Internal Quasiperiod Queries

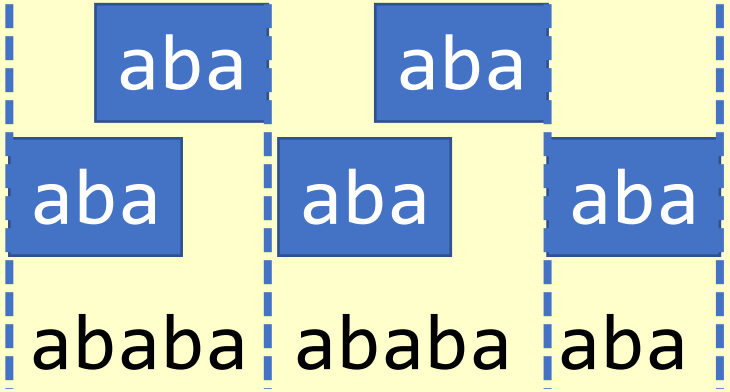
Maxime Crochemore
Costas S. Iliopoulos
Jakub Radoszewski
Wojciech Rytter
Juliusz Straszyński
Tomasz Waleń
Wiktor Zuba

Periodicity

5th power of aba

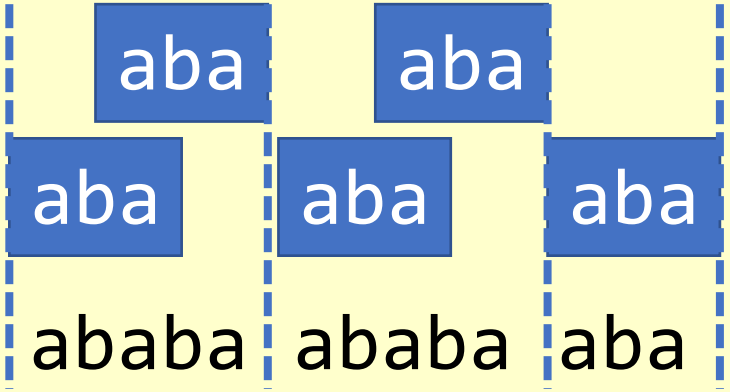
Periodic string with period aba

Quasiperiodicity



We allow overlaps!

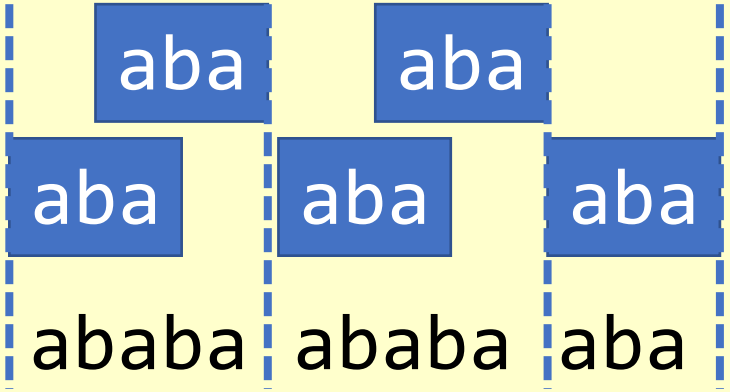
Quasiperiodicity



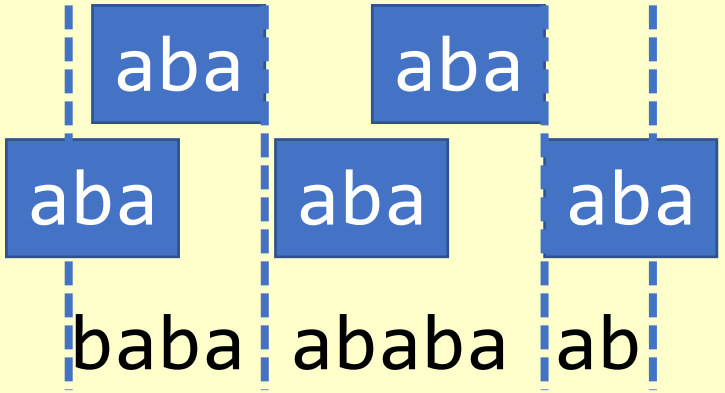
Cover – all occurrences fit in string
Therefore quasiperiod is both
prefix and suffix
i.e. a border

We allow overlaps!

Quasiperiodicity



Cover – all occurrences fit in string
Therefore quasiperiod is both
prefix and suffix
i.e. a border



Seed – occurrences can go
beyond string

Background

Shortest cover O(n) – Apostolico et al. 1991
Covers of prefixes O(n) – Dany Breslauer 1992
All covers O(n) – Moore and Smyth 1995

All seeds O(n log n) – Iliopoulos et al. 1996
All seeds O(n) – Kociumaka et al. 2012

Variants

- approximate covers
- lambda-covers
- approximate seeds
- covers in other models of computation
- etc.

