イベント巡り 2
（Event Hopping 2）

解説：北村祐稀

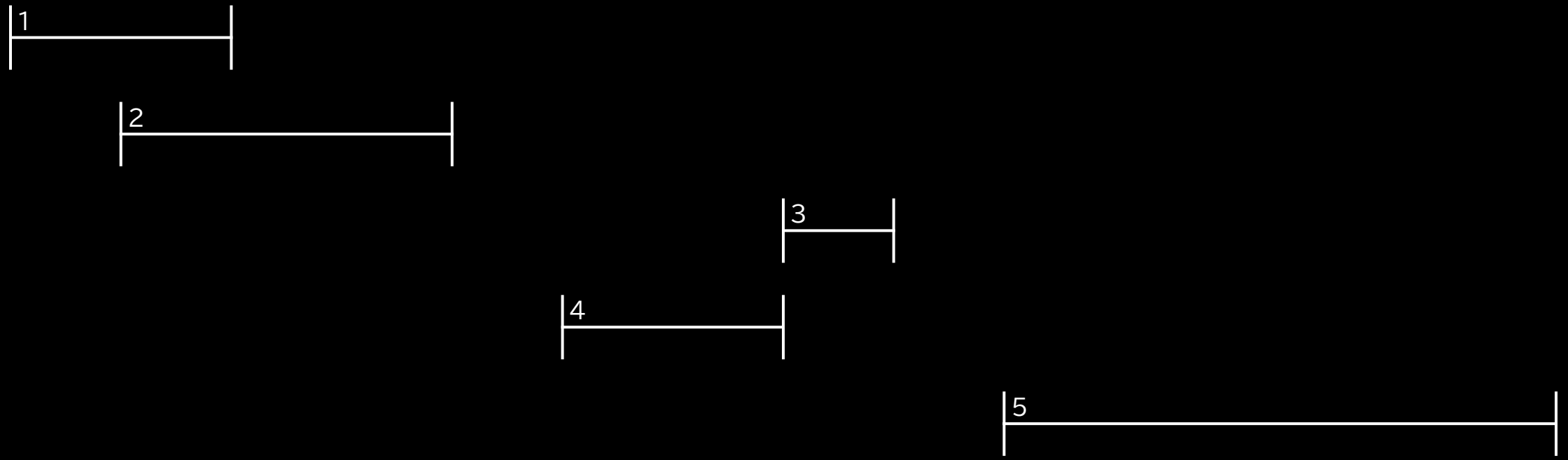
JOI 2020/2021 春合宿 Day 4

問題概要

- N 個の開区間 (L_i, R_i) が与えられる
- 重ならないように K 個の区間 $a_1, \ldots, a_K$ を選ぶ
- 辞書順最小となる数列 $(a_1, \ldots, a_K)$ を求めよ

