codes with many 1's in all sufficiently long substrings

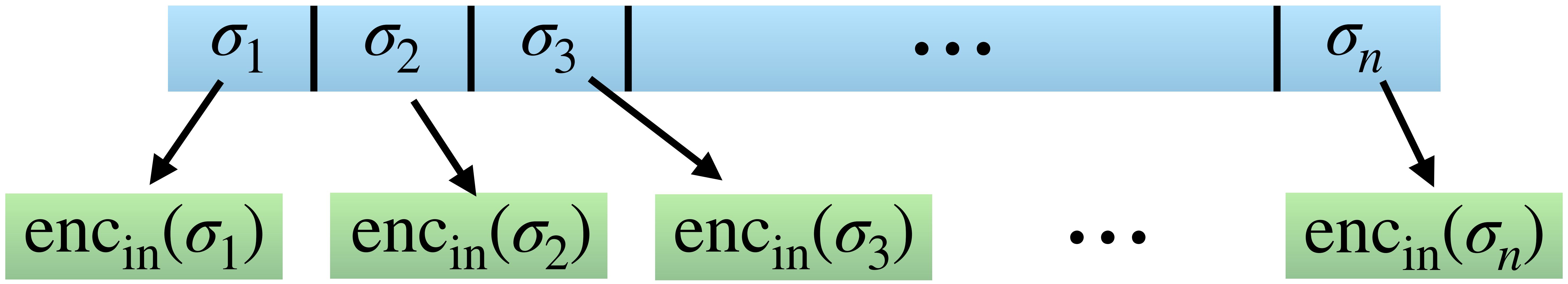
[Pernice, Li, Wootters 2022], originally for channels with iid deletions.

High-rate outer code over large alphabet correcting adversarial edit errors

From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process $\Longrightarrow$ Capacity-achieving codes with many 1's in all sufficiently long substrings

[Pernice, Li, Wootters 2022], originally for channels with iid deletions.



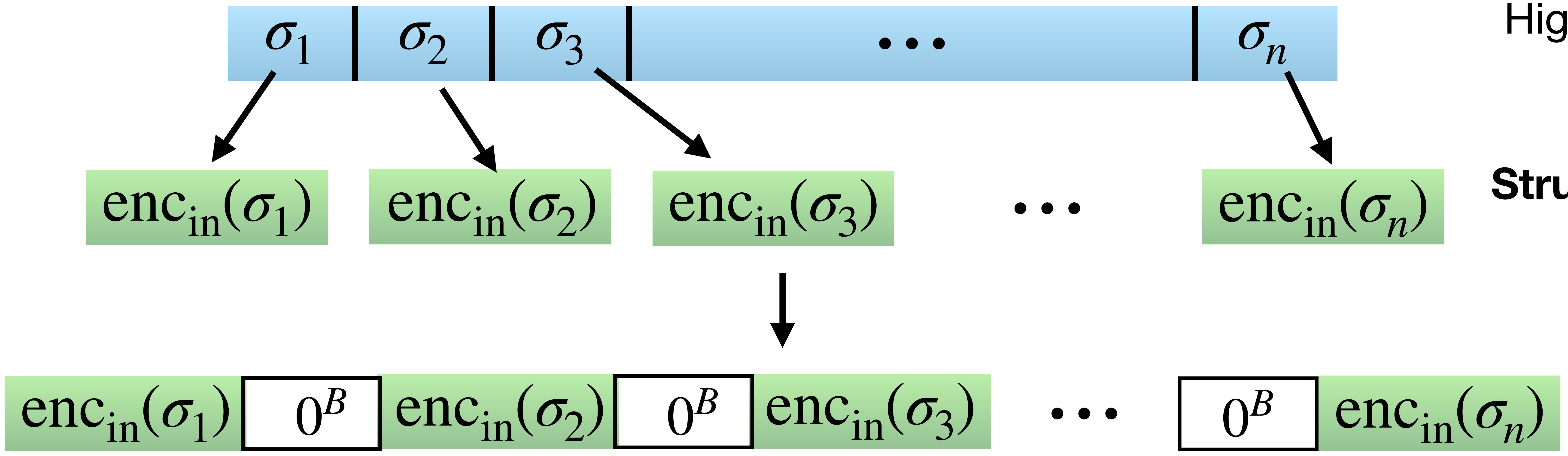
High-rate outer code over large alphabet correcting adversarial edit errors

Structured capacity-achieving inner code (can brute-force)

From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process $\Longrightarrow$ Capacity-achieving codes with many 1's in all sufficiently long substrings

[Pernice, Li, Wootters 2022], originally for channels with iid deletions.