Internal covers

Input:

Text

List of queries (i, j)

Output:

shortest cover of Text[i..j]

OR

all covers of Text[i..j]

Our results

Input:

Text

List of queries (i, j)

Output:

shortest cover of Text[i..j] <- O(log n log log n)

OR

O(log n) offline

OR

all covers of Text[i..j] <- $O(\log n (\log \log n)^2)$

after O(n log n) preprocessing

1. O(log n) candididates

Each cover is a border

There are O(log n) primitive (non-periodic) borders

We can find them in O(log n) time per query using stringology tools

1. O(log n) candididates

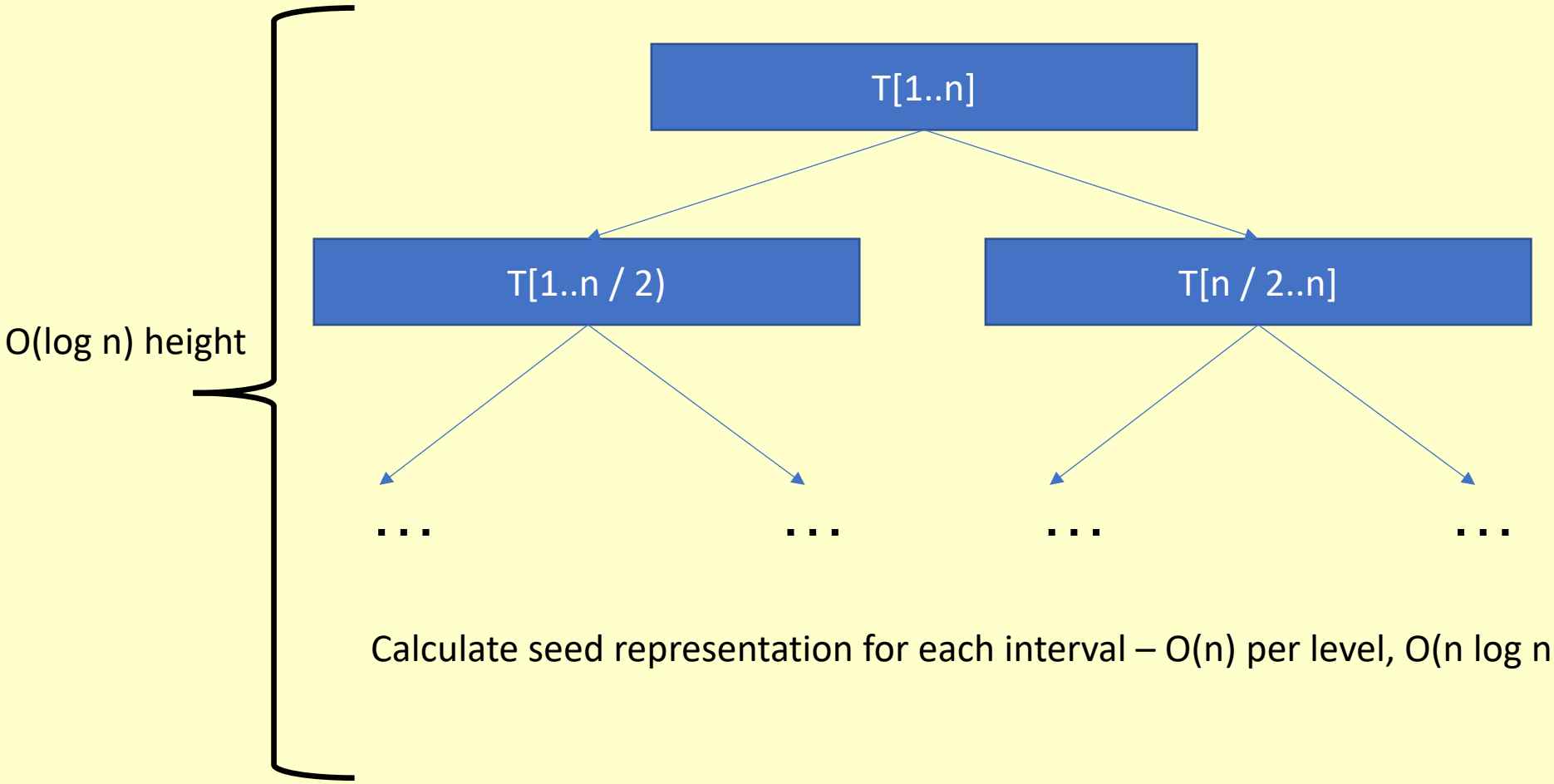
Each cover is a border

There are O(log n) primitive (non-periodic) borders

We can find them in O(log n) time per query using internal period queries

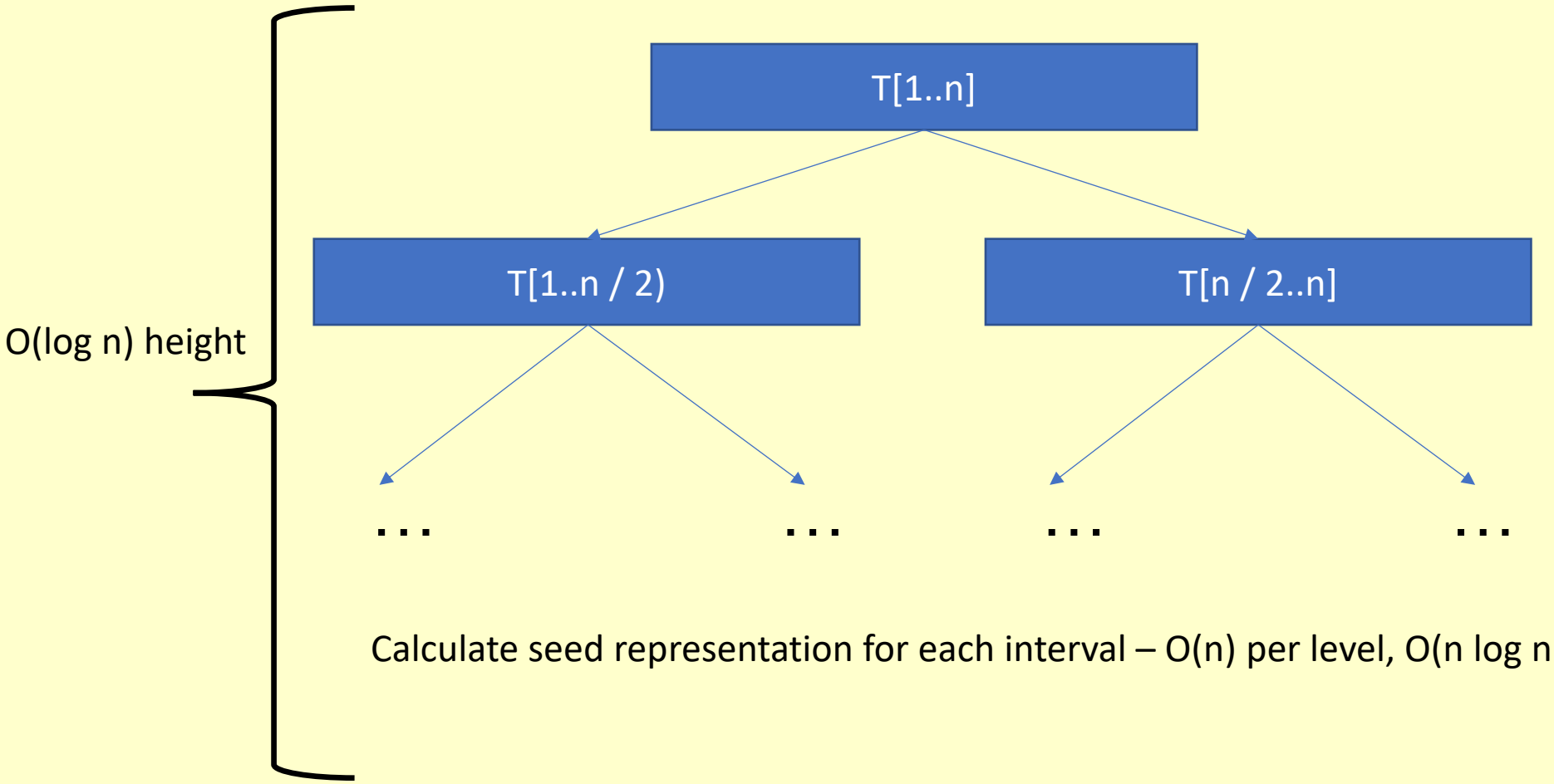
But how do we check them?