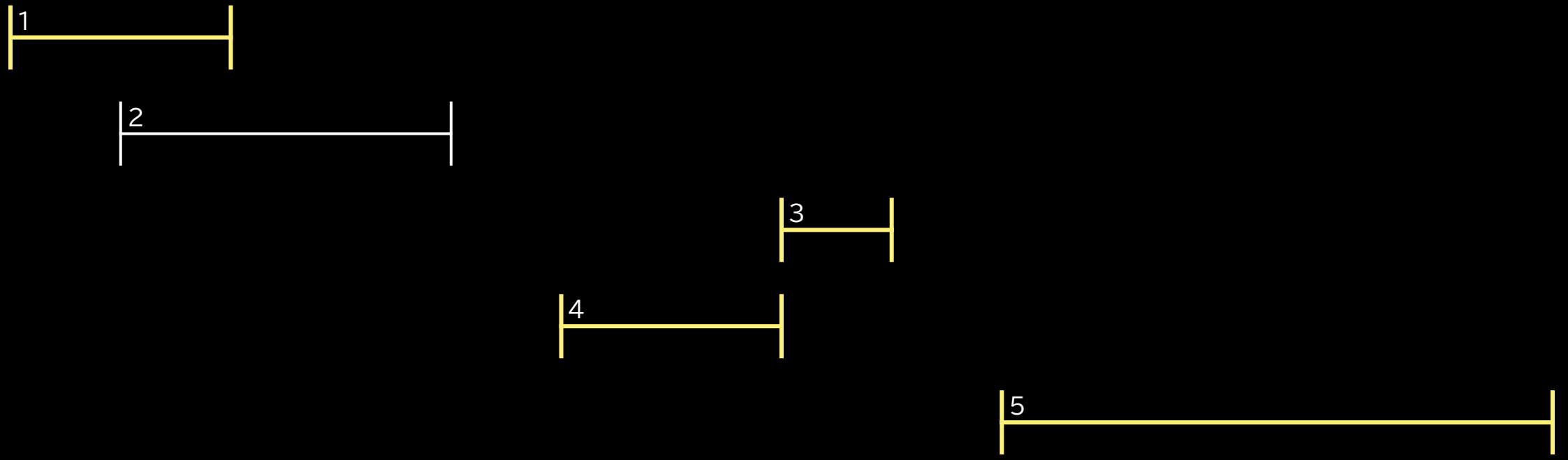
入出力例 1

$N = 5,\ K = 4$



入出力例 1

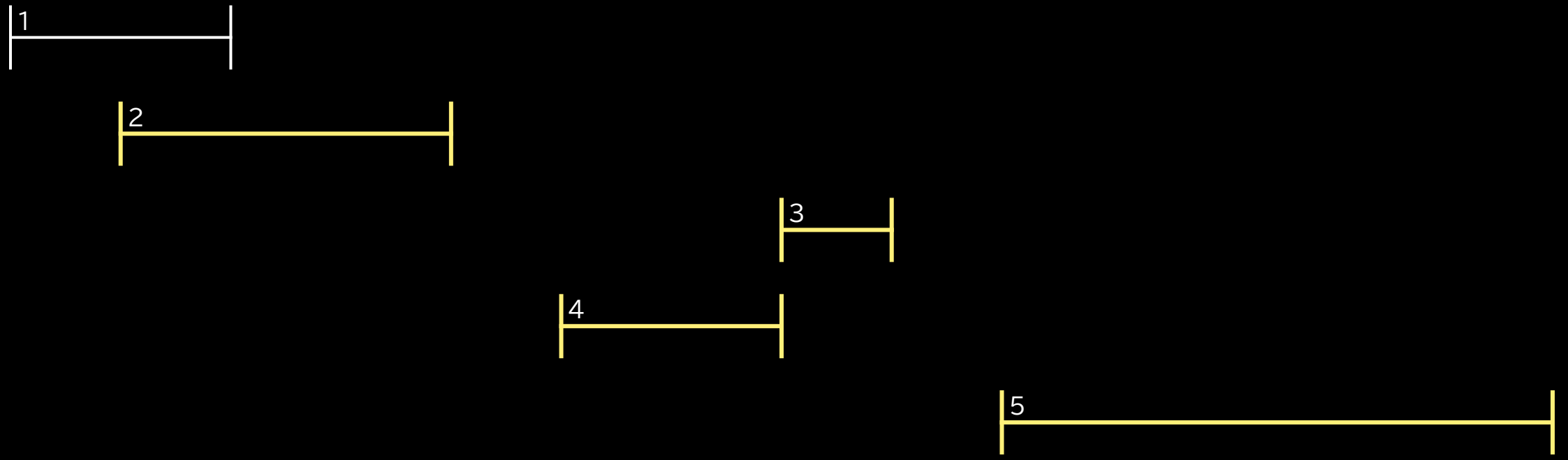
$N = 5,\ K = 4$



$(a_1, \ldots, a_K) = (1,3,4,5)$

入出力例 1

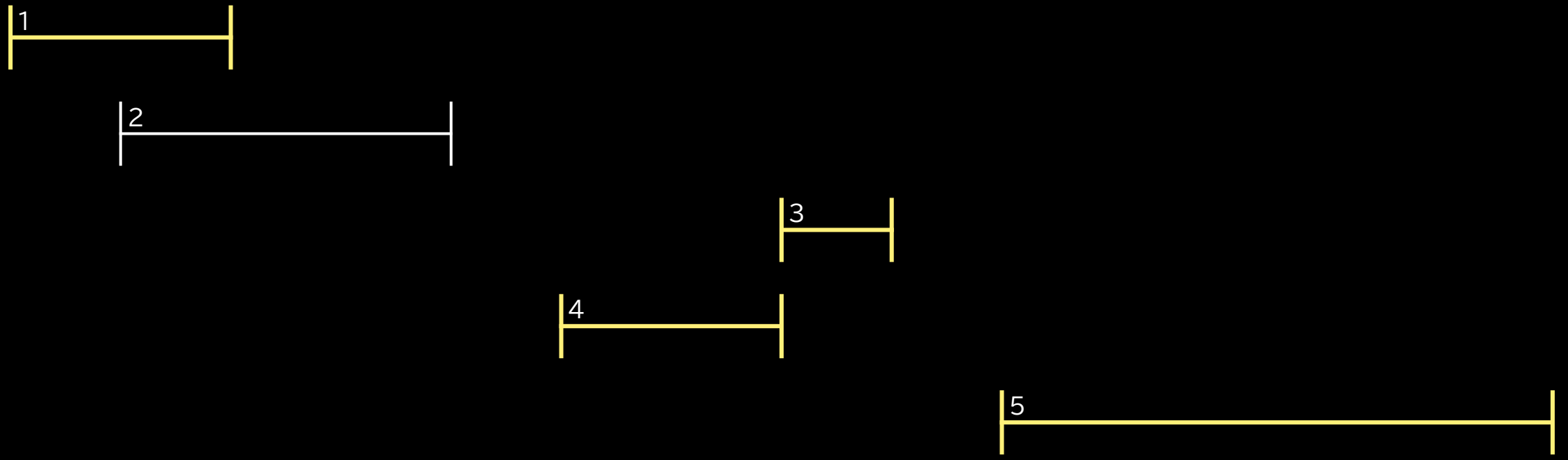
$N = 5,\ K = 4$



$(a_1, \ldots, a_K) = (2,3,4,5)$

入出力例 1

$N = 5,\ K = 4$



$(1,3,4,5) < (2,3,4,5)$ より答えは $(1,3,4,5)$

目次

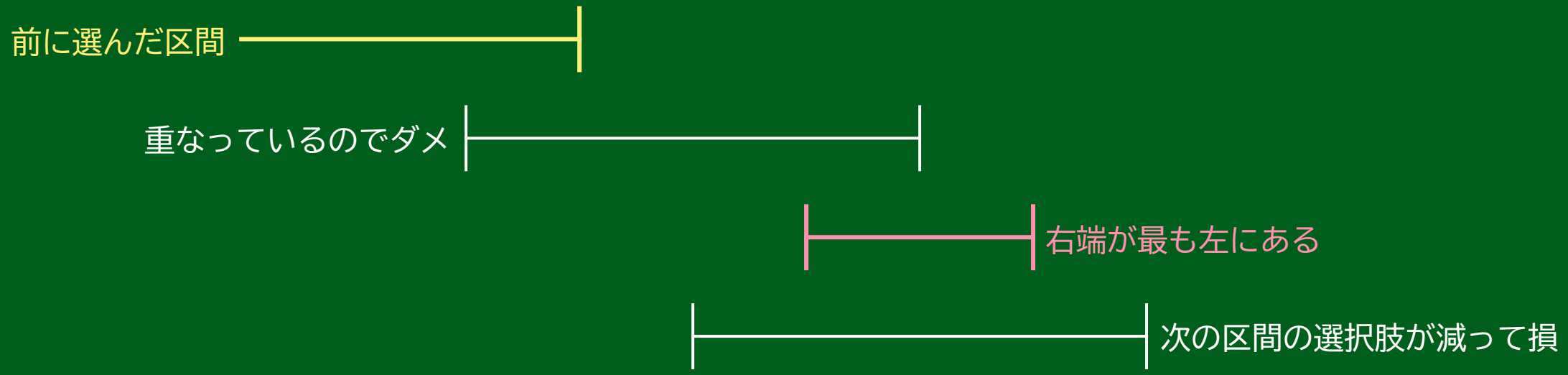
目次

小課題 1

$L_i \leqq L_{i+1}$ $(1 \leqq i \leqq N-1)$

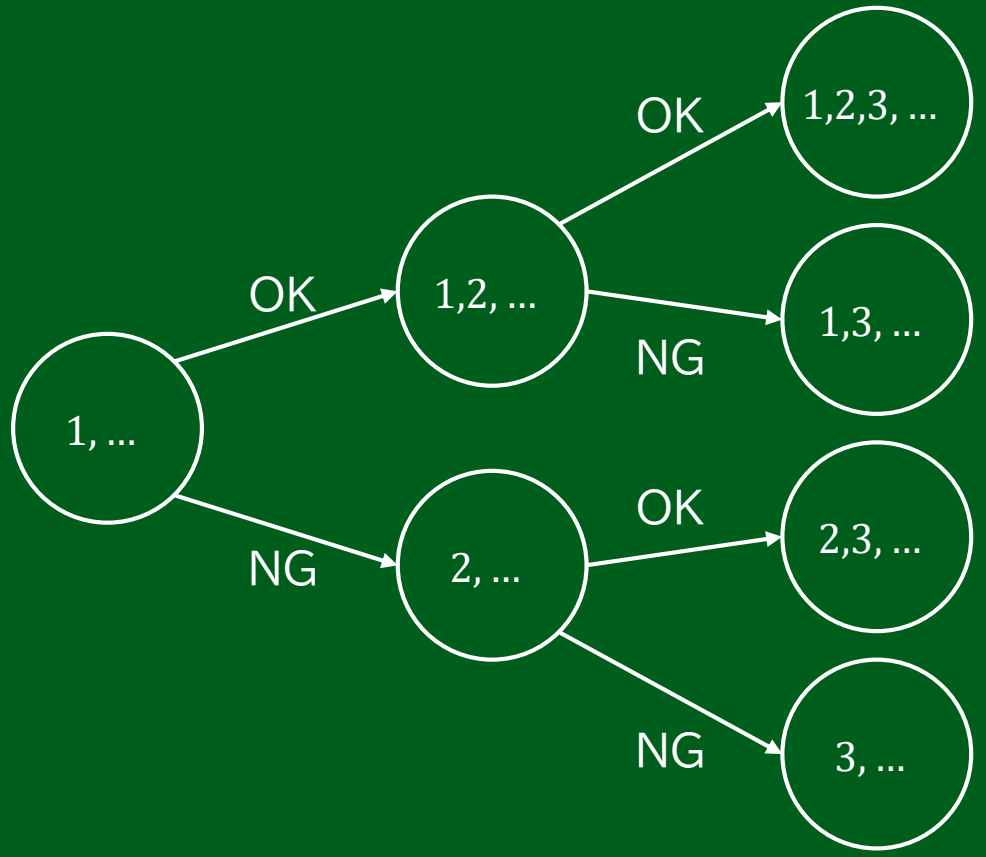
典型 重ならない区間の個数の最大化

- 貪欲法が使える
  - 右端でソートし，左から順に見て行く
  - 次に選べる区間のうち，右端が最も左の区間を選ぶ



典型 辞書順最小化

- 条件を満たす状態を維持しつつ選んでいく



1 を使ってみる　　2 を使ってみる　　3 を使ってみる

2,1,3 みたいなのも考えないと
いけないと大変だけど今回は不要

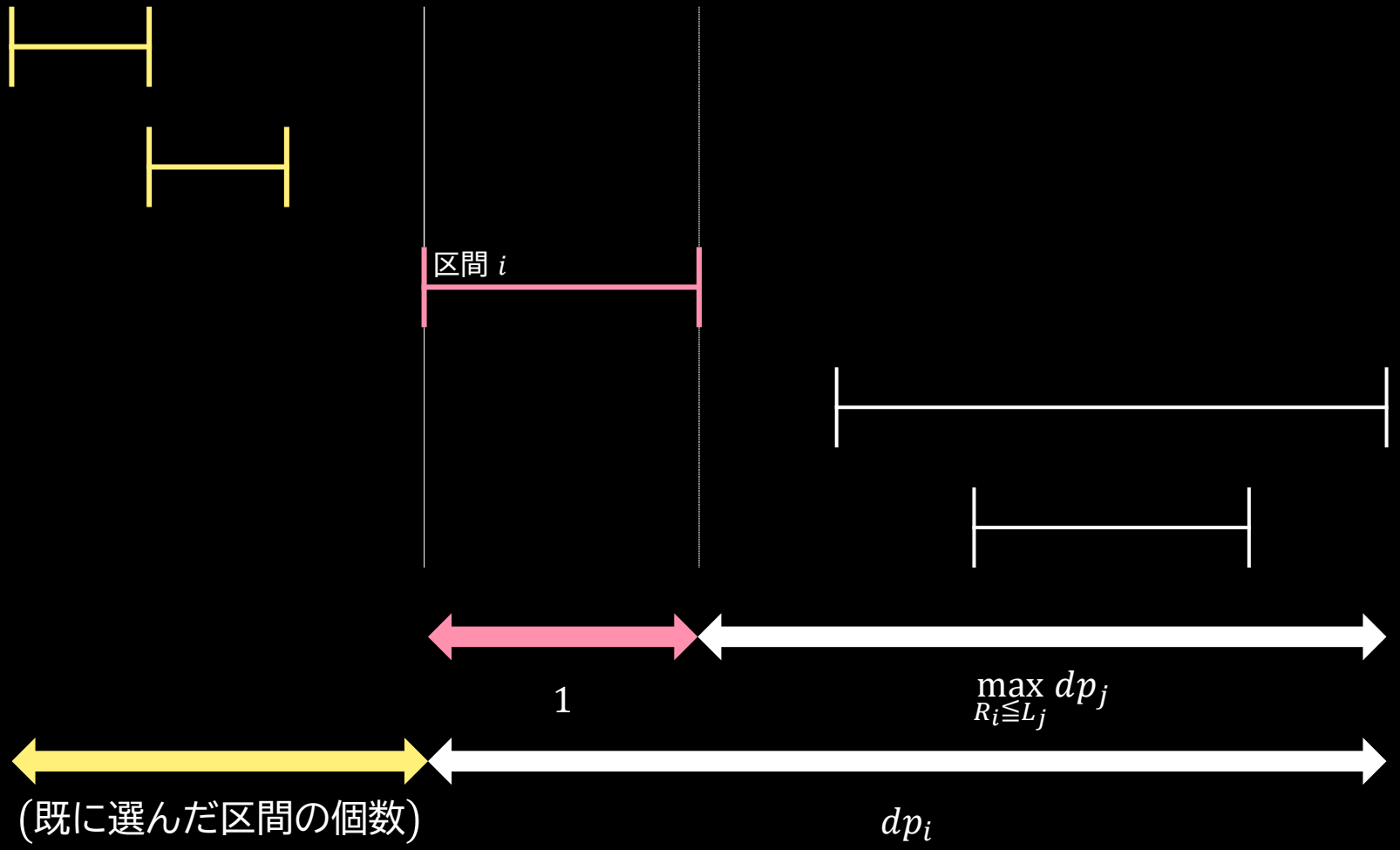
考察：方針

- ２ つの典型を組み合わせる
  - 左から順に選んでいく　→ 辞書順最小化
  - 条件を満たすか判定　　→ 重ならない区間の個数の最大化
  - (既に選んだ区間の個数) +
    (今選んだ区間とそれより右で選べる個数の最大値) $\geqq K$

考察：DP

- (今選んだ区間とそれより右で選べる個数の最大値) を DP で持っておくとよい
  - dp_i := (区間 i とそれより右で選べる個数の最大値)
  - $dp_i = 1 + \max_{R_i \leqq L_j} dp_j$
  - $R_i \leqq L_j$ な j は連続しているので,
    累積 max + 二分探索などで dp が求められる
- $O(N \log N)$ 時間

図示



実装例

```
for(int i = N - 1; i >= 0; i--){
    int j = lower_bound(L, L + N, R[i]) - L;
    dp[i] = 1 + (j < N ? dpmax[j] : 0);
    dpmax[i] = max(dp[i], i + 1 < N ? dpmax[i + 1] : 0);
}
int now = 0;
for(int i = 0; i < N && now < K; i++){
    if(now + dp[i] >= K){
        cout << i + 1 << endl;
        now++;
        i = lower_bound(L, L+N, R[i]) - L - 1;
    }
}
```

目次

目次

小課題２

$N \leqq 20$

考察

- 区間の選び方 2^N 通りを bit 全探索で試して，
  - 選んだ区間が被っていない：$O(N^2)$ で全組チェック
  - 選んだ区間の数が K 個

  な選び方のうち辞書順最小のものを出力する
- 区間 i と下から $N-1-i$ 番目の bit を対応付けると辞書順最小の判定が楽
- $O(2^N N^2)$ で間に合う
  - 区間をソートしておいて判定パートを軽くすると $O(2^N N)$

実装例

```
for(int i = (1 << N) - 1; i >= 0; i--){
    if(__builtin_popcount(i) != K) continue;
    bool ok = true;
    for(int j = 0; j < N; j++){
        if(~(i >> N - 1 - j) & 1) continue;
        for(int k = 0; k < N; k++){
            if(~(i >> N - 1 - k) & 1 || j == k) continue;
            ok &= (R[j] <= L[k] || R[k] <= L[j]);
        }
    }
    if(ok){
        for(int j = 0; j < N; j++){
        if((i >> N - 1 - j) & 1) cout << j + 1 << endl;
    }
        return 0;
    }
}
cout << -1 << endl;
```

目次

目次

小課題３

$N \leqq 3\,000$

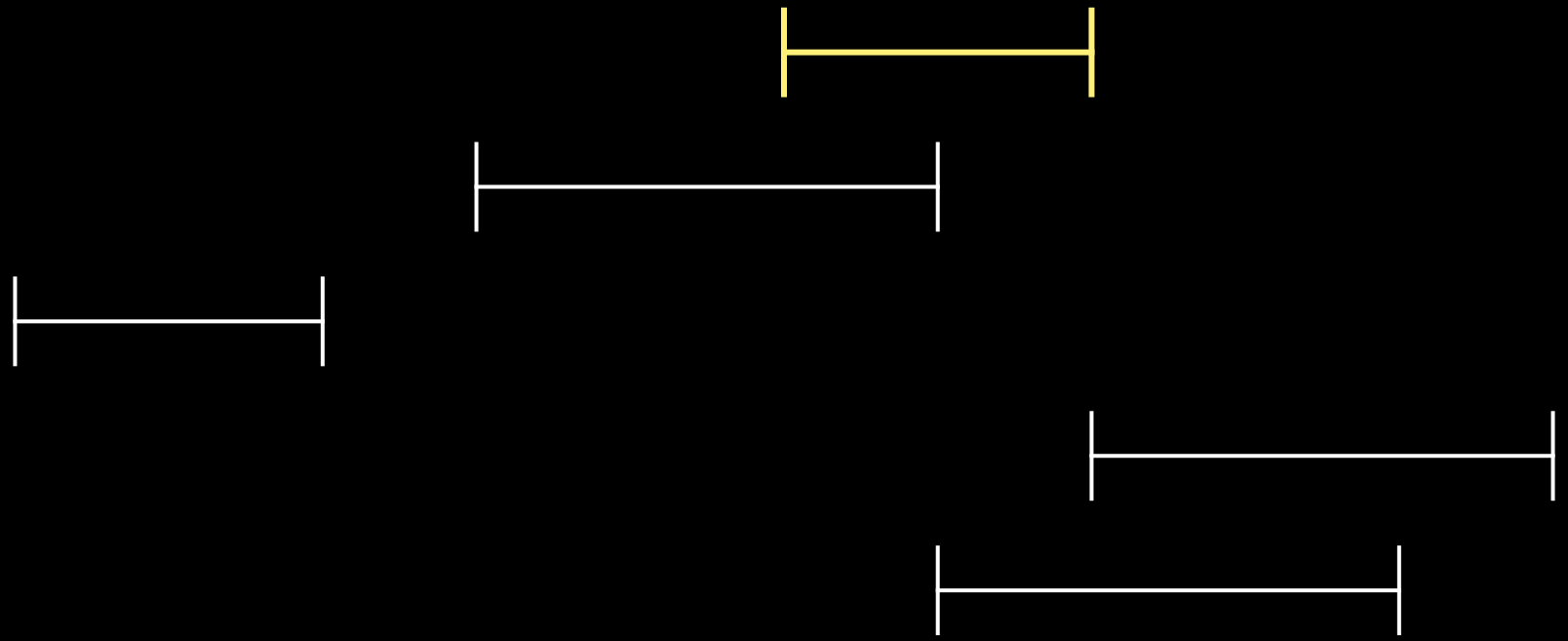
考察：方針

- ２ つの典型を組み合わせる
  - インデックスが小さいものから順に選んでいく
    → 辞書順最小化
  - 条件を満たすか判定
    → 重ならない区間の個数の最大化
- どうやって条件を満たすか判定しようか？