High-rate outer code over large alphabet correcting adversarial edit errors

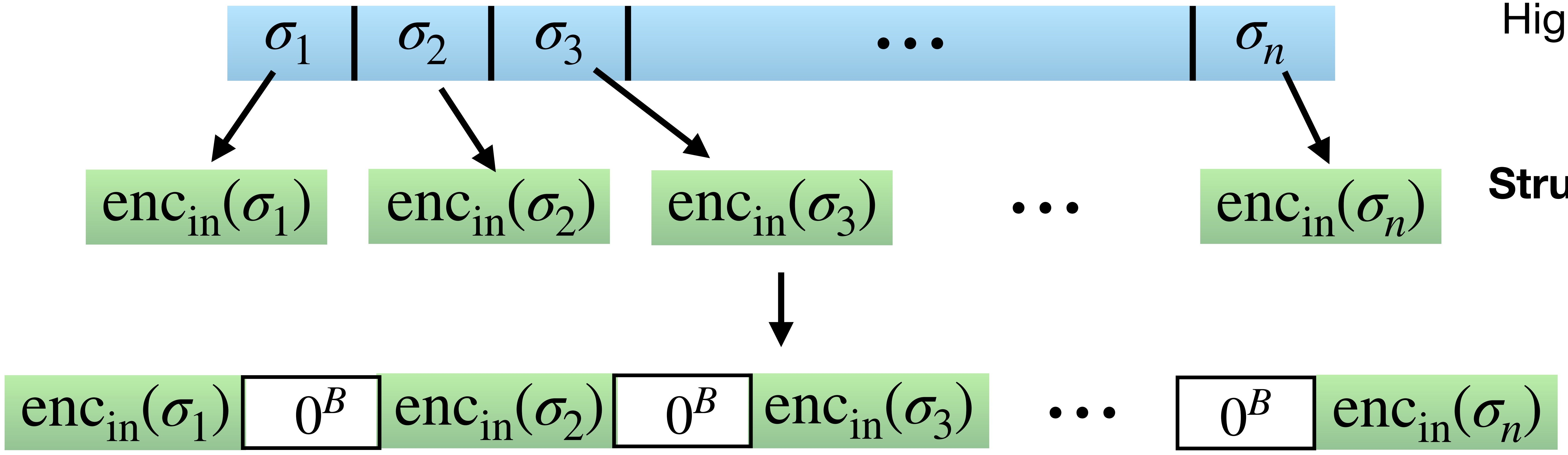
Structured capacity-achieving inner code (can brute-force)

Add markers between inner codewords

From capacity theorems to efficient optimal codes (sometimes)

Capacity achieved by Markov input process $\Longrightarrow$ Capacity-achieving codes with many 1's in all sufficiently long substrings

[Pernice, Li, Wootters 2022], originally for channels with iid deletions.



High-rate outer code over large alphabet correcting adversarial edit errors

Structured capacity-achieving inner code (can brute-force)

Add markers between inner codewords

concatenation + marker-based techniques are robust to changes in error model!

Combinatorial contextual deletions *after* long runs

with Yuan-Pon Chen, Olgica Milenkovic, Jin Sima

Recall:

We still haven't explored the effect of long runs on surrounding bits in DNA storage.

Combinatorial contextual deletions *after* long runs

with Yuan-Pon Chen, Olgica Milenkovic, Jin Sima

Recall:

We still haven't explored the effect of long runs on surrounding bits in DNA storage.

Want simple theoretical model that captures effect of long runs on surrounding bits.

Combinatorial contextual deletions *after* long runs

with Yuan-Pon Chen, Olgica Milenkovic, Jin Sima

Recall:

We still haven't explored the effect of long runs on surrounding bits in DNA storage.

Want simple theoretical model that captures effect of long runs on surrounding bits.

- Budget of t deletions;
- Given a threshold k, deletions can only occur *immediately after* runs of length $\geq k$.

Combinatorial contextual deletions *after* long runs

with Yuan-Pon Chen, Olgica Milenkovic, Jin Sima

Recall:

We still haven't explored the effect of long runs on surrounding bits in DNA storage.

Want simple theoretical model that captures effect of long runs on surrounding bits.

- Budget of t deletions;
- Given a threshold k, deletions can only occur *immediately after* runs of length $\geq k$.

$k = 2$ 0 0 **1** 1 **0** 1 1 1 **0** 1 1 **0** 0 0

Combinatorial contextual deletions *after* long runs

with Yuan-Pon Chen, Olgica Milenkovic, Jin Sima

Recall:

We still haven't explored the effect of long runs on surrounding bits in DNA storage.

Want simple theoretical model that captures effect of long runs on surrounding bits.

- Budget of t deletions;
- Given a threshold k, deletions can only occur *immediately after* runs of length $\geq k$.

$k = 2$ 0 0 **1** 1 **0** 1 1 1 **0** 1 1 **0** 0 0

t-deletion-correcting codes work here, but can we do better by exploiting contextuality?

1 contextual deletion vs. 1 arbitrary deletion

A binary code correcting 1 arbitrary deletion requires $\approx \log n$ bits of redundancy.

1 contextual deletion vs. 1 arbitrary deletion

A binary code correcting 1 arbitrary deletion requires $\approx \log n$ bits of redundancy.

How about a binary code correcting 1 **contextual** deletion with threshold $k > 1$?

