Algorithmentheorie

10 – Greedy Verfahren

Robert Elsässer

Greedy Verfahren

1. Allgemeine Vorbemerkungen
2. Einfache Beispiele
   - Münzwechselproblem
   - Handlungsreisenden-Problem
3. Das Aktivitäten Auswahlproblem

Greedy Verfahren zur Lösung eines Optimierungsproblems

Treffe in jedem Verfahrensschritt diejenige Entscheidung, die im Moment am besten ist!

Möglichkeiten:

1. Wir erhalten stets die optimale Gesamtlösung.

2. Wir erhalten eine Lösung, die zwar nicht immer optimal ist, aber vom Optimum stets nur wenig abweicht.

3. Die berechnete Lösung kann beliebig schlecht werden.

Einfache Beispiele: Münzwechsel-Problem

EUR Bargeld-Werte:

500, 200, 100, 50, 20, 10, 5, 2, 1

Beobachtung

Jeder EUR Betrag kann durch Münzen und Banknoten mit diesen Werten bezahlt werden.

Ziel

Bezahlung eines Betrages n mit möglichst wenig Münzen und Banknoten

Greedy-Verfahren

Wähle die maximale Zahl von Banknoten und Münzen mit jeweils größtmöglichem Wert, bis der gewünschte Betrag n erreicht ist.

Beispiel: $n = 487$

500 200 100 50 20 10 5 2 1

Allgemeines Münzwechselproblem

Werte von Münzen und Banknoten: $n_1, n_2, ..., n_k$

$n_1 > n_2 > ... > n_k$, und $n_k = 1$.

Greedy Zahlungsverfahren:

1. $w = n$
2. **for** $i = 1$ **to** k **do**

   # Münzen mit Wert $m_i = \lfloor w / n_i \rfloor$

   $w = w - m_i \, n_i$

Jeder Geldbetrag kann bezahlt werden!

Land Absurdia

Drei Münzen:

$n_3 = 1, n_2 > 1$ *beliebig*, $n_1 = 2\, n_2 + 1$

Beispiel: 41, 20, 1

Zu zahlender Betrag: $n = 3\, n_2$ (z.B. $n = 60$)

Optimale Zahlungsweise:

Greedy Zahlungsverfahren:

Handlungsreisenden-Problem (TSP)

Gegeben: n Orte und Kosten $c(i,j)$, um von i nach j zu reisen

Gesucht: Eine billigste Rundreise, die alle Orte genau einmal besucht.

Formal: Eine Permutation p von $\{1, 2, ..., n\}$, so dass

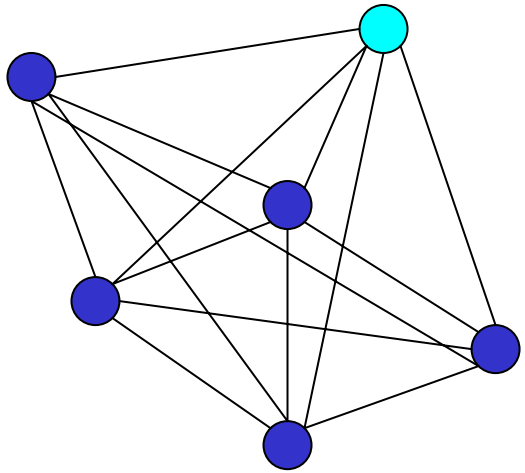
$c(p(1),p(2)) + \bullet\bullet\bullet + c(p(n-1),p(n)) + c(p(n),p(1))$

minimal ist.

Handlungsreisenden-Problem (TSP)

Greedy Verfahren zur Lösung von TSP

Beginne mit Ort 1 und gehe jeweils zum nächsten bisher noch nicht besuchten Ort. Wenn alle Orte besucht sind, kehre zum Ausgangsort 1 zurück.



Handlungsreisenden-Problem (TSP)

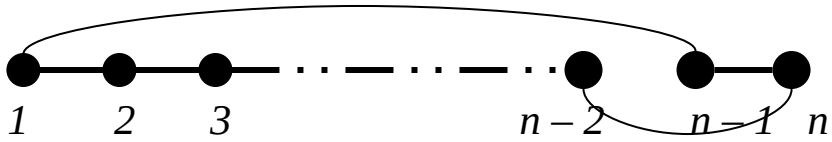
Beispiel

$c(i,i+1) = 1$, für $i = 1, ..., n - 1$

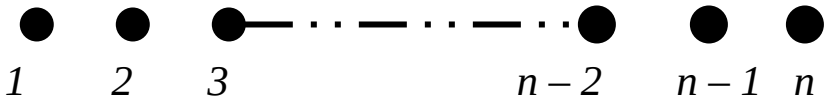
$c(n,1) = M$ (für eine sehr große Zahl M)

$c(i,j) = 2$, sonst

Optimale Tour:



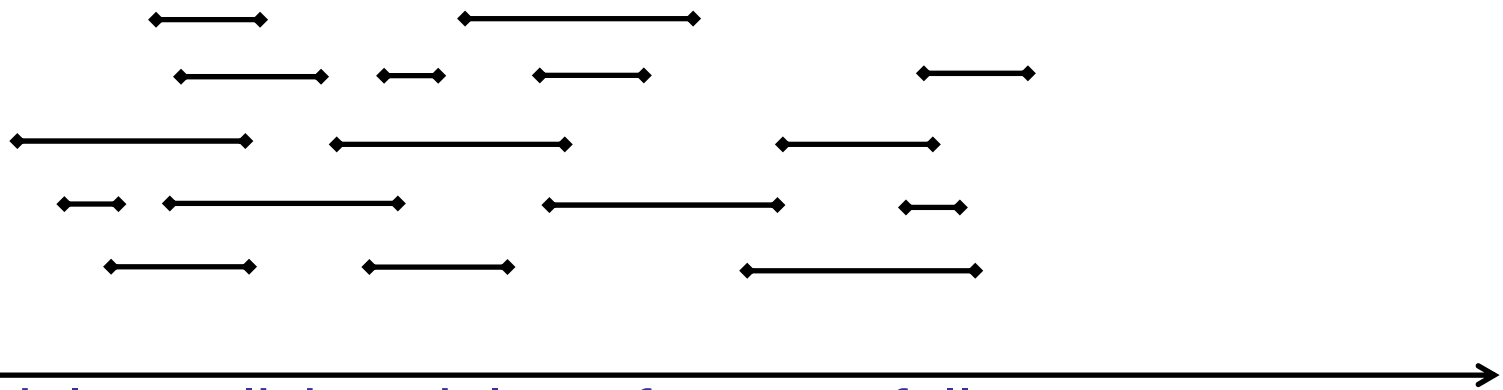
Vom Greedy Verfahren berechnete Tour:



Aktivitäten Auswahlproblem

Intervall Scheduling:

- Ressource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Kann ich die Ressource für den Zeitraum (t_1, t_2) nutzen?

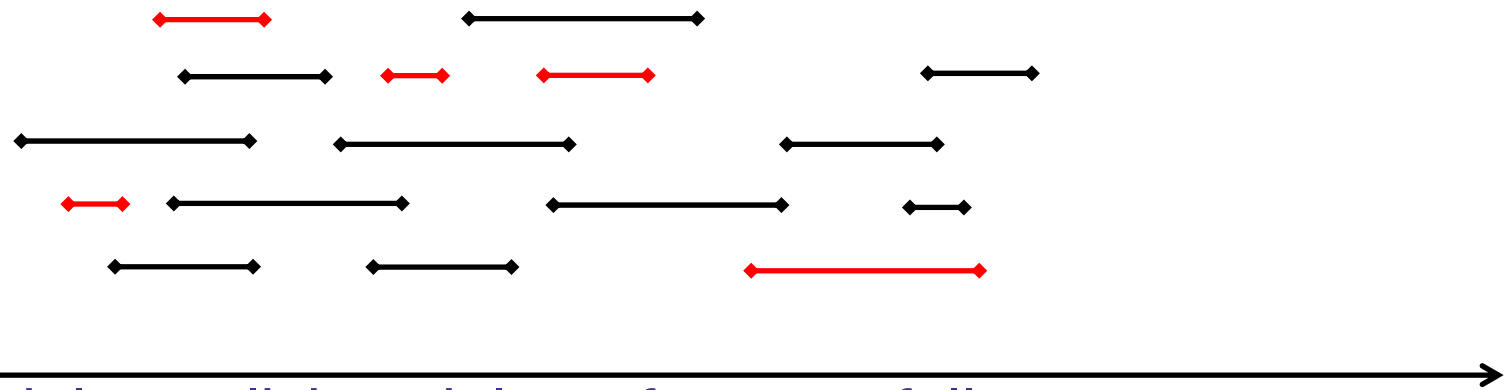


- Ziel: Möglichst viele Anfragen erfüllen

Intervall Scheduling

Intervall Scheduling:

- Ressource (Hörsaal, Parallelrechner, Elektronenmikroskop,..)
- Anfragen: Kann ich die Ressource für den Zeitraum (t_1, t_2) nutzen?

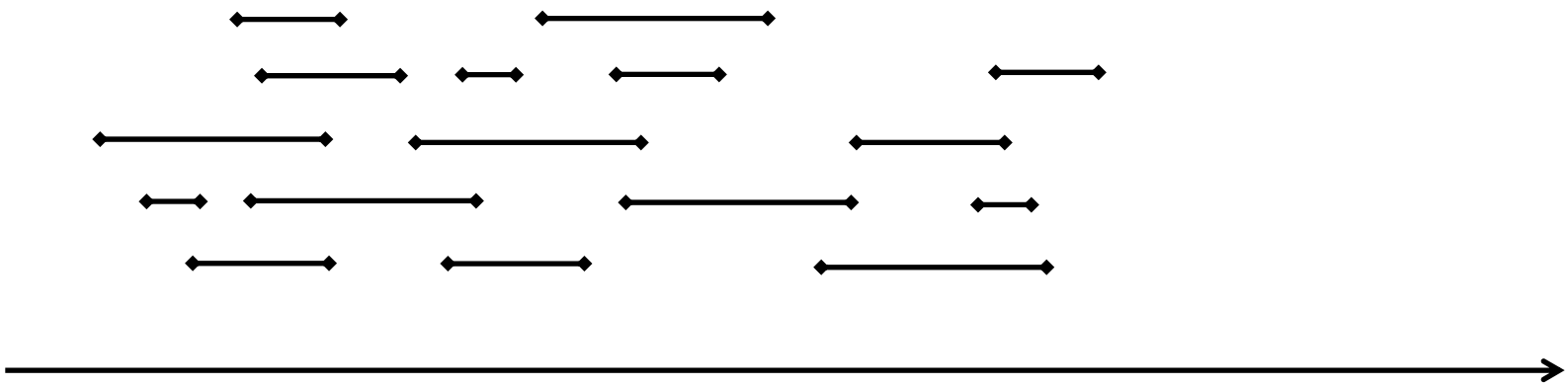


- Ziel: Möglichst viele Anfragen erfüllen

Intervall Scheduling

Definition:

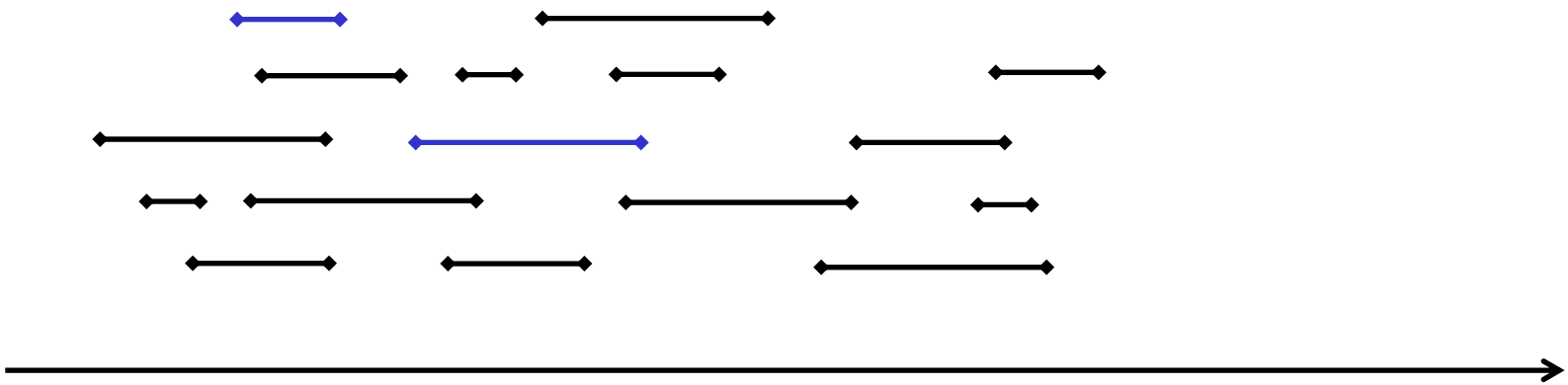
- Zwei Anfragen heißen kompatibel, wenn sich die Intervalle nicht überschneiden.



Intervall Scheduling

Definition:

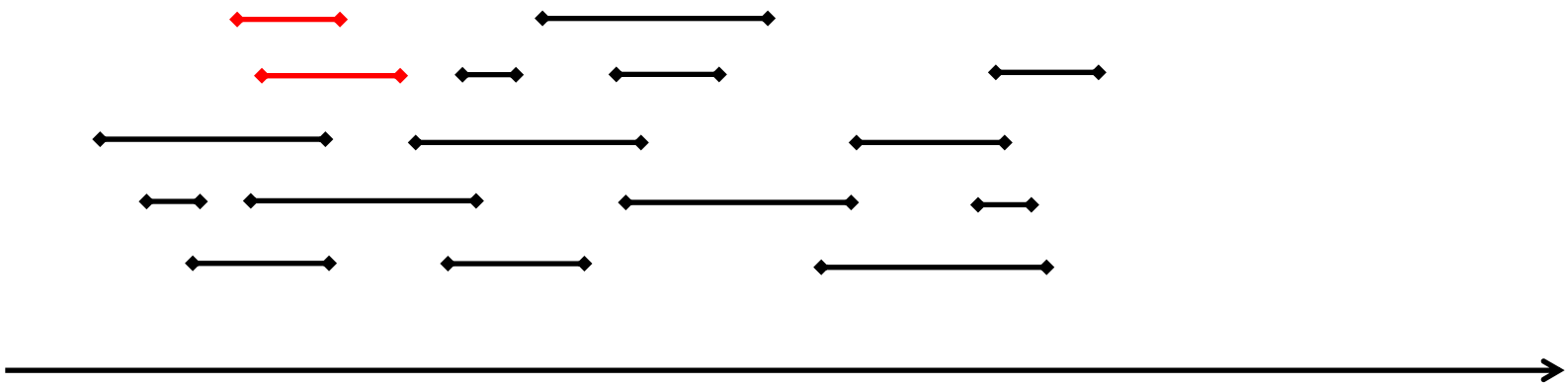
- Zwei Anfragen heißen kompatibel, wenn sich die Intervalle nicht überschneiden.



Intervall Scheduling

Definition:

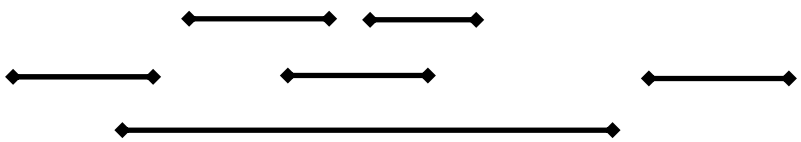
- Zwei Anfragen heißen kompatibel, wenn sich die Intervalle nicht überschneiden.



Intervall Scheduling

Generelle Überlegung:

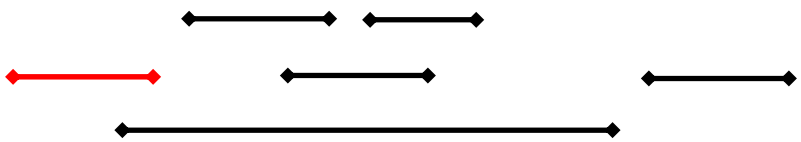
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Intervall Scheduling

Generelle Überlegung:

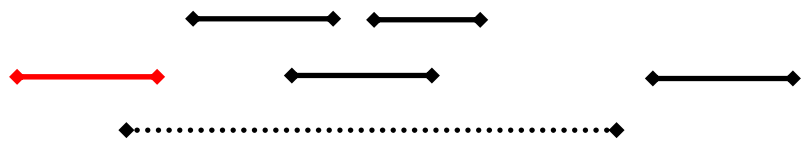
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Intervall Scheduling