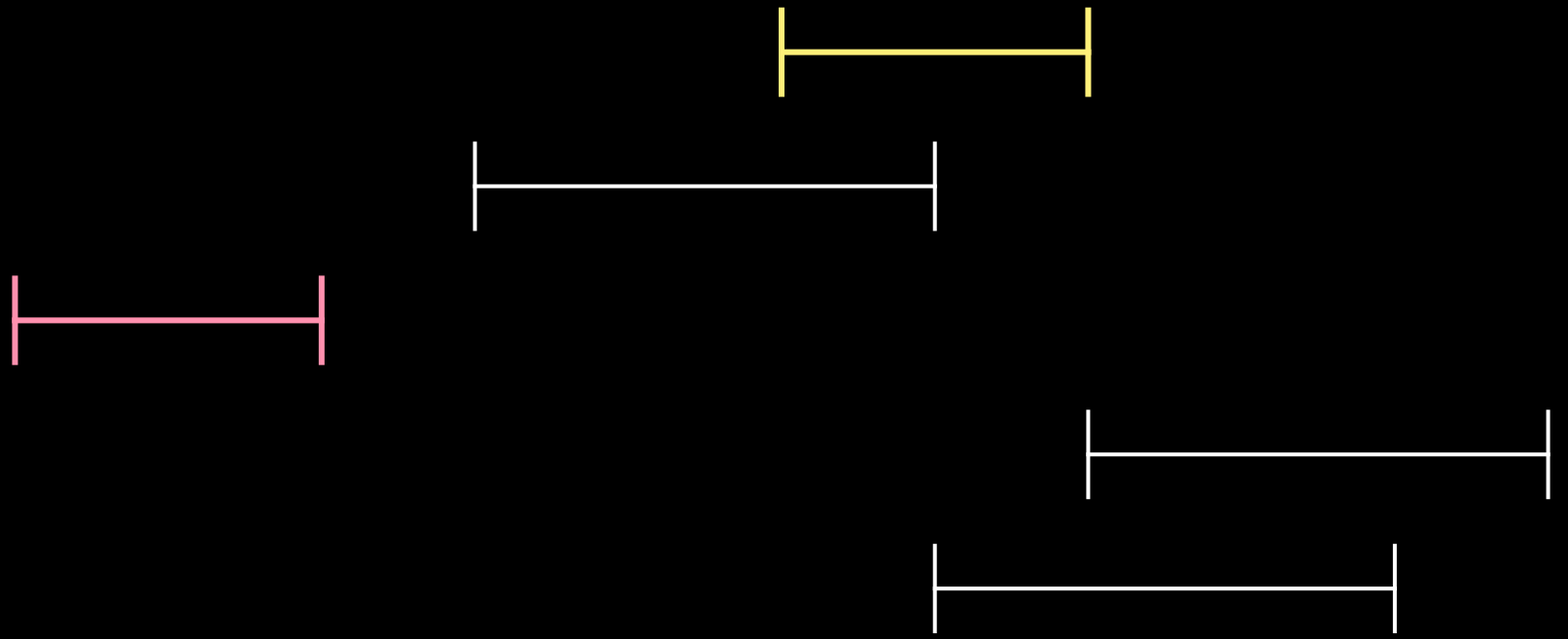
考察：判定方法

- 「重ならない区間の個数の最大化」を普通にやる
- ただし，選ばれている区間は必ず使うようにする

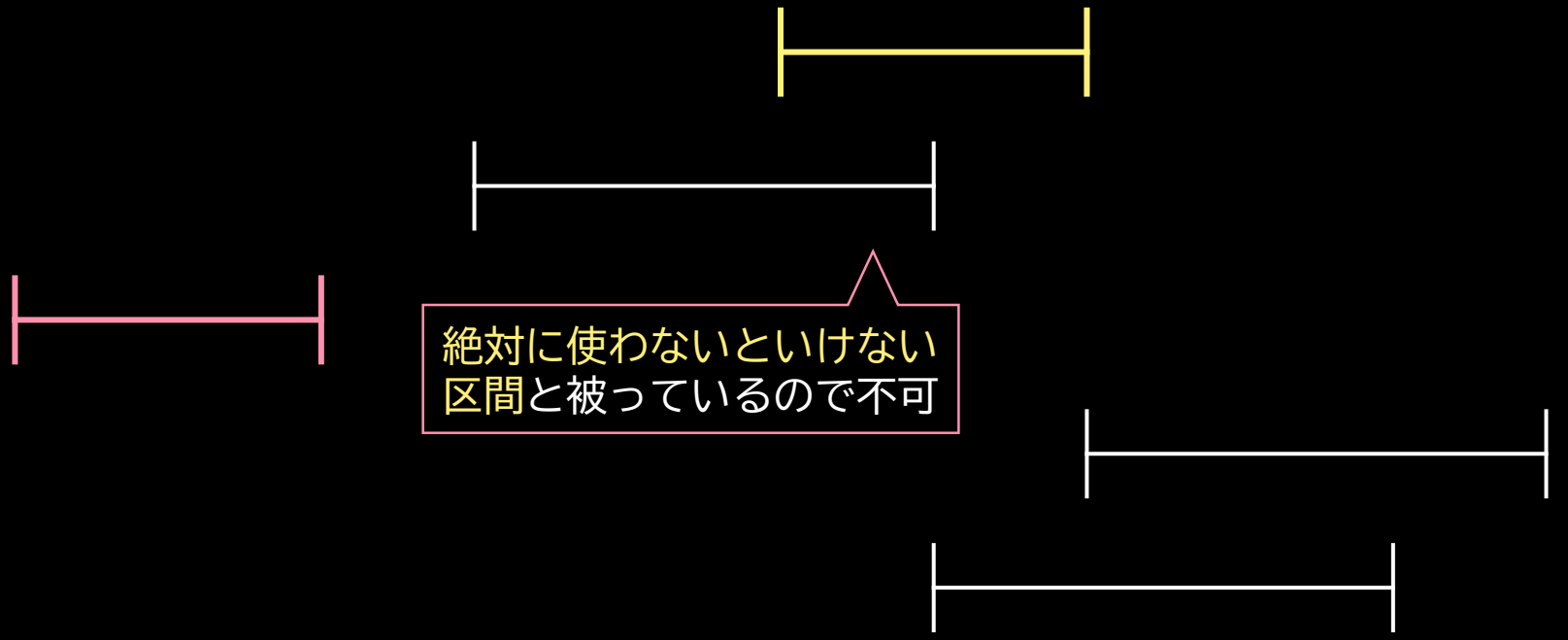
图示



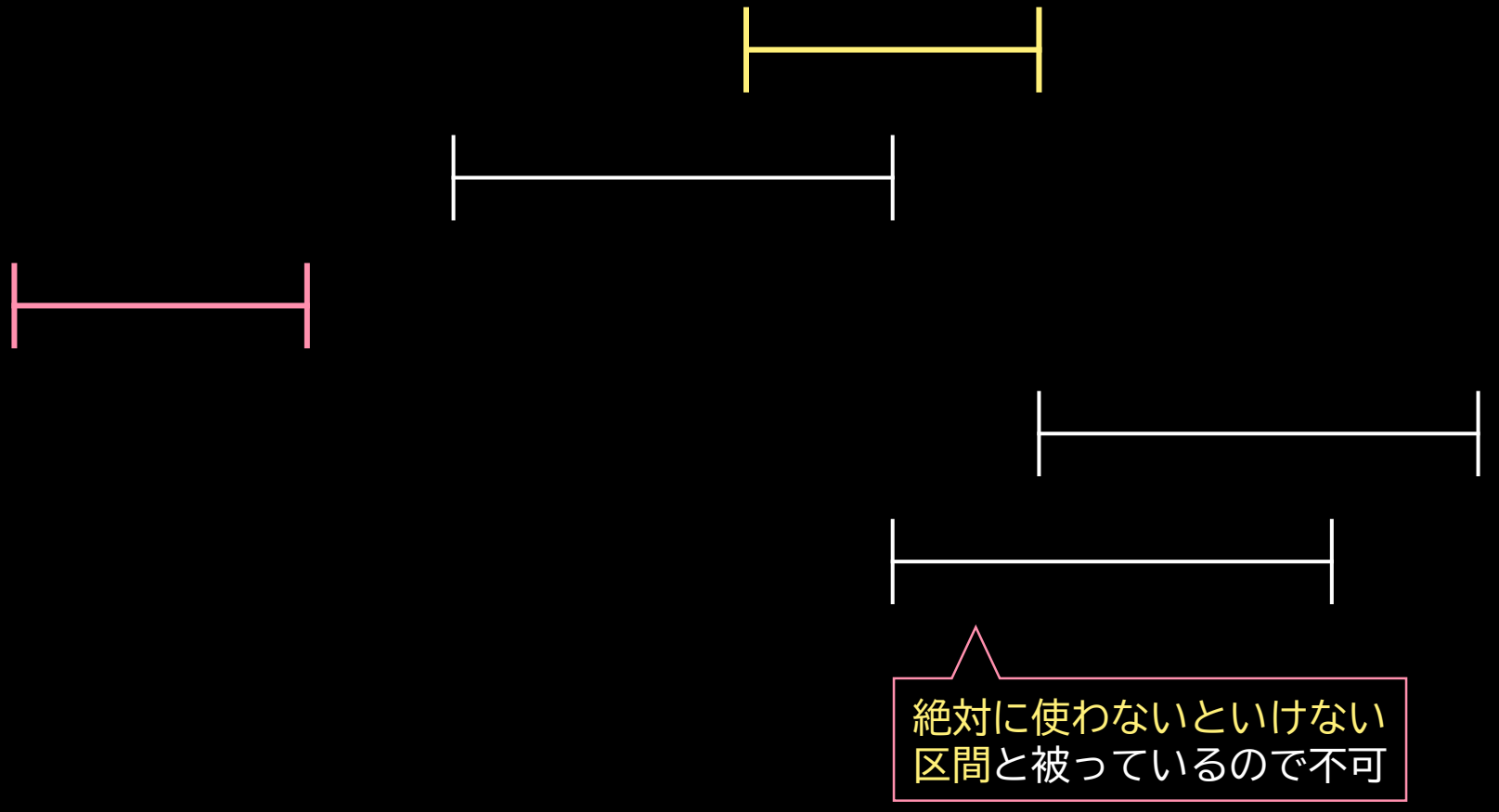
图示



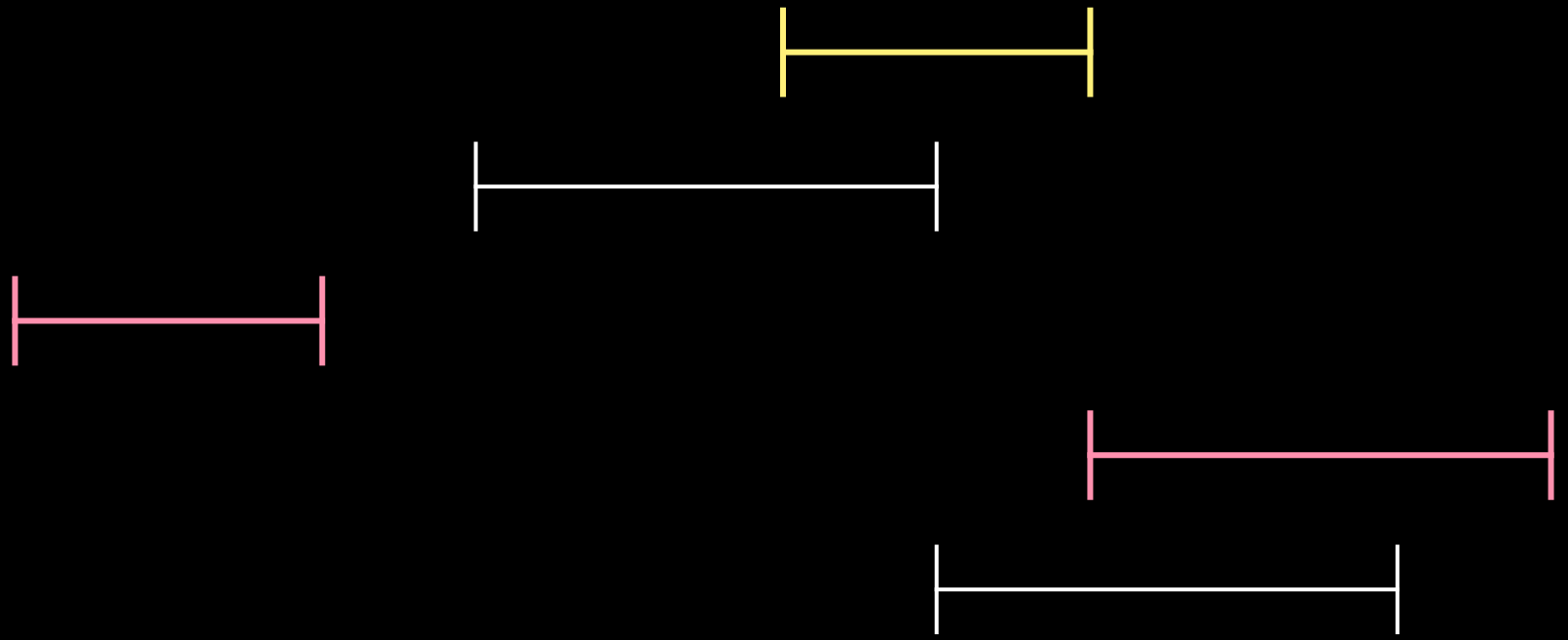
図示



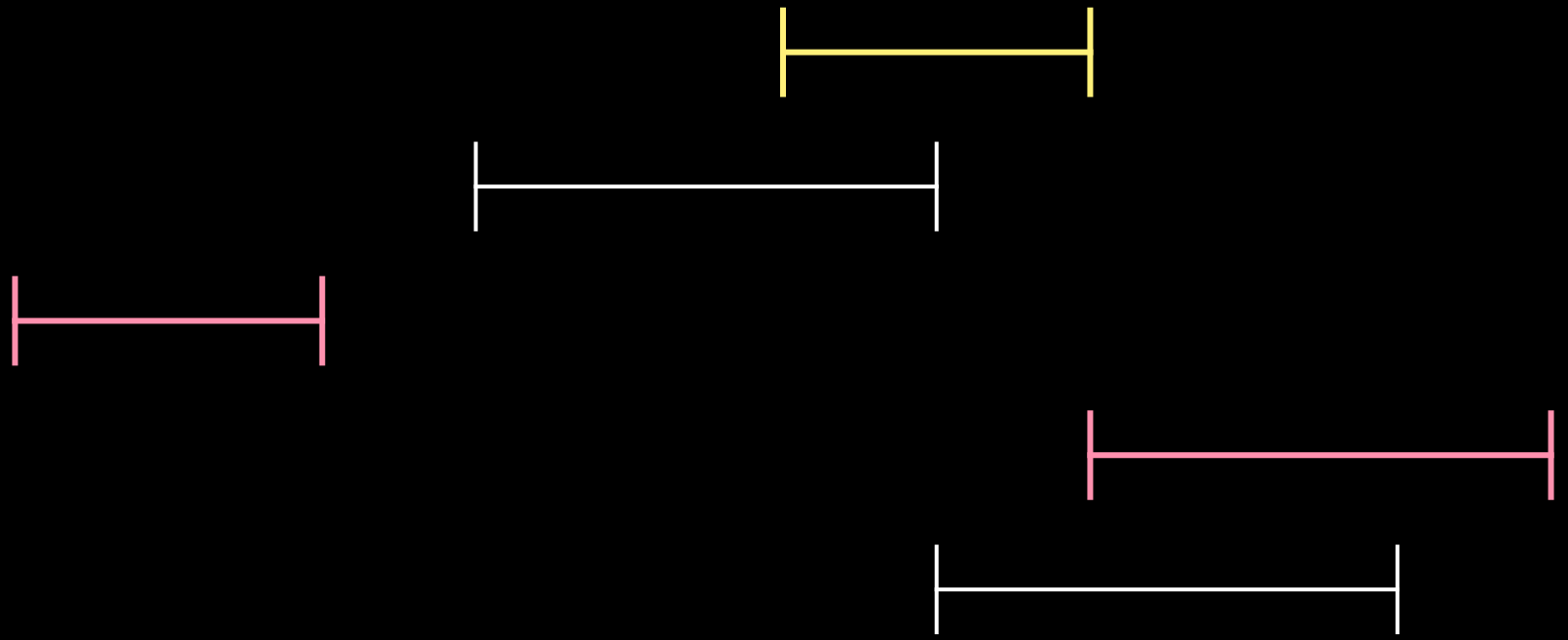
図示



图示



図示



$3 \geqq K$ ？

実装？

- 選択済み区間を vector に入れてごにょごにょ・・・

典型 座標圧縮

- 座標圧縮をすると実装が楽になることがある ← 今回
- 座標圧縮をしないと解けないこともある

実装

- 既に選んだ区間で使われてしまっている単位時間を 1 として累積和で管理する
- あとは $sum_r - sum_l > 0$ な区間を選ばないようにして，重ならない区間の個数の最大化をやるだけ
- 全体で $O(N^2)$ 時間