1 contextual deletion vs. 1 arbitrary deletion

A binary code correcting 1 arbitrary deletion requires $\approx \log n$ bits of redundancy.

How about a binary code correcting 1 **contextual** deletion with threshold $k > 1$?

- If $k > \log n$, then $O(1)$ bits of redundancy suffice (runlength-limited coding).

1 contextual deletion vs. 1 arbitrary deletion

A binary code correcting 1 arbitrary deletion requires $\approx \log n$ bits of redundancy.

How about a binary code correcting 1 **contextual** deletion with threshold $k > 1$?

- If $k > \log n$, then $O(1)$ bits of redundancy suffice (runlength-limited coding).
- If $k = C \log n$ with $C \in (0,1)$, then $\approx 2(1 - C)\log n$ bits of redundancy suffice.

1 contextual deletion vs. 1 arbitrary deletion

A binary code correcting 1 arbitrary deletion requires $\approx \log n$ bits of redundancy.

How about a binary code correcting 1 **contextual** deletion with threshold $k > 1$?

- If $k > \log n$, then $O(1)$ bits of redundancy suffice (runlength-limited coding).
- If $k = C \log n$ with $C \in (0,1)$, then $\approx 2(1 - C)\log n$ bits of redundancy suffice.
  - When $C > 1/2$, this strictly improves on redundancy for 1 arbitrary deletion!

1 contextual deletion vs. 1 arbitrary deletion

A binary code correcting 1 arbitrary deletion requires $\approx \log n$ bits of redundancy.

How about a binary code correcting 1 **contextual** deletion with threshold $k > 1$?

- If $k > \log n$, then $O(1)$ bits of redundancy suffice (runlength-limited coding).

- If $k = C \log n$ with $C \in (0,1)$, then $\approx 2(1 - C)\log n$ bits of redundancy suffice.
  - When $C > 1/2$, this strictly improves on redundancy for 1 arbitrary deletion!

Why?

Most strings have roughly $n/2^k = n^{1-C}$ runs of length at least k.

So, there are roughly n^{1-C} locations for the contextual deletion.

1 contextual deletion vs. 1 arbitrary deletion

A binary code correcting 1 arbitrary deletion requires $\approx \log n$ bits of redundancy.

How about a binary code correcting 1 **contextual** deletion with threshold $k > 1$?

- If $k > \log n$, then $O(1)$ bits of redundancy suffice (runlength-limited coding).
- If $k = C \log n$ with $C \in (0,1)$, then $\approx 2(1 - C)\log n$ bits of redundancy suffice.
  - When $C > 1/2$, this strictly improves on redundancy for 1 arbitrary deletion!

Why?

Most strings have roughly $n/2^k = n^{1-C}$ runs of length at least k.

So, there are roughly n^{1-C} locations for the contextual deletion.

Can actually achieve this redundancy with efficient encoding/decoding!

Wrapping up

Wrapping up

- **Synchronization errors in DNA storage are contextual.**
    - For example, more errors in and around longer runs.

Wrapping up

How to capture important properties while keeping the model simple?

Capture other properties of real-world systems?

- **Synchronization errors in DNA storage are contextual.**
  - For example, more errors in and around longer runs.

Wrapping up

How to capture important properties while keeping the model simple?

Capture other properties of real-world systems?

- **Synchronization errors in DNA storage are contextual.**
    - For example, more errors in and around longer runs.

- **Capacity theorems hold for a large class of channels with contextual synchronization errors.**
    - Sometimes can use this to get efficient capacity-achieving codes.

Wrapping up

How to capture important properties while keeping the model simple?

Capture other properties of real-world systems?

- **Synchronization errors in DNA storage are contextual.**
    - For example, more errors in and around longer runs.

- **Capacity theorems hold for a large class of channels with contextual synchronization errors.**
    - Sometimes can use this to get efficient capacity-achieving codes.

Capacity bounds?

Wrapping up

How to capture important properties while keeping the model simple?

Capture other properties of real-world systems?

- **Synchronization errors in DNA storage are contextual.**
  - For example, more errors in and around longer runs.
- **Capacity theorems hold for a large class of channels with contextual synchronization errors.**
  - Sometimes can use this to get efficient capacity-achieving codes.

Capacity bounds?

- **Can significantly lower redundancy by exploiting contextuality of errors.**
  - Deletions only right after sufficiently long runs vs. arbitrary deletions.

Wrapping up

How to capture important properties while keeping the model simple?

Capture other properties of real-world systems?

- **Synchronization errors in DNA storage are contextual.**
  - For example, more errors in and around longer runs.

- **Capacity theorems hold for a large class of channels with contextual synchronization errors.**
  - Sometimes can use this to get efficient capacity-achieving codes.

Capacity bounds?

- **Can significantly lower redundancy by exploiting contextuality of errors.**
  - Deletions only right after sufficiently long runs vs. arbitrary deletions.

Better bounds on redundancy?

Lower thresholds?