2. Divide & Conquer



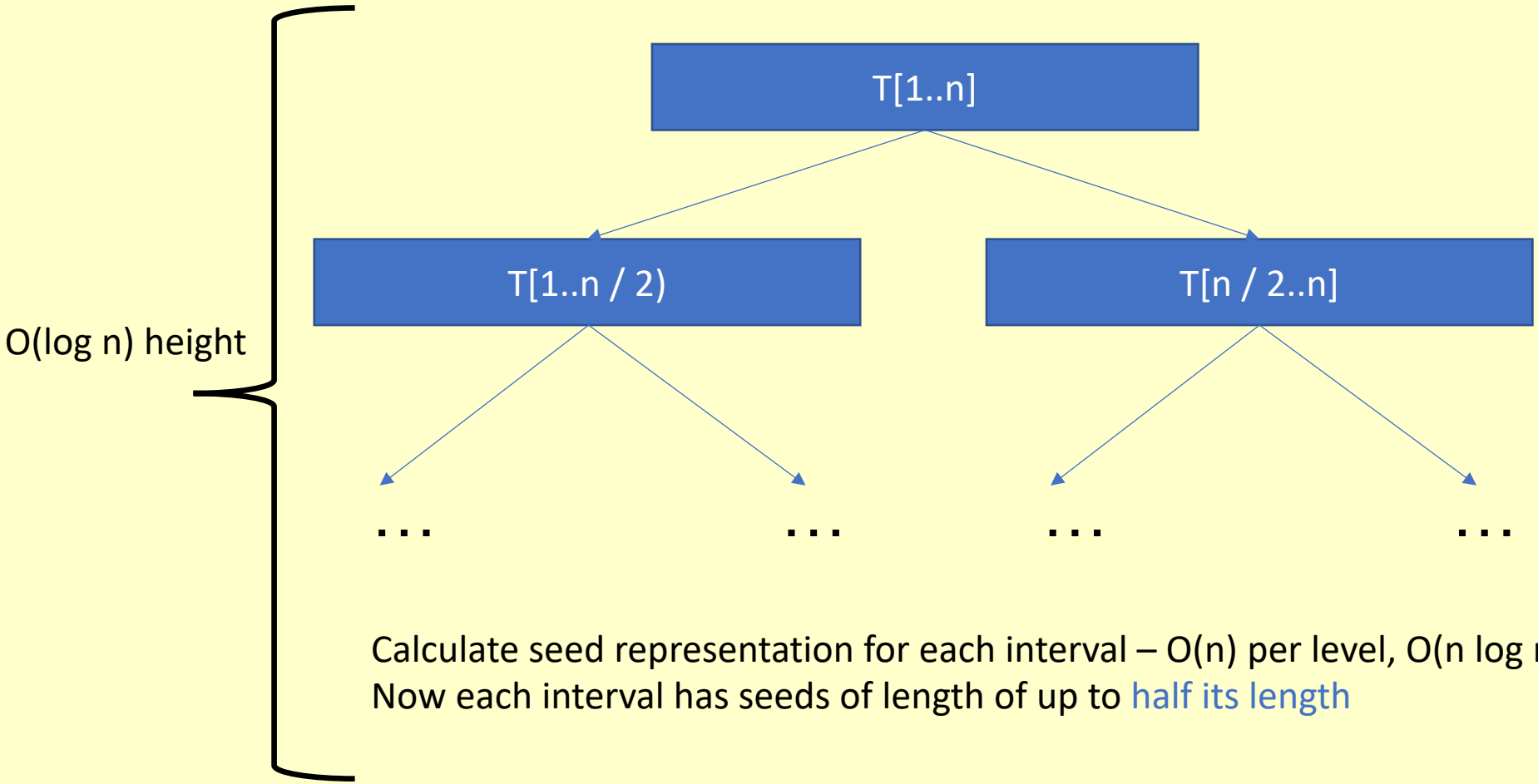
Calculate seed representation for each interval – O(n) per level, O(n log n) in total

2. Divide & Conquer



Calculate seed representation for each interval – O(n) per level, O(n log n) in total

2. Divide & Conquer



Calculate seed representation for each interval – O(n) per level, O(n log n) in total
Now each interval has seeds of length of up to half its length

2. Divide & Conquer

Now every query can be decomposed into O(log n) disjoint intervals.

Quest: check candidate if it's a cover of query's interval

1. It must be a seed of longer intervals
2. We can check if it covers short intervals with Internal Pattern Matching!

Internal Pattern Matching

Input:

Text

Queries: interval [a..b] + substring [c..d]

intervals of similar length (O(1) ratio)

Output:

O(1) occurrences

or

O(1) arithmetic progressions of occurrences

2. Divide & Conquer

Now every query can be decomposed into O(log n) disjoint intervals.

Quest: check candidate if it's a cover of query's interval

1. It must be a seed of longer intervals
2. We can check if it covers short intervals with Internal Pattern Matching!

2. Divide & Conquer

Now every query can be decomposed into O(log n) disjoint intervals.