- 他にも実装方針多数
  - たぶんこれが最も細かいことを考えなくてよい

実装例

```
for(int i = 0; i < N; i++) P[i] = {R[i],i};
sort(P, P + N);
if(count() < K){
    cout << -1 << endl;
    return 0;
}
int ans = 0;
for(int i = 0; i < N && ans < K; i++){
    if(sum[R[i]] - sum[L[i]] > 0) continue;
    for(int j = L[i]; j < R[i]; j++) used[j]++;
    for(int j = 0; j < z.size(); j++) sum[j + 1] = sum[j] + used[j];
    if(ans + 1 + count() >= K){
        cout << i + 1 << endl;
        ans++;
    }
    else{
        for(int j = L[i]; j < R[i]; j++) used[j]--;
        for(int j = 0; j < z.size(); j++) sum[j + 1] = sum[j] + used[j];
    }
}
```

実装例

```
pair<int,int> P[MAX_N];
int used[2*MAX_N], sum[2*MAX_N+1];

int count(){
    int bef_r = 0, res = 0;
    for(int i = 0; i < N; i++){
        if(bef_r <= L[P[i].second] && sum[R[P[i].second]] - sum[L[P[i].second]] == 0){
            bef_r = R[P[i].second];
            res++;
        }
    }
    return res;
}
```

目次

目次

小課題４

追加制約なし

お詫び

- サルでもわかるように解説を書こうとしたら，
  ものすごく遠回りしてしまった気がします

お気持ち

- 計算回数を少しでも減らしたい・・・
- どこが削れるかな・・・

典型 差分

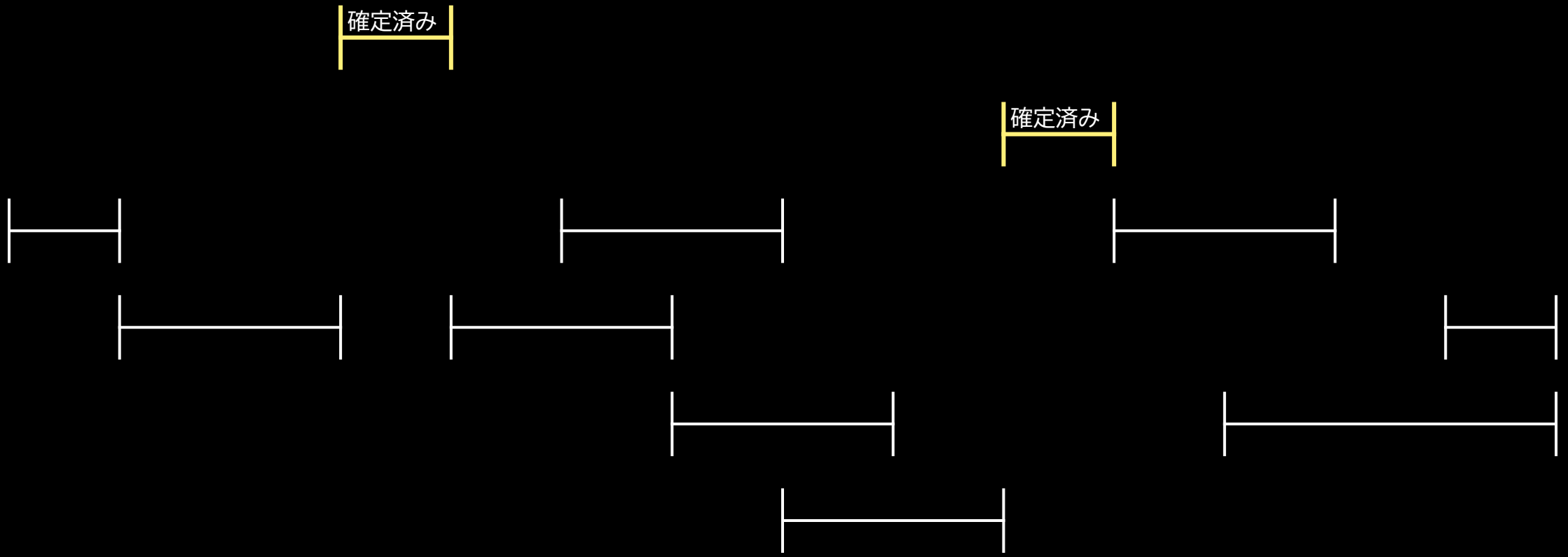
- 毎回全部計算しなおすのは大変
- 差分だけ計算しなおすことで,
  高速化できることがある

小課題３考察：判定方法

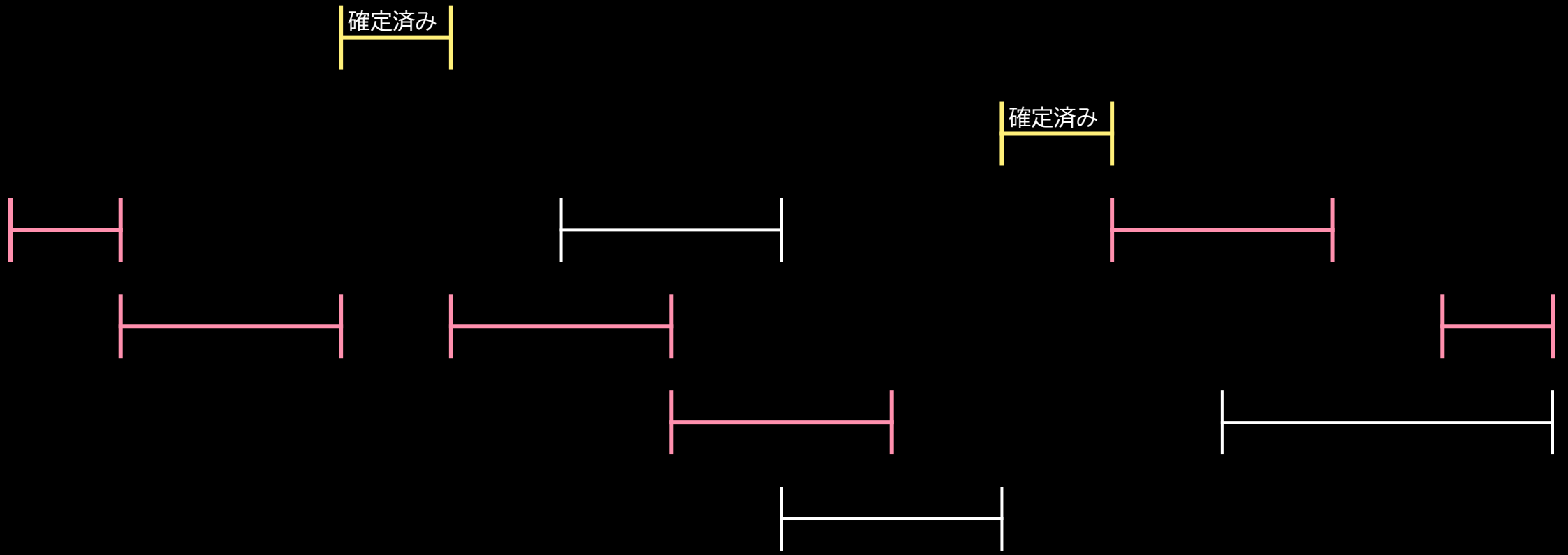
- 「重ならない区間の個数の最大化」を普通にやる
- ただし，選ばれている区間は必ず使うようにする