Generelle Überlegung:

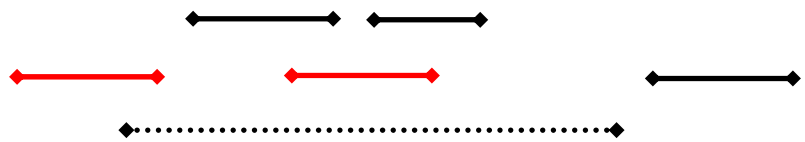
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Intervall Scheduling

Generelle Überlegung:

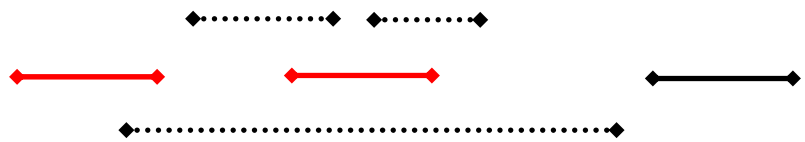
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Intervall Scheduling

Generelle Überlegung:

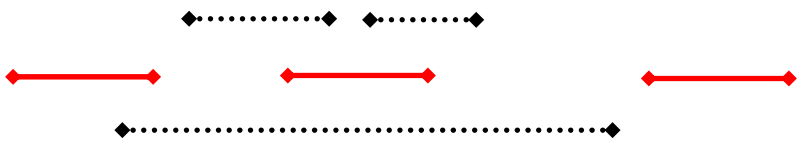
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Intervall Scheduling

Generelle Überlegung:

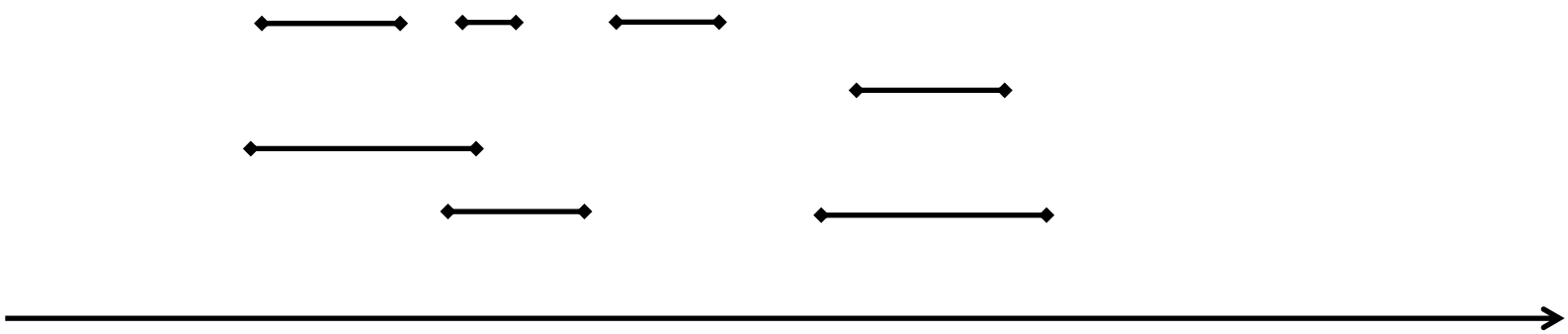
- Wähle erste Anfrage i_1 geschickt
- Ist i_1 akzeptiert, weise alle Anfragen zurück, die nicht kompatibel sind
- Wähle nächste Anfrage i_2 und weise alle Anfragen zurück, die nicht mit i_2 kompatibel sind
- Mache weiter, bis keine Anfragen mehr übrig sind



Intervall Scheduling

Strategie 1:

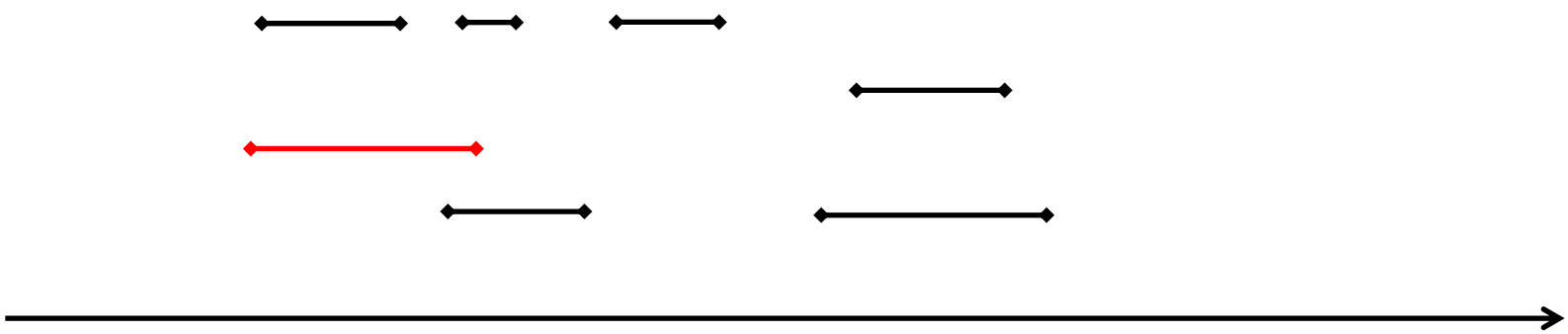
- Wähle immer die Anfrage, die am frühesten beginnt



Intervall Scheduling

Strategie 1:

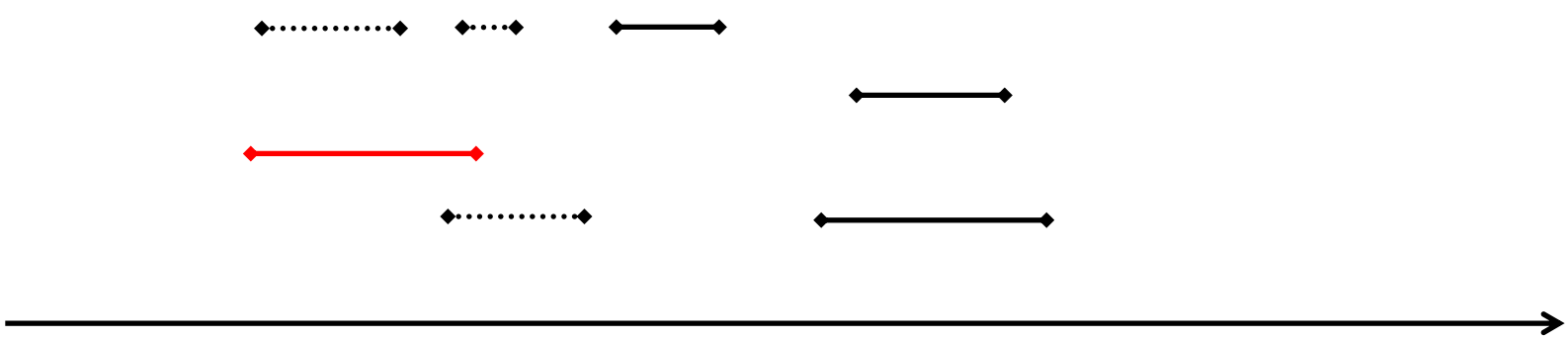
- Wähle immer die Anfrage, die am frühesten beginnt



Intervall Scheduling

Strategie 1:

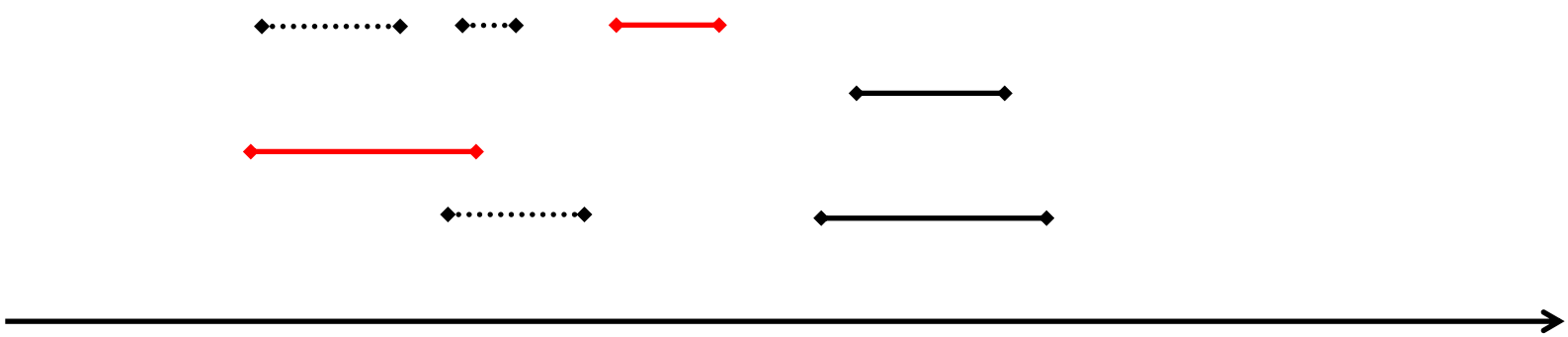
- Wähle immer die Anfrage, die am frühesten beginnt



Intervall Scheduling

Strategie 1:

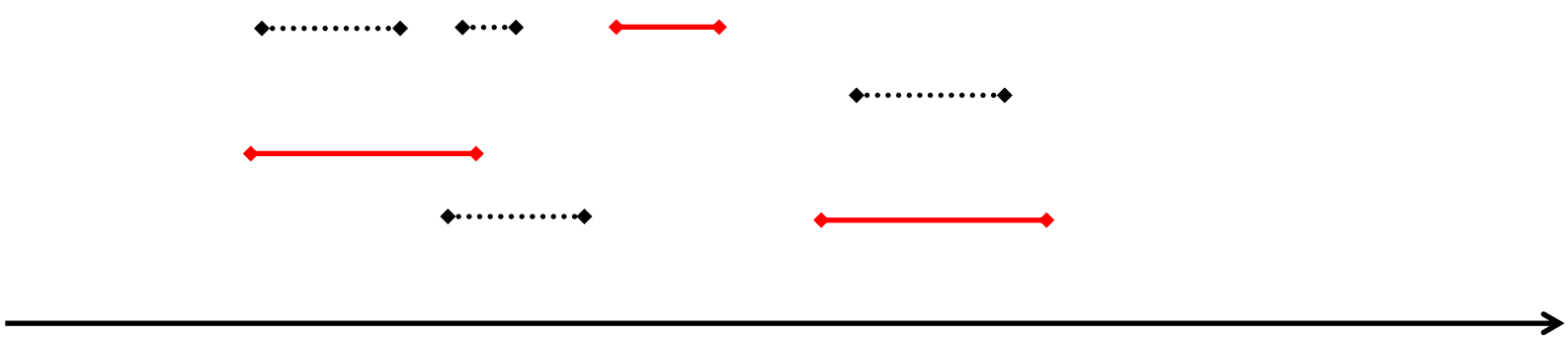
- Wähle immer die Anfrage, die am frühesten beginnt