Quest: check candidate if it's a cover of query's interval

1. It must be a seed of longer intervals
2. We can check if it covers short intervals with Internal Pattern Matching!
3. We still need to check if candidate connects neighbouring intervals well:

Being a seed doesn't imply good coverage in that area!
But we can check that with IPM

2. Divide & Conquer

Now every query can be decomposed into O(log n) disjoint intervals.

Quest: check candidate if it's a cover of query's interval

1. It must be a seed of longer intervals O(log log n) – weighted ancestor queries
2. We can check if it covers short intervals with Internal Pattern Matching O(1)
3. We still need to check if candidate connects neighbouring intervals well:

Being a seed doesn't imply good coverage in that area!
But we can check that with IPM O(1)

2. Divide & Conquer

Now every query can be decomposed into O(log n) disjoint intervals.

Quest: check candidate if it's a cover of query's interval

1. It must be a seed of longer intervals O(log log n) - weighted ancestor queries
2. We can check if it covers short intervals with Internal Pattern Matching O(1)
3. We still need to check if candidate connects neighbouring intervals well:

Being a seed doesn't imply good coverage in that area!
But we can check that with IPM O(1)
O(log n) candidates O(log n) intervals O(log log n) per interval = $O(\log^2(n)\log\log n)$
We can do better!

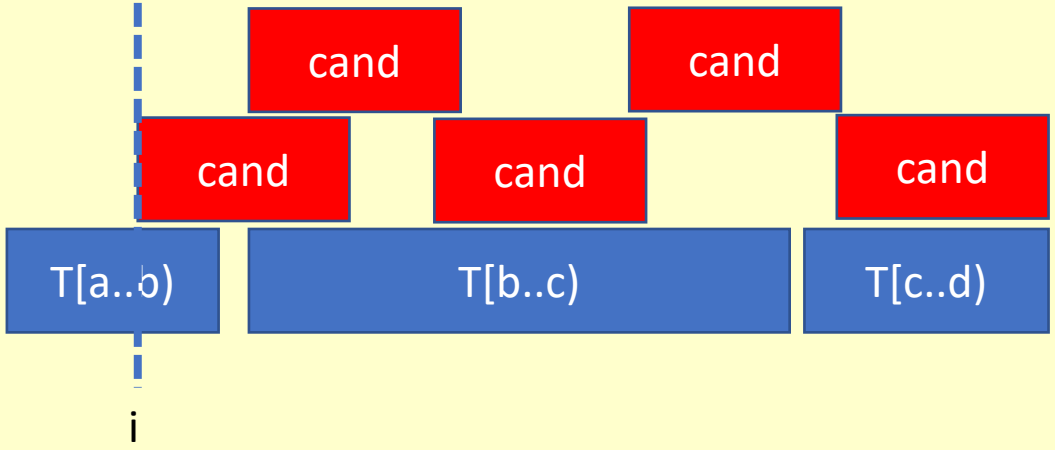
3. Refinement

Query: [i..j]
Let’s check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate



3. Refinement

Query: [i..j]
Let’s check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate

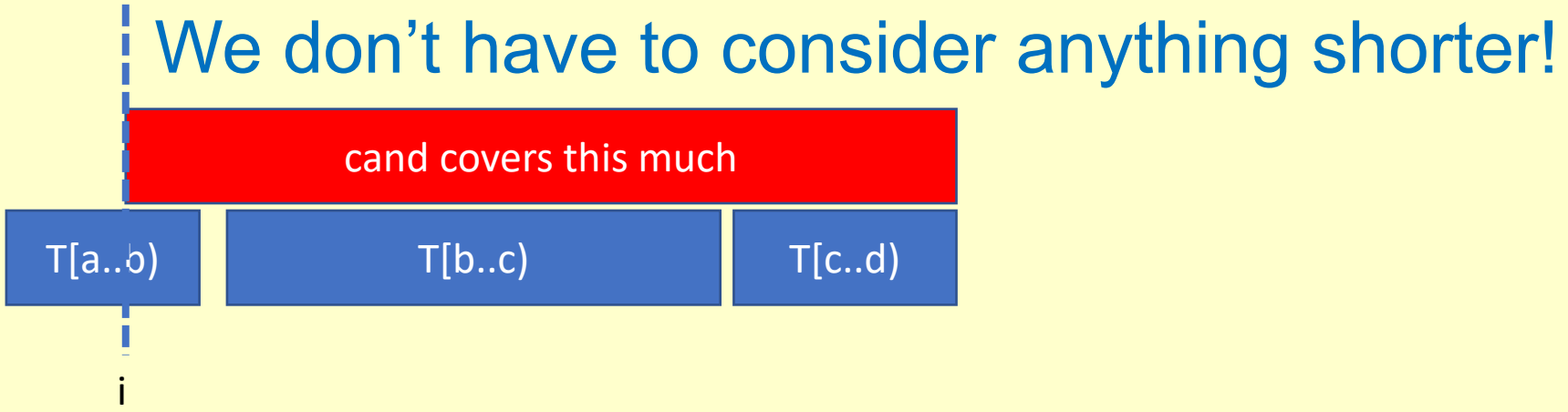
3. Refinement

Query: [i..j]
Let’s check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate



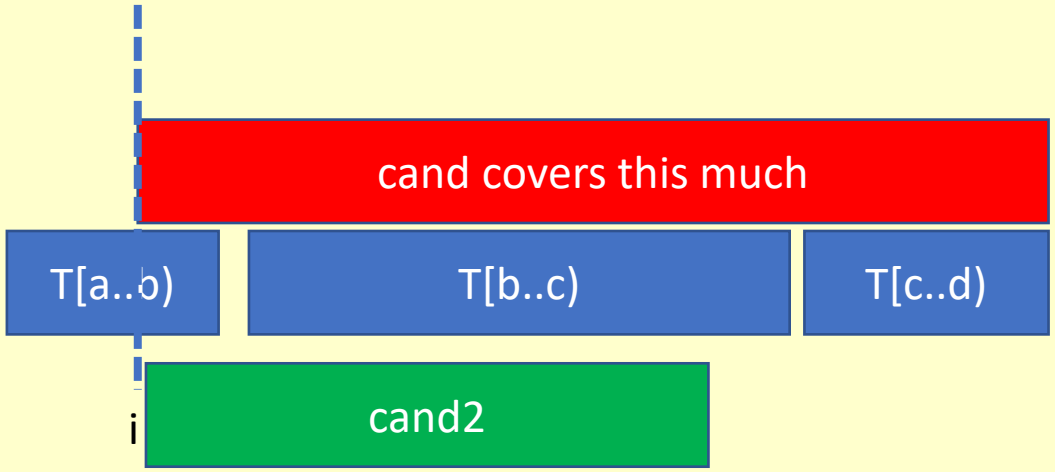
3. Refinement

Query: [i..j]
Let’s check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate



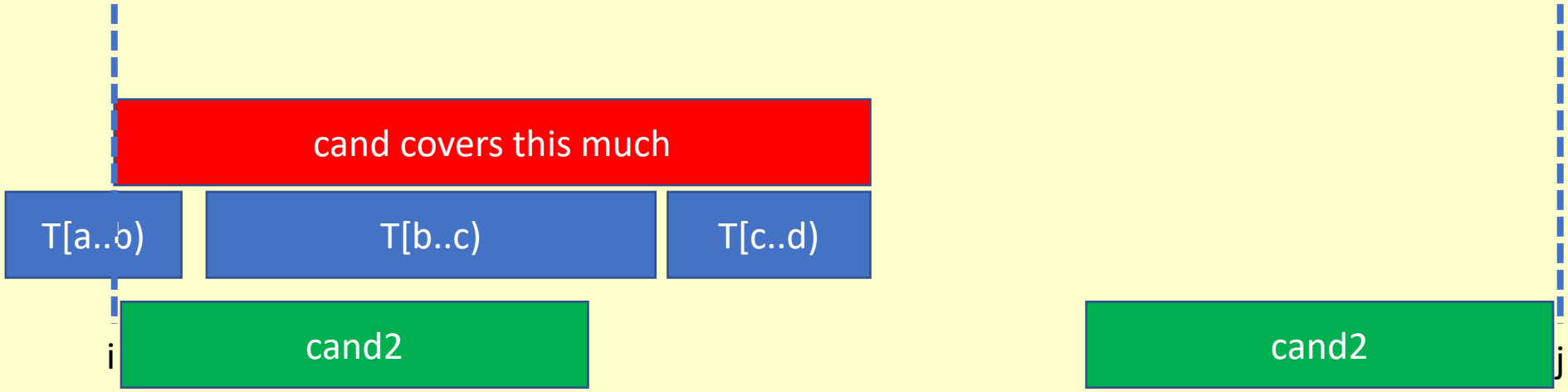
3. Refinement

Query: [i..j]
Let’s check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate



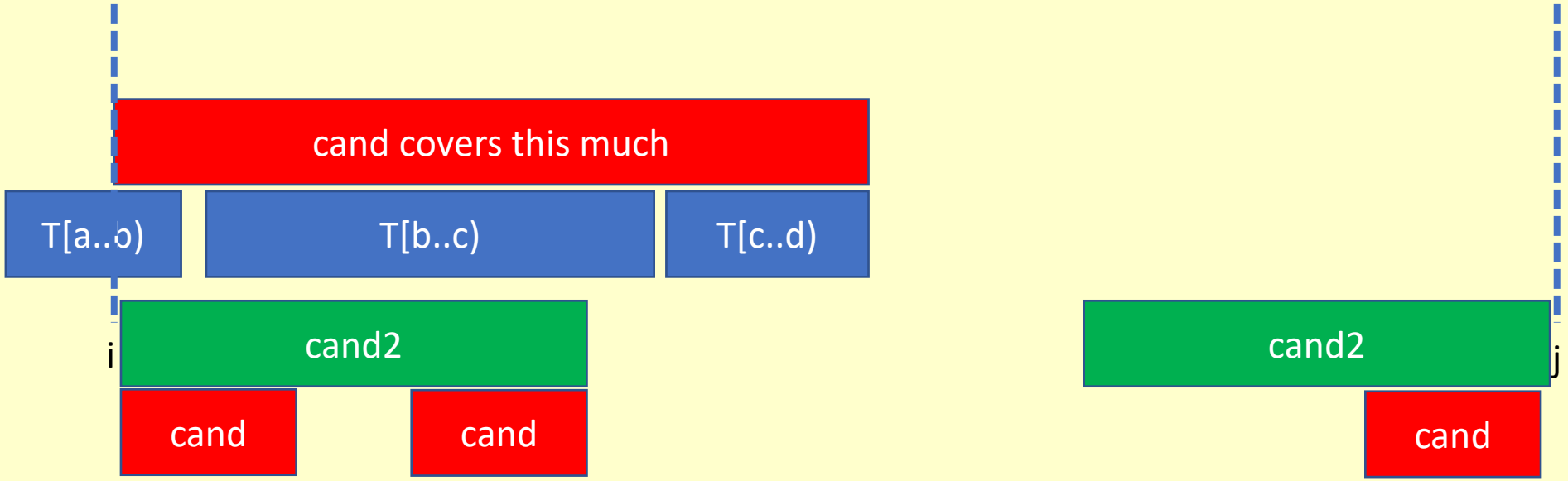
3. Refinement

Query: [i..j]
Let’s check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate



Cand also covers cand2!

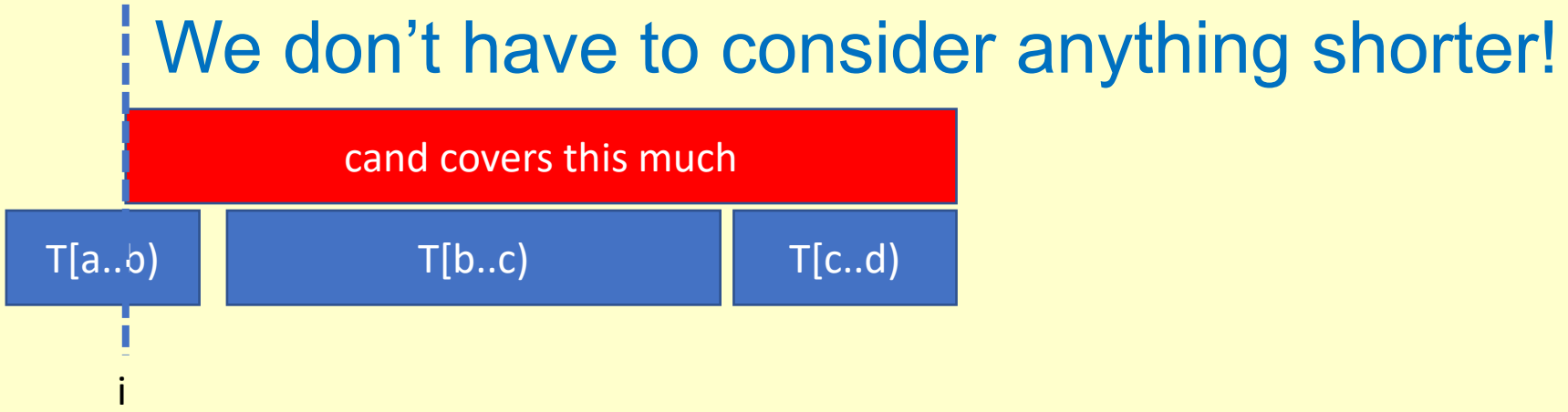
3. Refinement

Query: [i..j]
Let’s check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate



3. Refinement

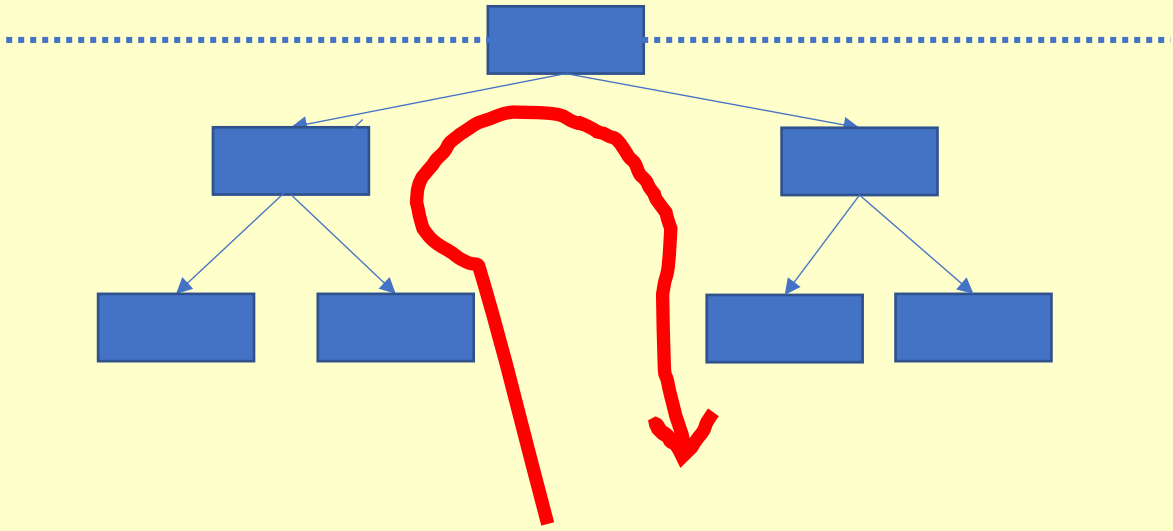
Query: [i..j]
Let’s check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate

How does interval decomposition in BST works here?



1. Start at some level
2. Go up to find some big intervals that satisfy condition
   This finds rough estimate of an answer
3. Go down to find the exact answer

Remarks:
The answer we have found (longest covered prefix)
is roughly the same size as nodes above the upper level

We can start there and never go lower than this.

3. Refinement

We visit O(log n) nodes in total for all candidates
We do each node in O(log log n) time
Hence: O(log n log log n) per query

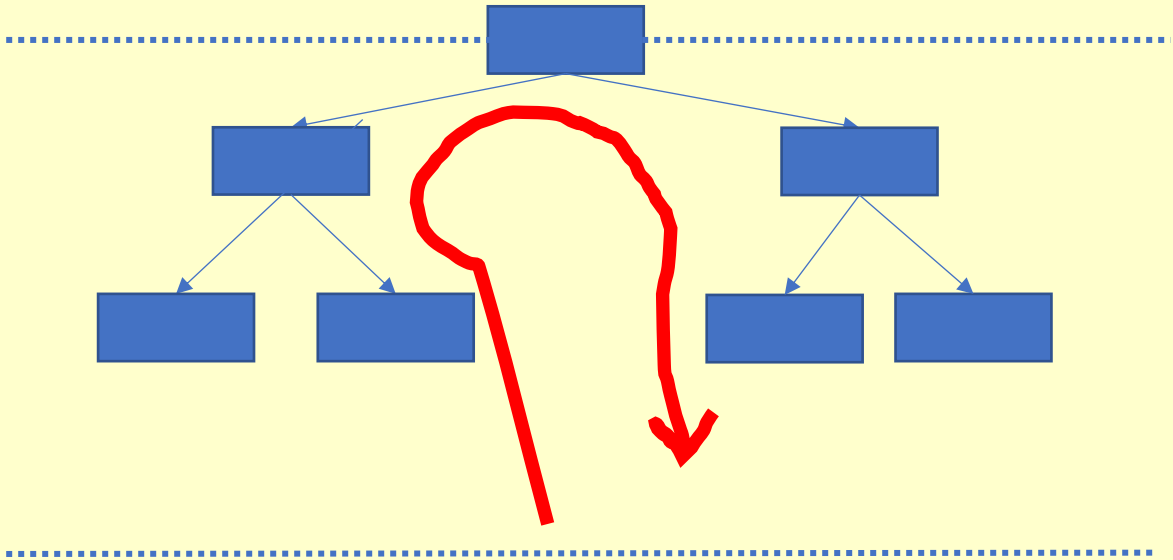
Query: [i..j]
Let's check the shortest candidate as said before.

Is it cover?

Yes -> we won

No -> find how much we can cover with our candidate

How does interval decomposition in BST works here?



1. Start at some level
2. Go up to find some big intervals that satisfy condition
   This finds rough estimate of an answer
3. Go down to find the exact answer

Remarks:
The answer we have found (longest covered prefix)
is roughly the same size as nodes above the upper level

We can start there and never go lower than this.

3. All covers

In the paper ☺

Thank you for attention!