- 差分をよくよく観察してみよう

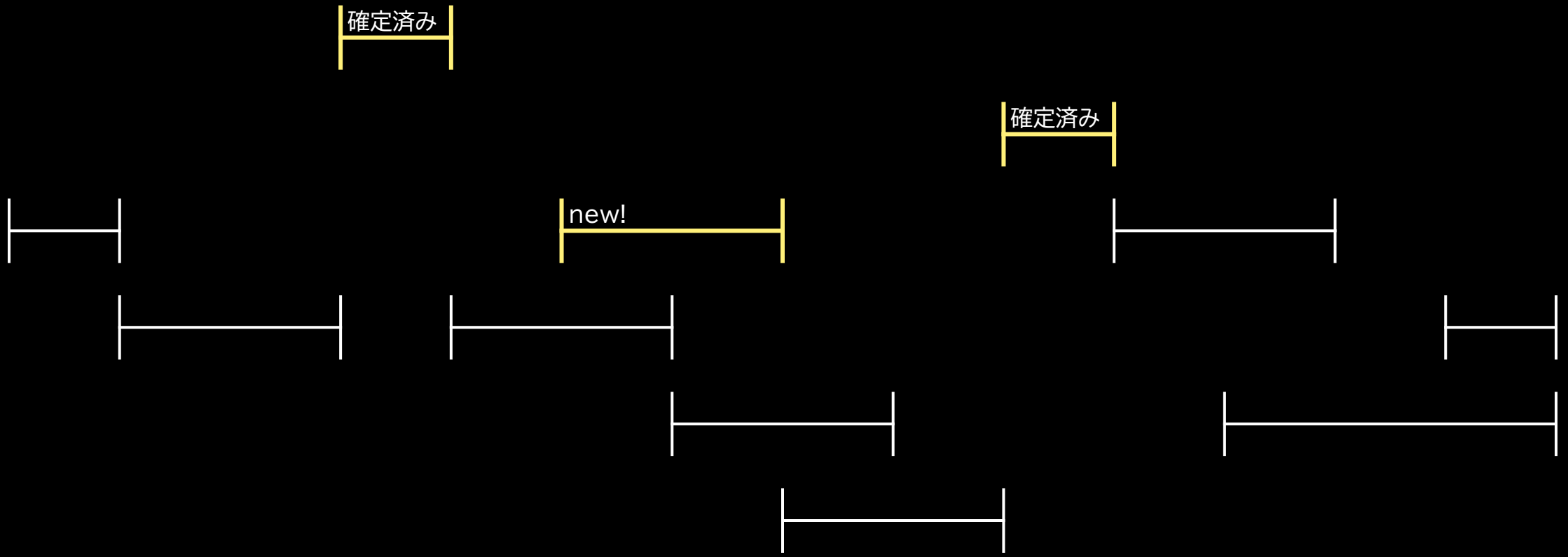
観察：例 1



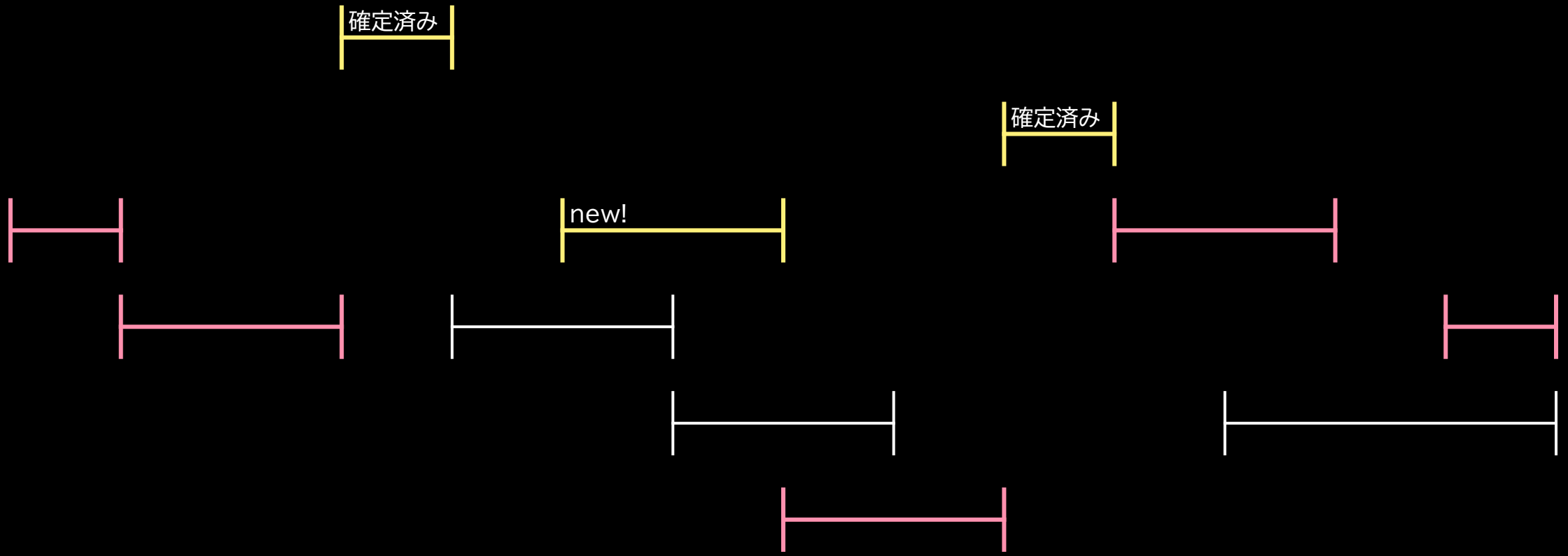
観察：例 1



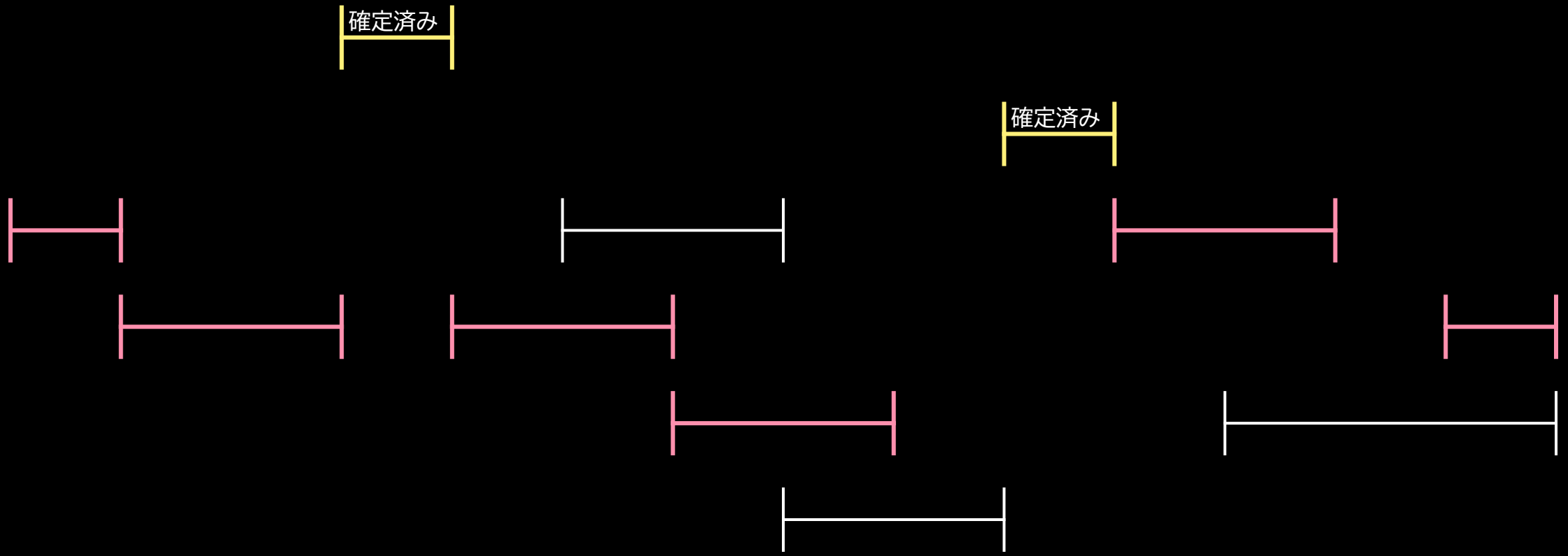
観察：例２



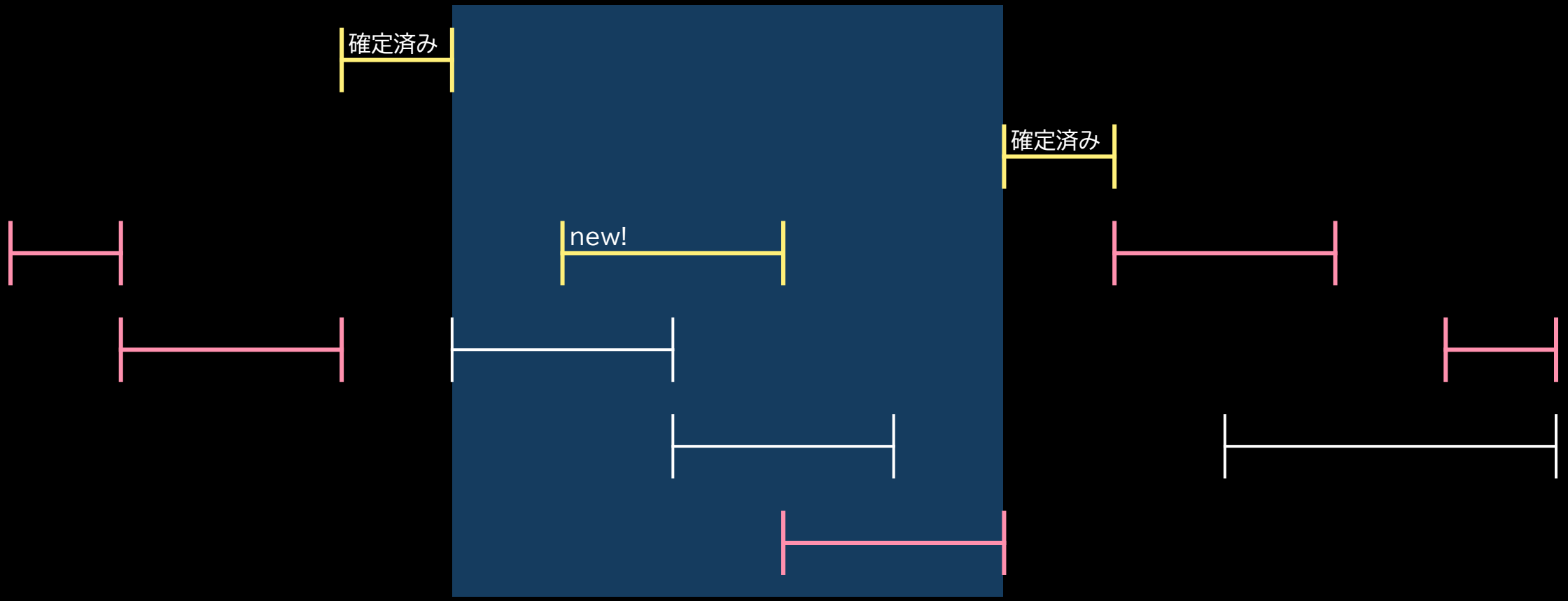
観察：例２



観察：例 1



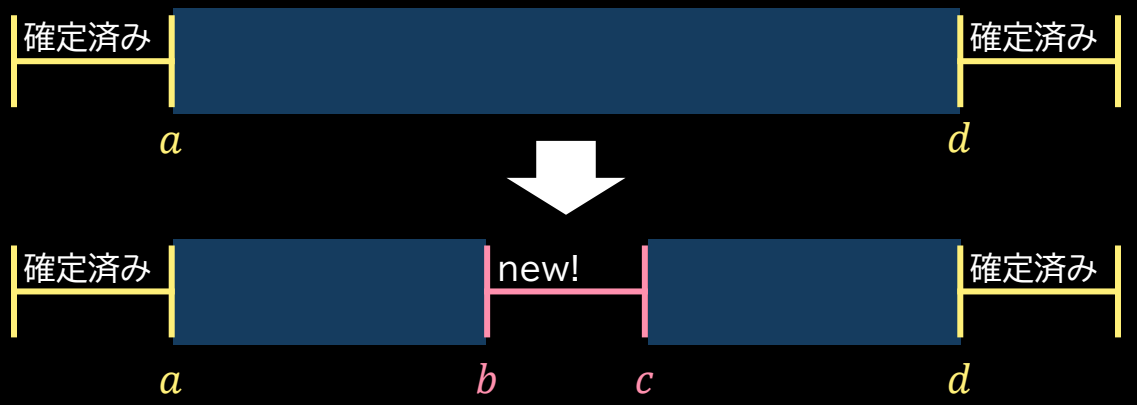
観察：例２



この範囲しか選び方が変わっていない！

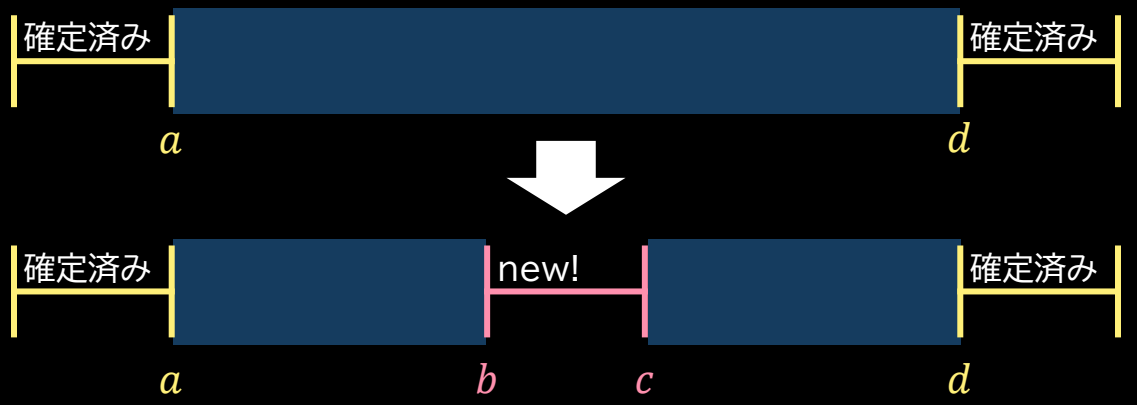
考察：判定方法

- 今まで
  - (既に選んだ個数) + (それを避けたときに選べる最大値) ≧ K
- 差分に着目した別の表現
  - (前回の個数) − (前回範囲 (a, d) で選んでいた個数)
    +(範囲 (a, b) で選べる最大値) + 1 + (範囲 (c, d) で選べる最大値)
    ≧ K



考察：判定方法

- 今まで
  - (既に選んだ個数) + (それを避けたときに選べる最大値) ≧ K
- 差分に着目した別の表現
  - (前回の個数) − (範囲 (a, d) で選べる最大値)
    +(範囲 (a, b) で選べる最大値) + 1 + (範囲 (c, d) で選べる最大値)
    ≧ K



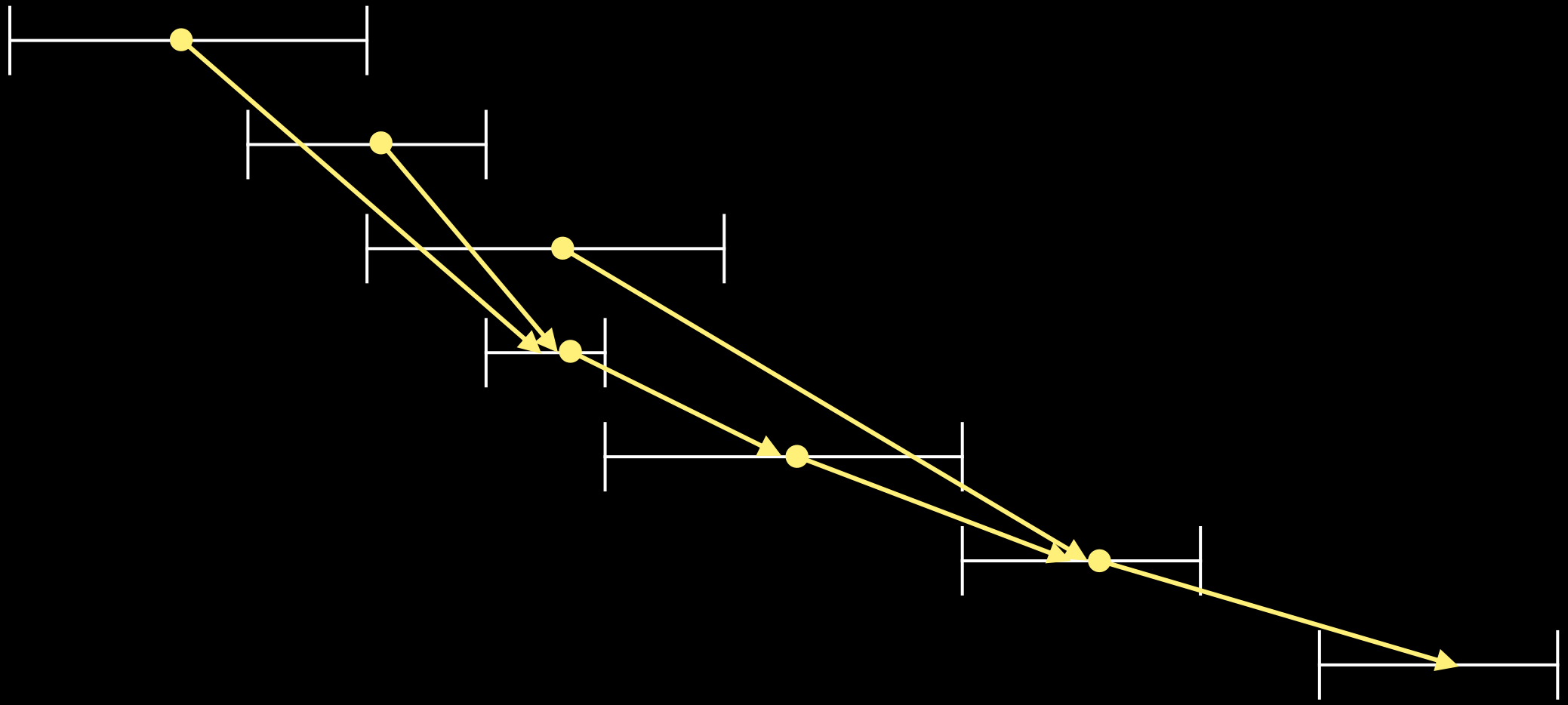
考察：範囲 (l, r) で選べる最大値

- 「範囲 (l, r) で選べる最大値」を
  高速に求められればよいことがわかった
- 「重ならない区間の個数の最大化」を
  普通にやっても遅い
- どうしよう・・・

典型 矢印

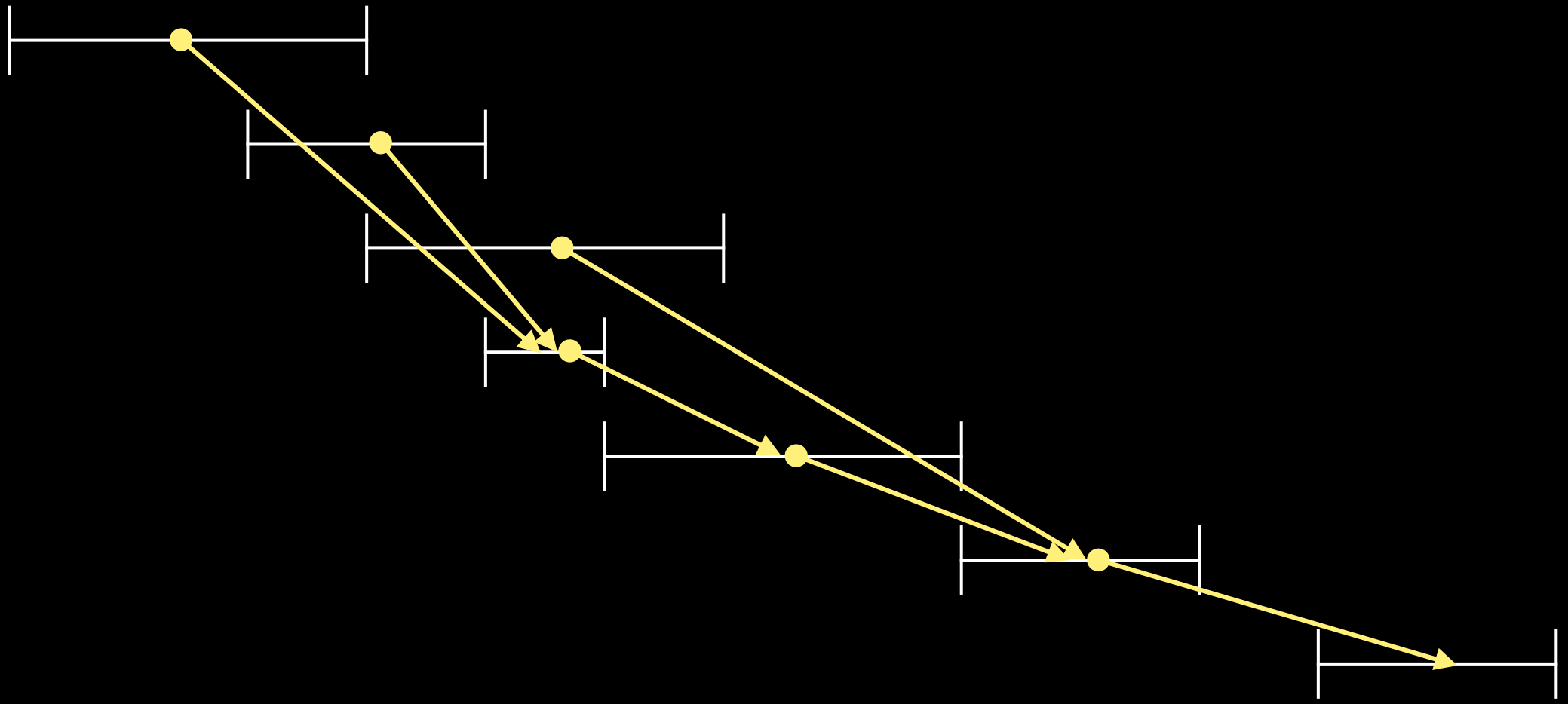
- 「前」「次」みたいな概念があるときは
  図に矢印を引いてみることで，考察が捗ることがある

観察



自分より右側にある区間のうち，右端が最も左のものに矢印を引いてみた

観察



根つき木（森）になっている！

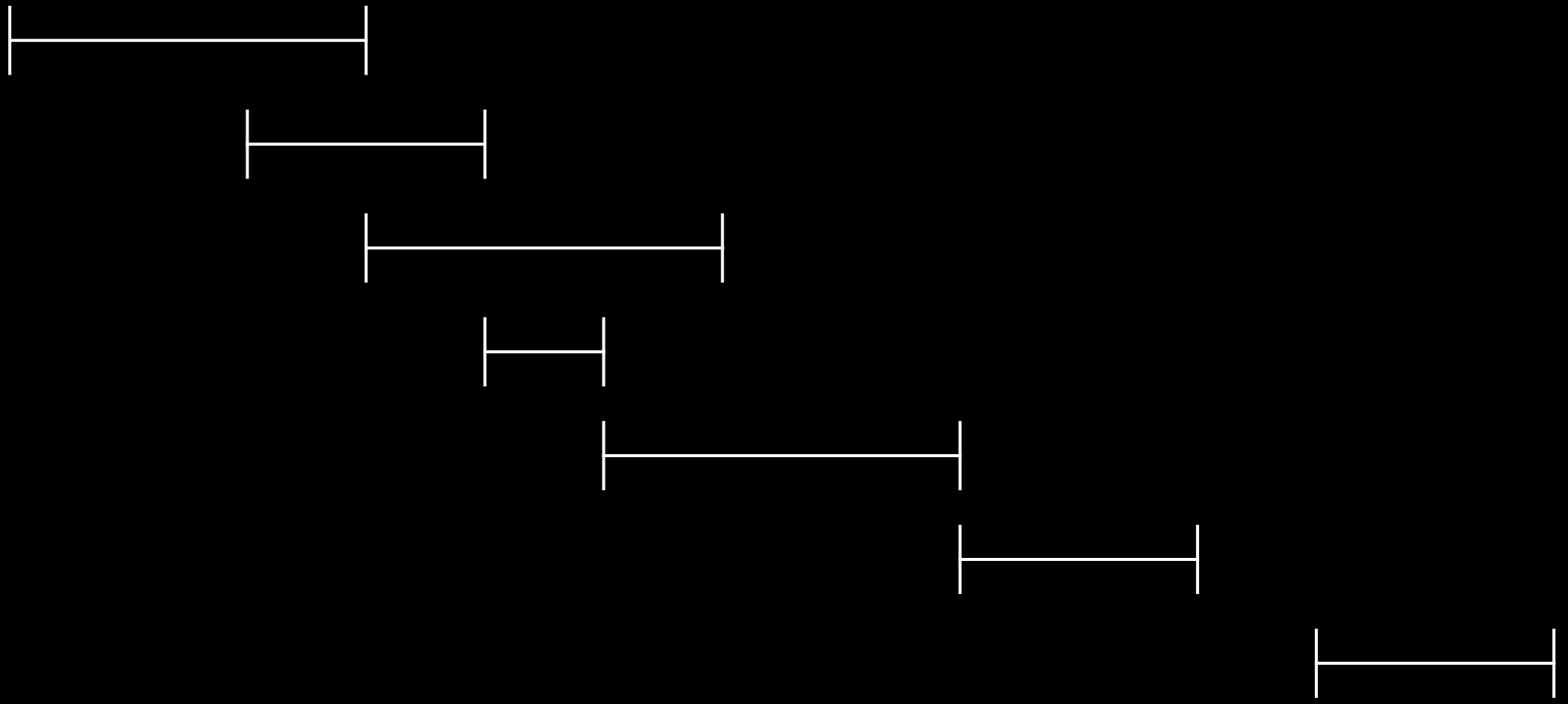
考察：方針

- 「重ならない区間の個数の最大化」は
  この木上の最長パスを求めることと対応している
  - R_i が最小の頂点が深さ最大
- 「範囲 (l, r) で選べる最大値」も同じで，
  - $l \leqq L_i$ な i のうち R_i が最小の頂点を見つけて
  - $R_j \leqq r$ な間，木を登り続ければよい

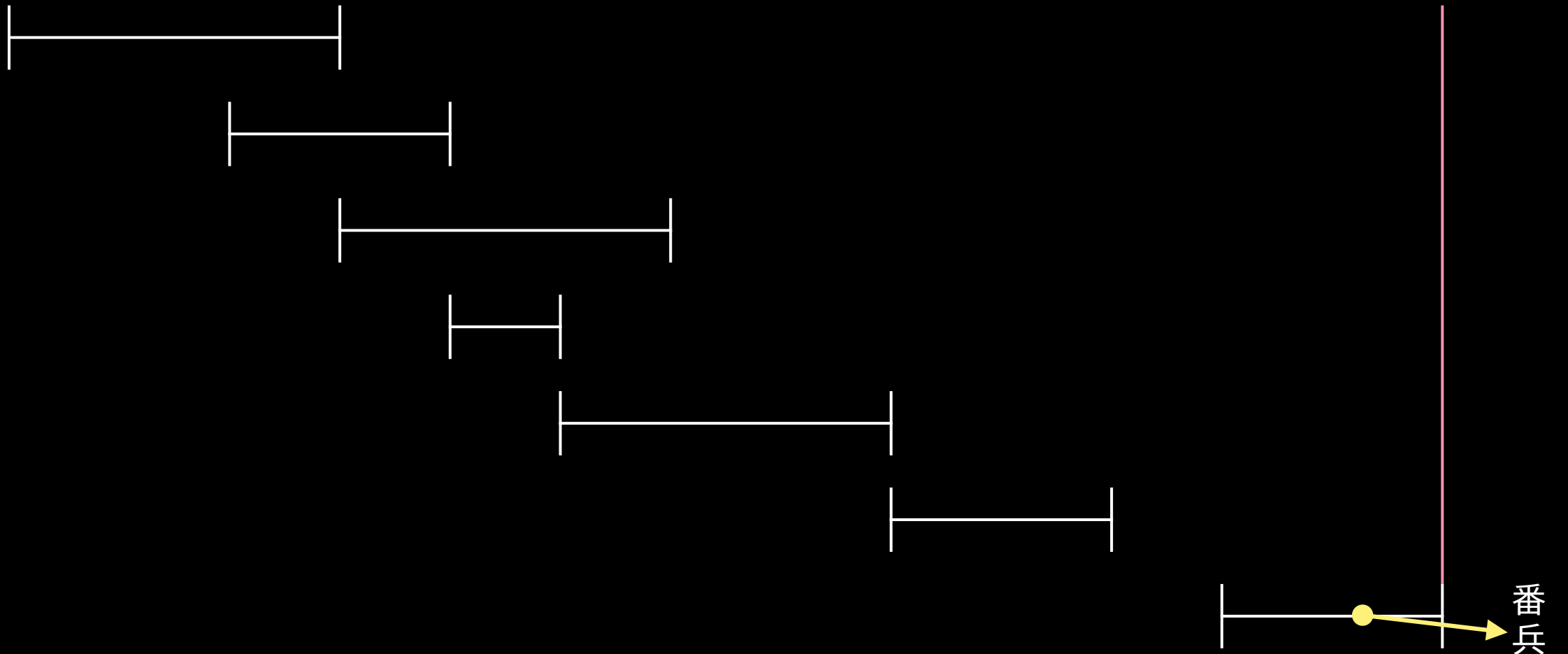
考察：$l \leqq L_i$ な i のうち R_i が最小の頂点

- 木を作るのと一緒に前計算しておく
  - 座圧しておく
  - 右から平面（?）走査する
  - 今見ているところより左にある区間のうち，
    右端が最も小さいものをメモする
  - 区間の左端をくぐったら，右端が最も小さいものを更新する
- 前計算 $O(N \log N)$

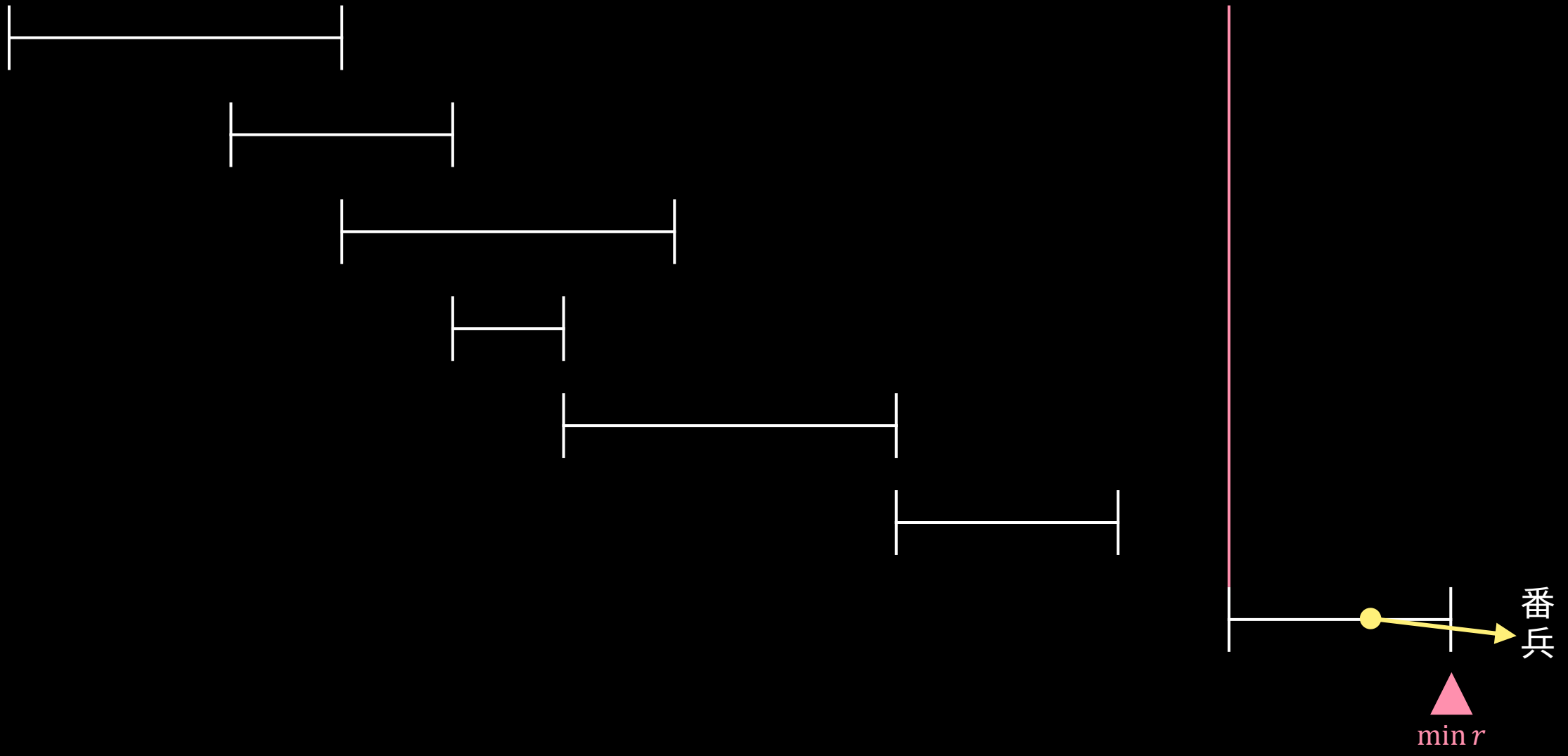
図示



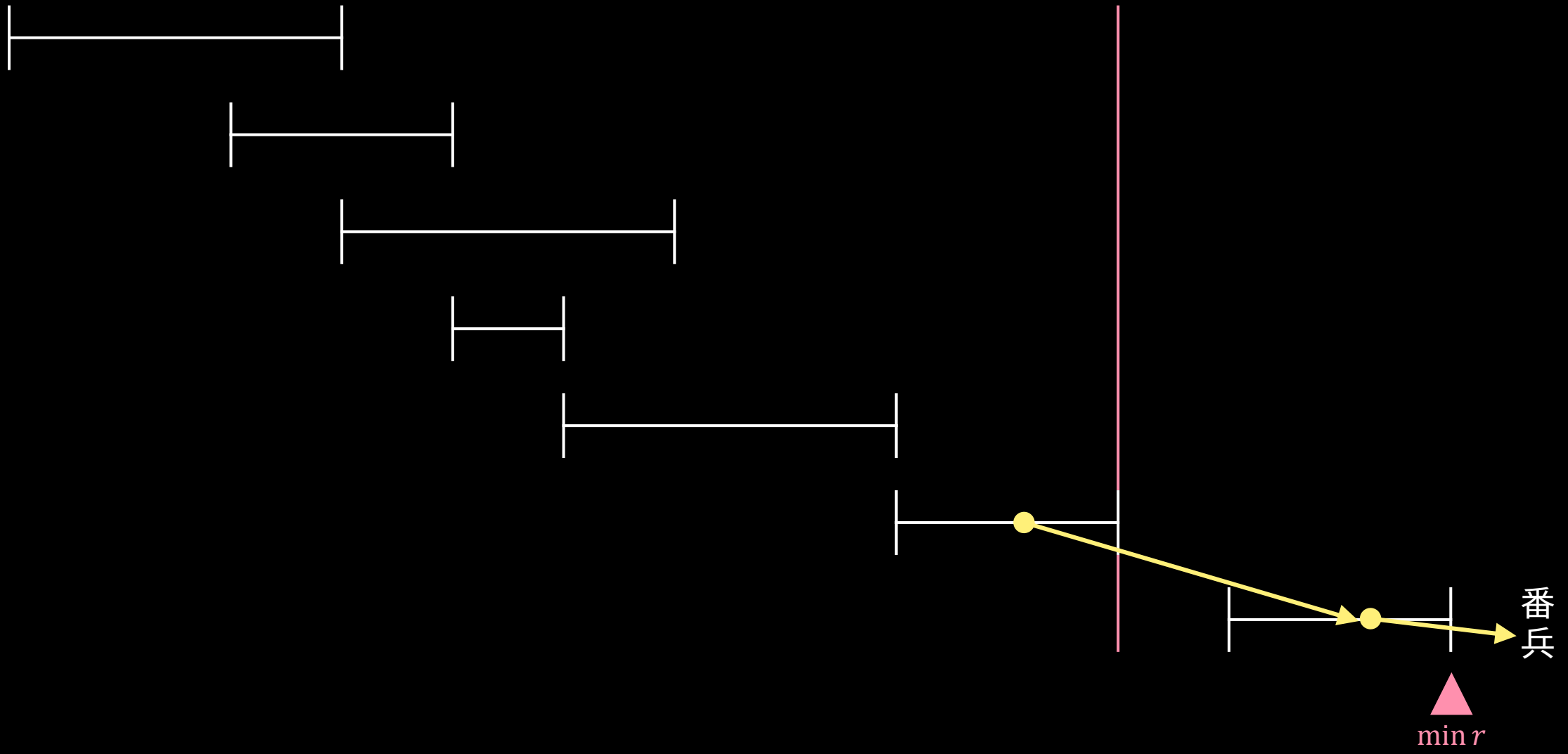
図示



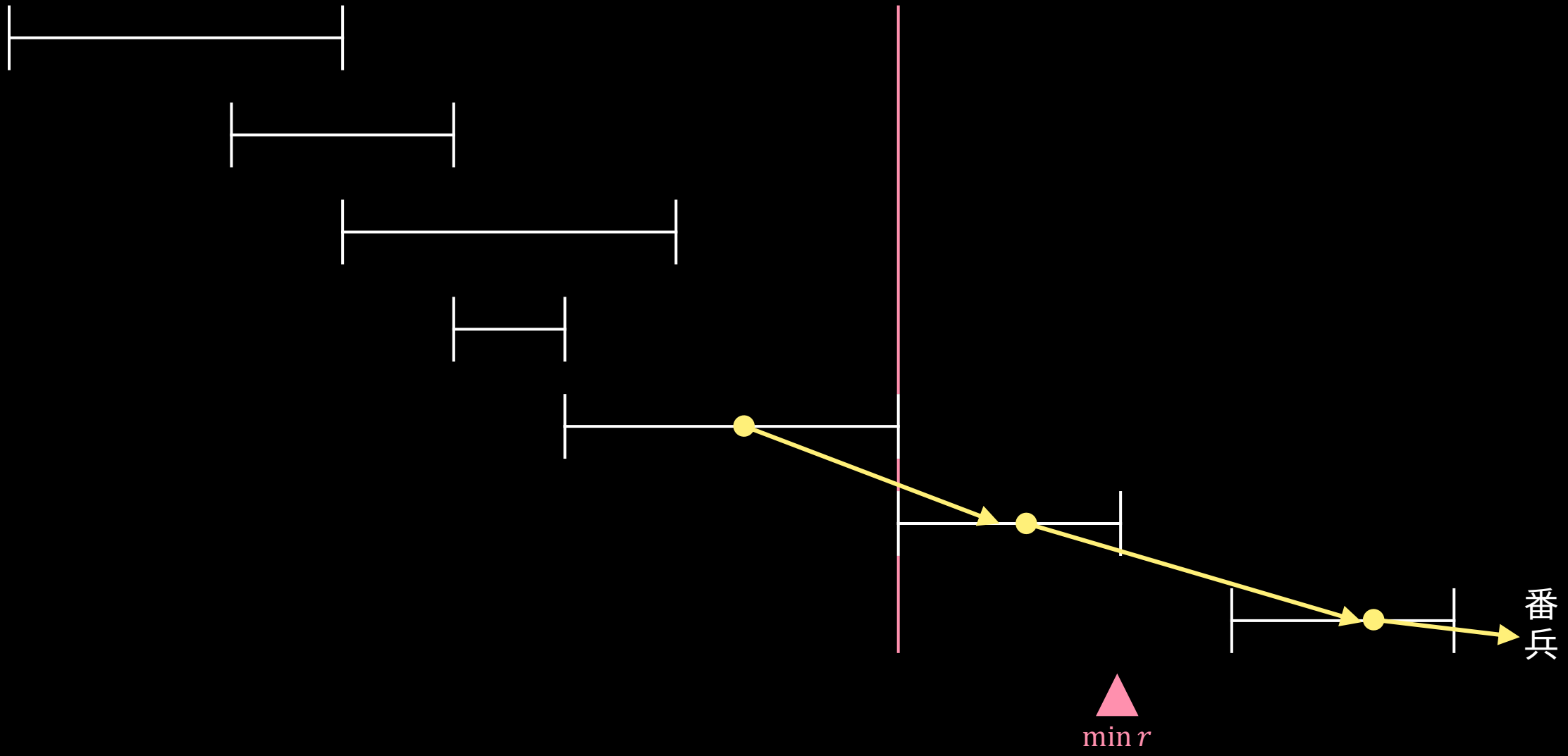
図示



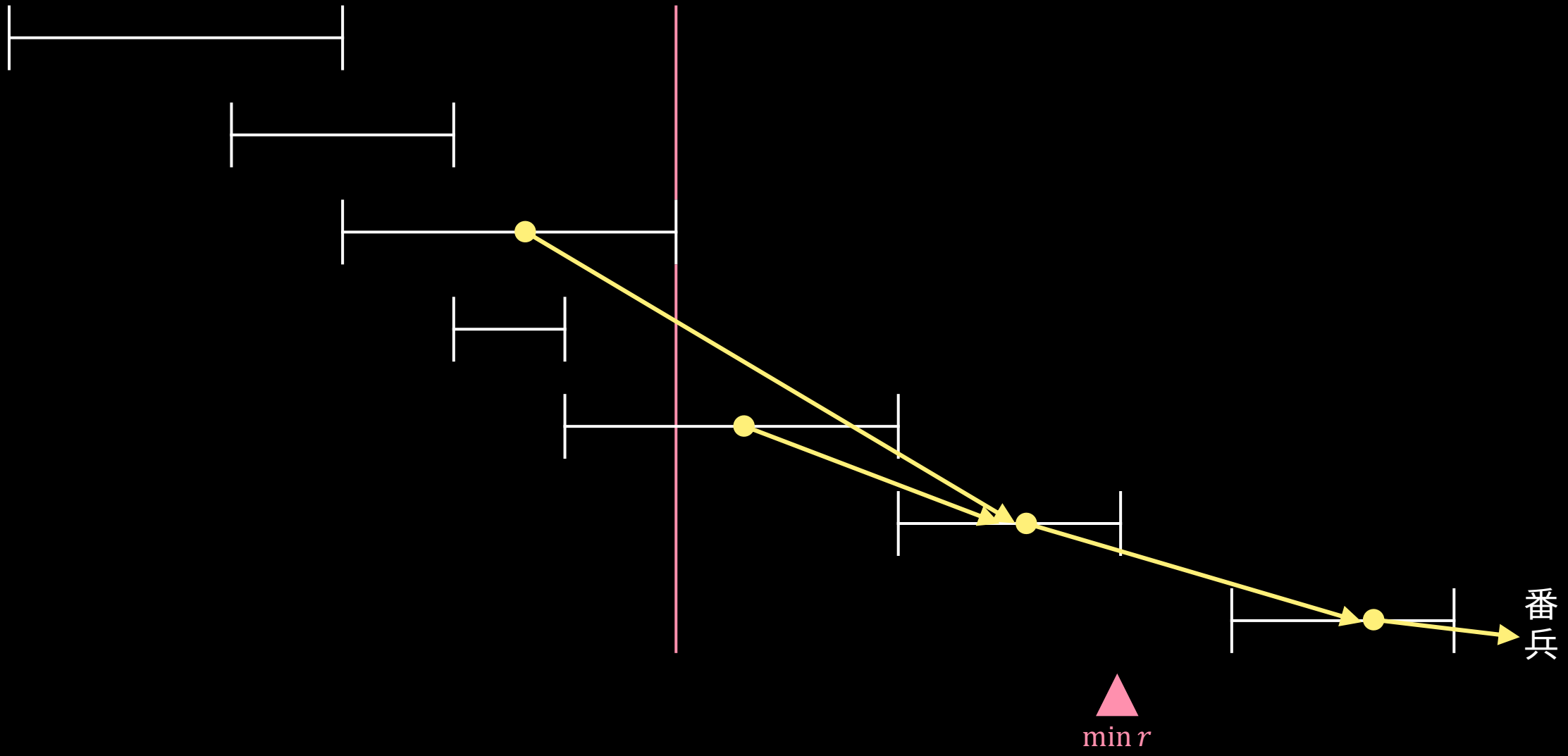
図示



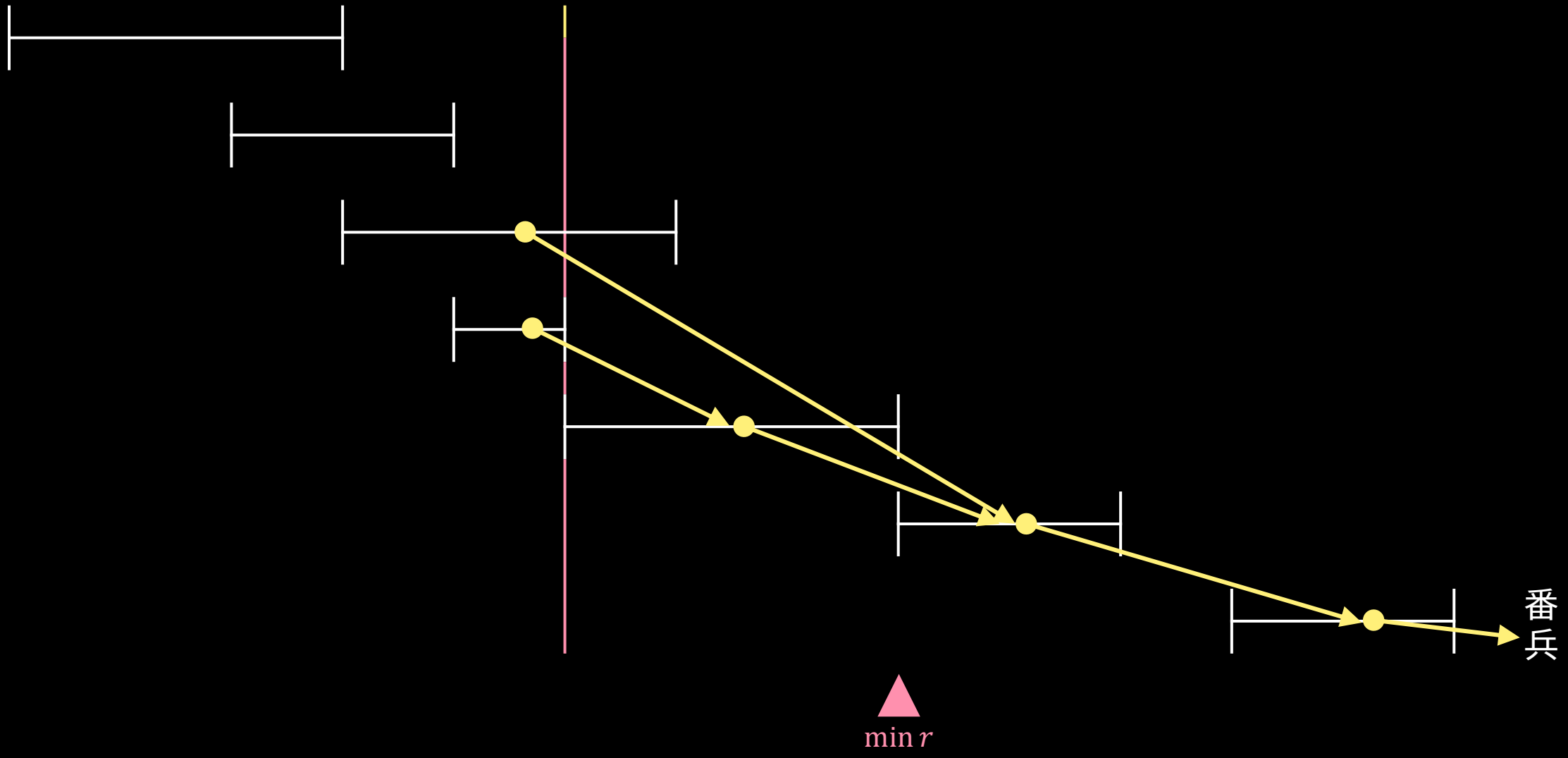
図示



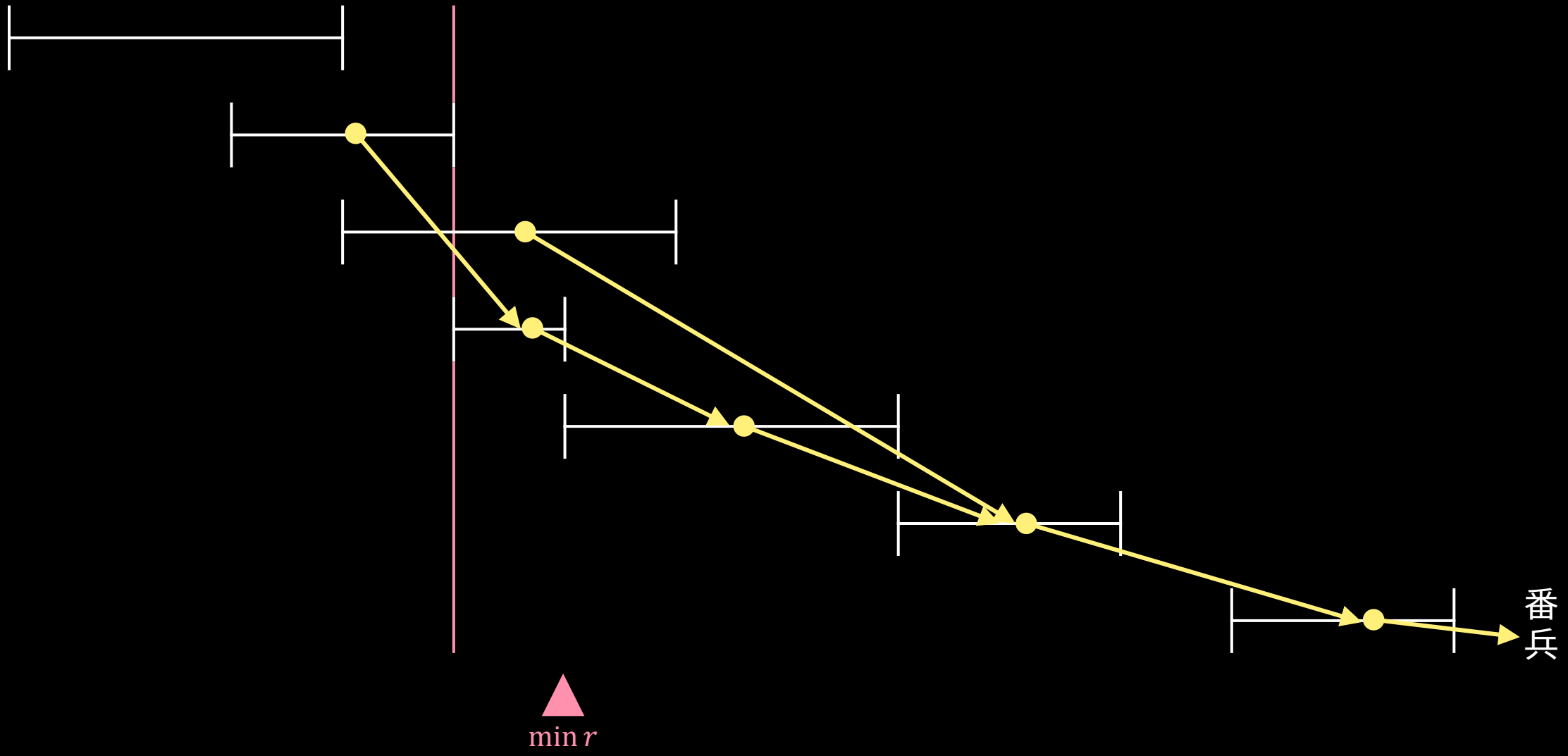
图示



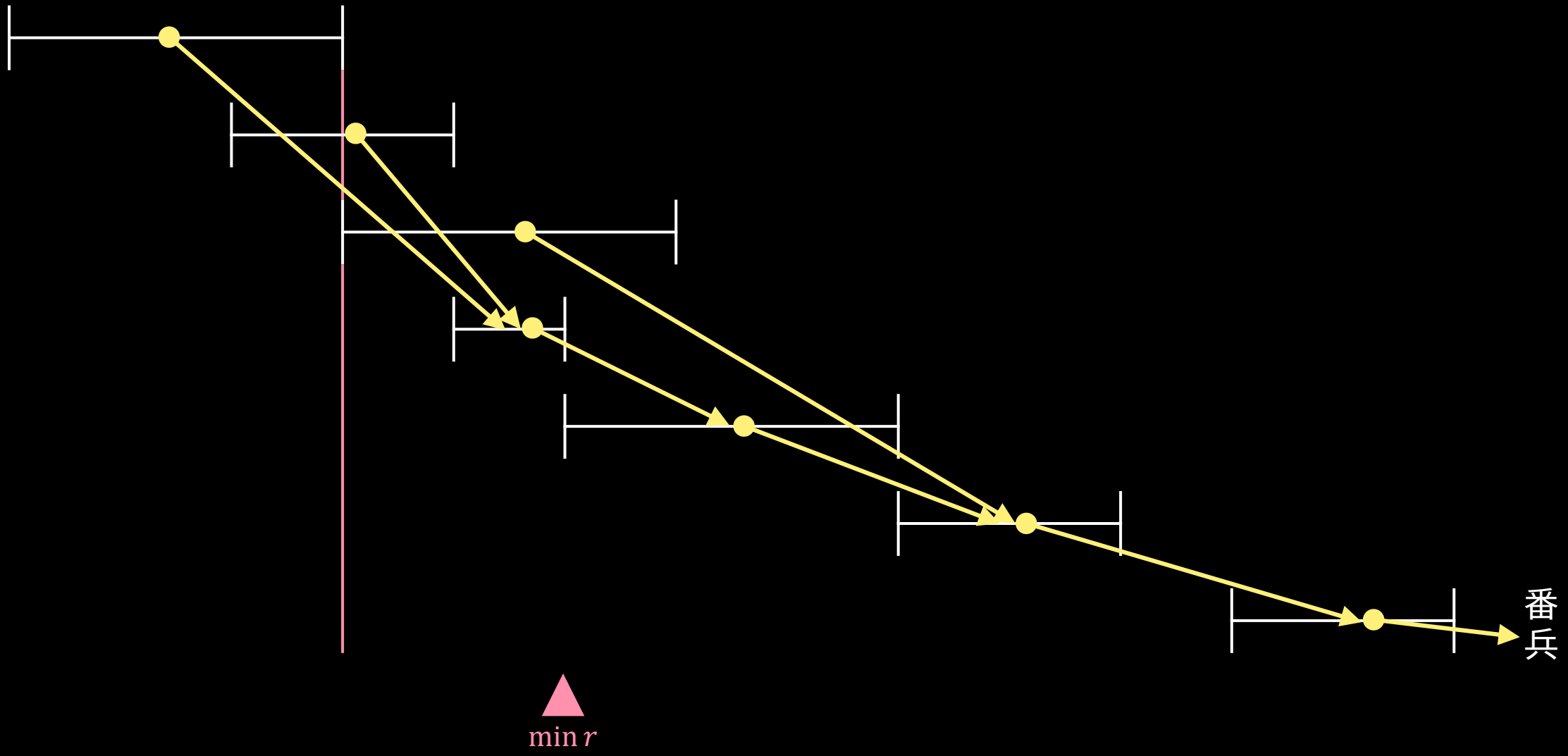
図示



図示



図示



考察：$R_j \leqq r$ な間，木を登り続ける

- １つずつ登っていっては毎回 $O(N)$ かかってしまう
- どうしよう・・・

典型 ダブリング

- 特定の条件を満たすギリギリのところまで登るには，ダブリングが使える
- 発想は二分探索と近い

考察：$R_j \leqq r$ な間，木を登り続ける

- 「2^i 祖先」を前計算しておく
- ダブリングを使って $O(\log N)$ ステップで登る
  - 2^{18} 祖先に登っても大丈夫？
  - 2^{17} 祖先に登っても大丈夫？
  - 2^{16} 祖先に登っても大丈夫？
  - ⋮

まとめ

- 座標圧縮
- （明示的でなくてよいので）
  木を作ってダブリングを前計算しておく：$O(N \log N)$
- 辞書順最小化の典型通り，条件を満たす状態を維持しながら 1 つずつ決めていく：$O(N \log N)$
  - 条件を満たすかの判定には，差分をダブリングで $O(\log N)$ で求められることを用いる
- 全体で $O(N \log N)$ 時間

実装例（木もどきの構築とダブリング）

```
vector<int> invL[MAX_N * 2 + 1];
int doubling[MAX_N * 2 + 1][MAX_logN];

for (int i = 0; i < N; i++) invL[L[i]].push_back(i);
int r = z.size();
for (int i = z.size(); i >= 0; i--) {
    for (int j = 0; j < invL[i].size(); j++) r = min(r, R[invL[i][j]]);
    doubling[i][0] = r;
}
for (int i = 1; i < MAX_logN; i++) {
    for (int j = 0; j <= z.size(); j++) {
        doubling[j][i] = doubling[doubling[j][i - 1]][i - 1];
    }
}
```

実装例（範囲 (l, r) で選べる最大値）

```
int count(int l, int r) {
    int res = 0;
    for (int j = MAX_logN - 1; j >= 0; j--) {
        if (doubling[l][j] <= r) {
            l = doubling[l][j];
            res += 1 << j;
        }
    }
    return res;
}
```

実装例（辞書順最小化）

```
int now = count(0, z.size() - 1);
if (now < K) {
    printf("-1¥n");
    return 0;
}
set<pair<int, int>> used = {{0, 0}, {z.size() - 1, z.size() - 1}};
for (int i = 0; used.size() - 2 < K; i++) {
    int l = prev(used.lower_bound({R[i], 0}))->second;
    int r = used.lower_bound({R[i], 0})->first;
    if (L[i] < l) continue;
    int tmp = now - count(l, r) + count(l, L[i]) + 1 + count(R[i], r);
    if (tmp >= K) {
        printf("%d¥n", i + 1);
        now = tmp;
        used.insert({L[i], R[i]});
    }
}
```

統計情報

得点分布

小課題ごとの正答者数

小課題 1	$L_i \leqq L_{i+1}$	9 / 14 人正解
小課題 2	$N \leqq 20$	12 / 14 人正解
小課題 3	$N \leqq 3\,000$	12 / 14 人 正解
小課題 4	追加制約なし	5 / 14 人 正解