Intervall Scheduling

Strategie 1:

- Wähle immer die Anfrage, die am frühesten beginnt

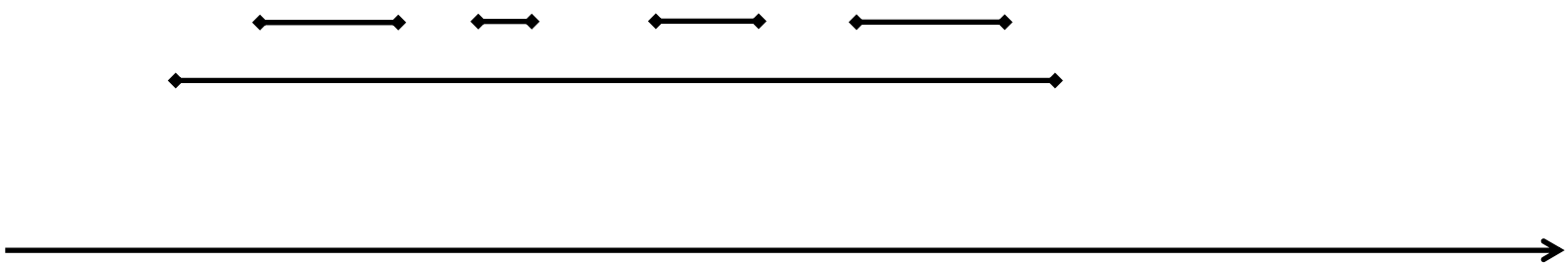


Intervall Scheduling

Strategie 1:

- Wähle immer die Anfrage, die am frühesten beginnt

Optimalität ?

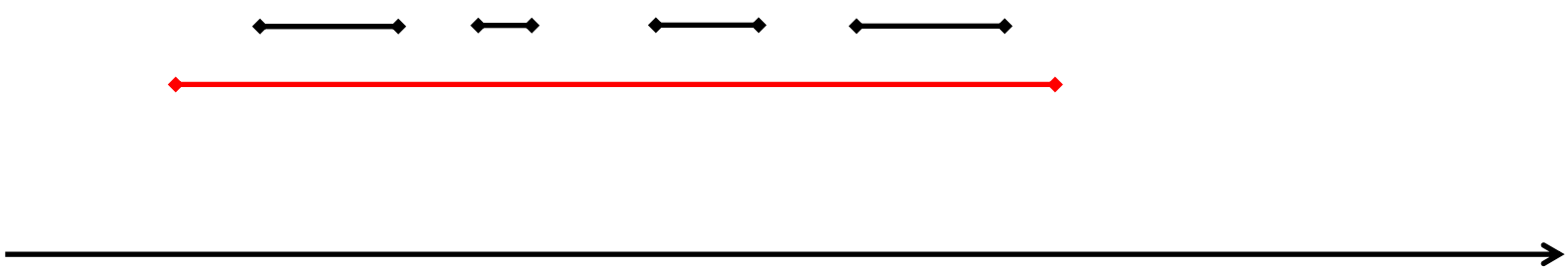


Intervall Scheduling

Strategie 1:

- Wähle immer die Anfrage, die am frühesten beginnt

Optimalität ?

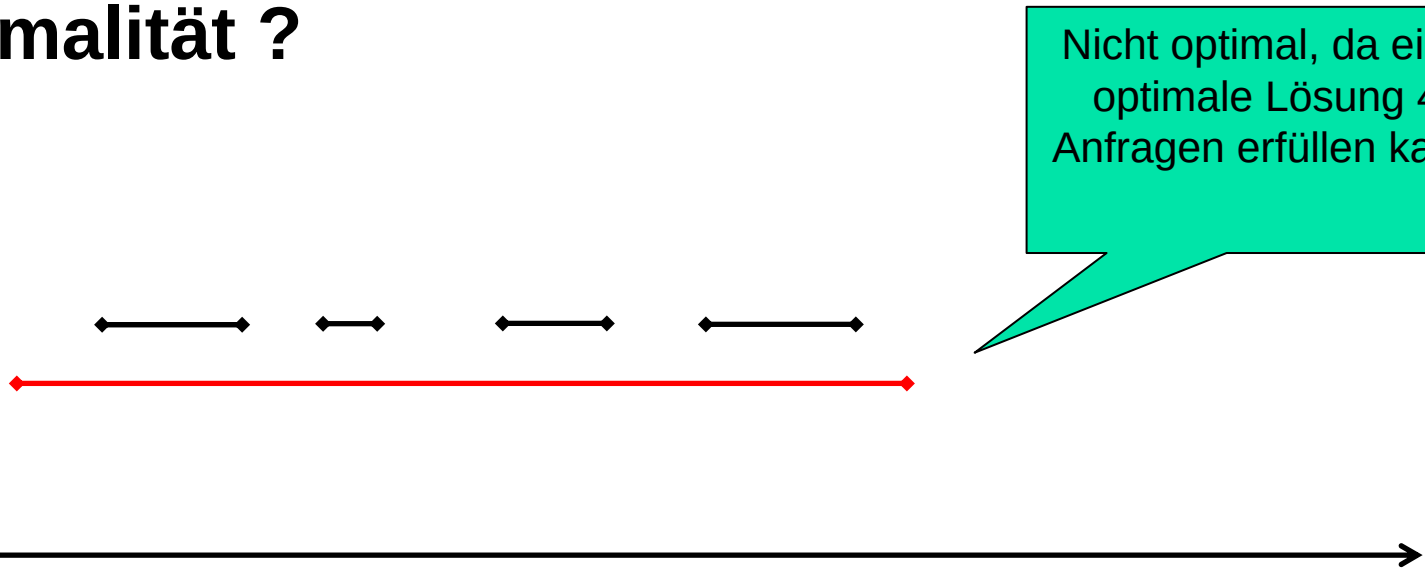


Intervall Scheduling

Strategie 1:

- Wähle immer die Anfrage, die am frühesten beginnt

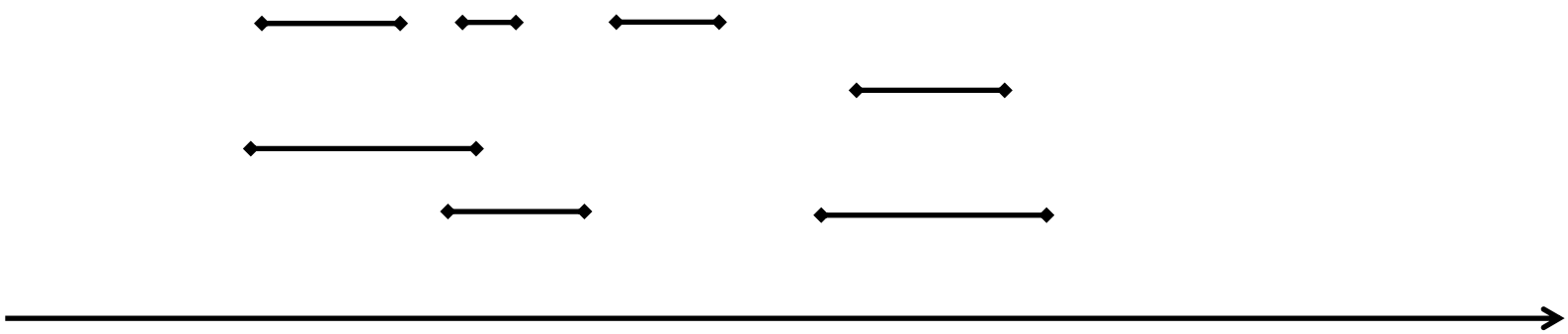
Optimalität ?



Intervall Scheduling

Strategie 2:

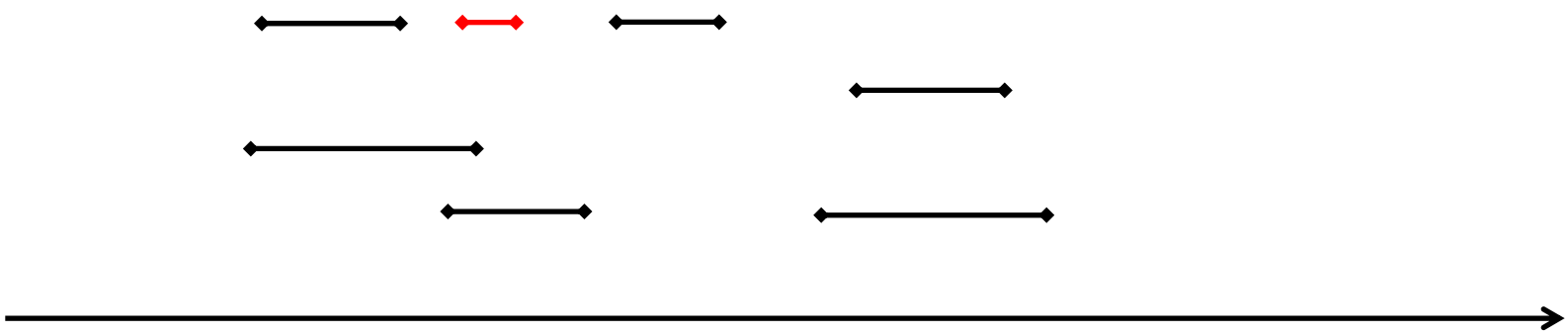
- Wähle immer das kürzeste Intervall



Intervall Scheduling

Strategie 2:

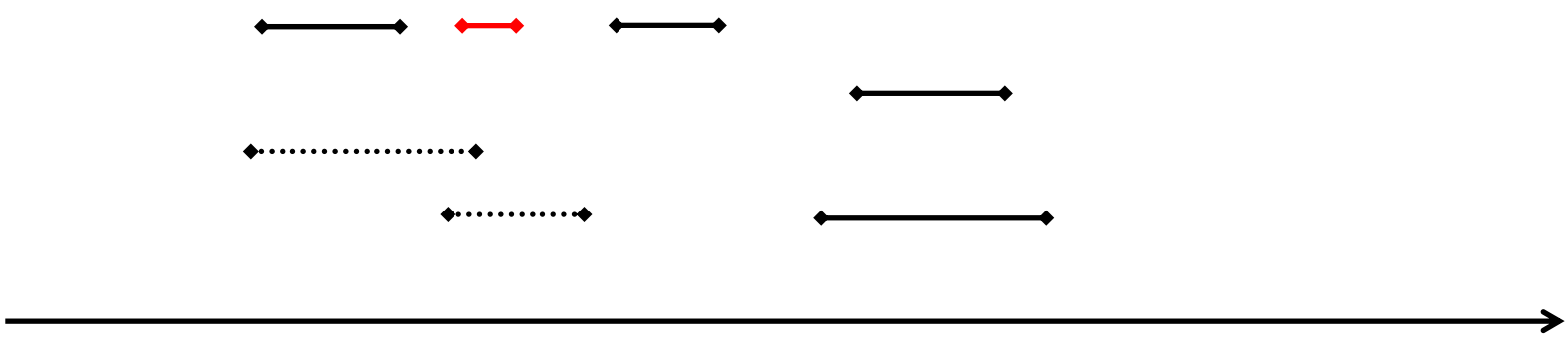
- Wähle immer das kürzeste Intervall



Intervall Scheduling

Strategie 2:

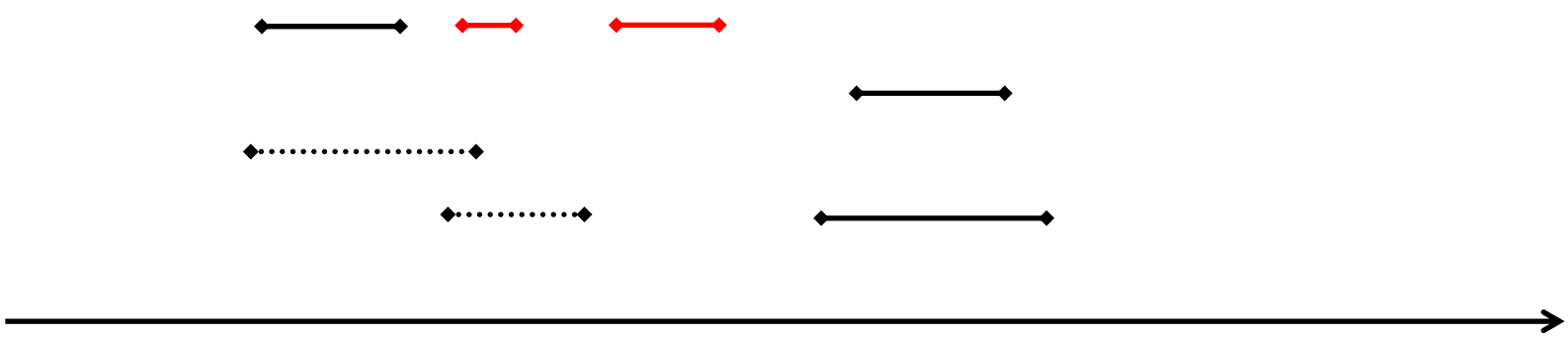
- Wähle immer das kürzeste Intervall



Intervall Scheduling

Strategie 2:

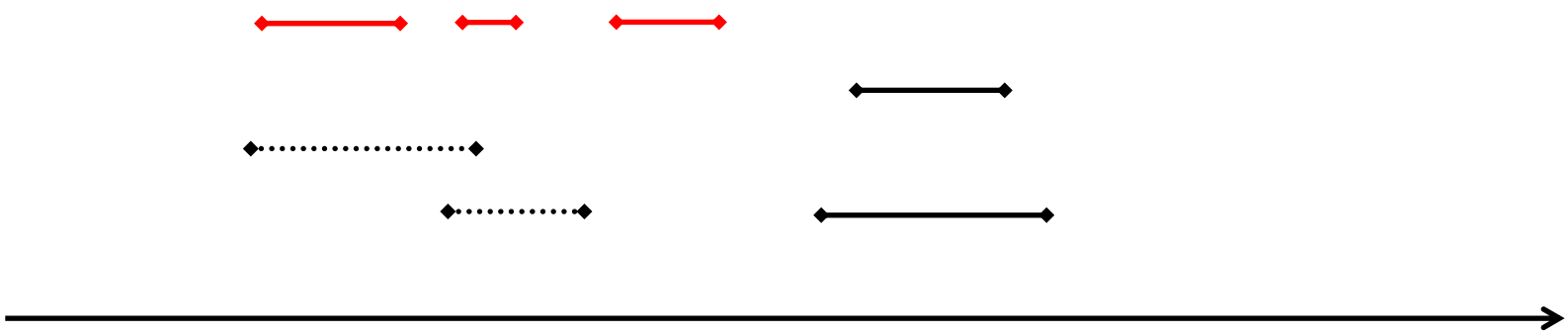
- Wähle immer das kürzeste Intervall



Intervall Scheduling

Strategie 2:

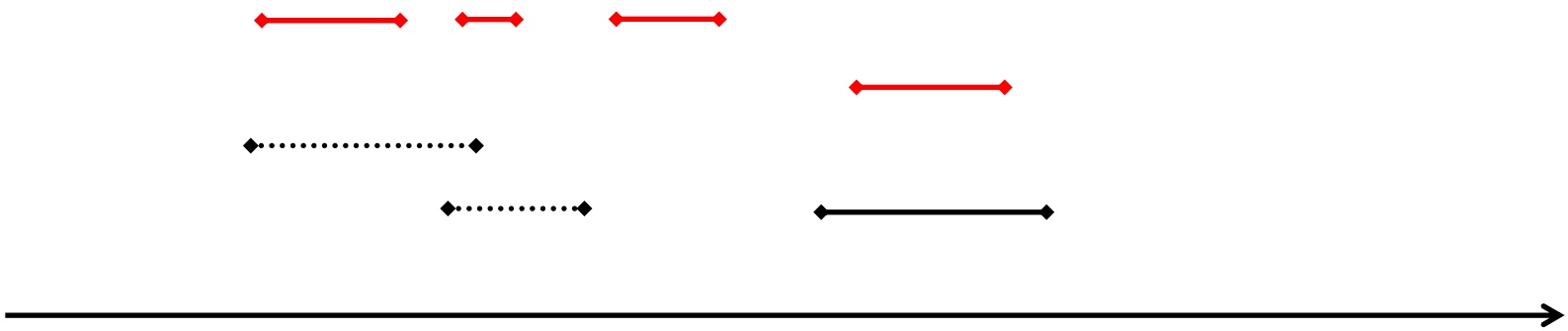
- Wähle immer das kürzeste Intervall



Intervall Scheduling

Strategie 2:

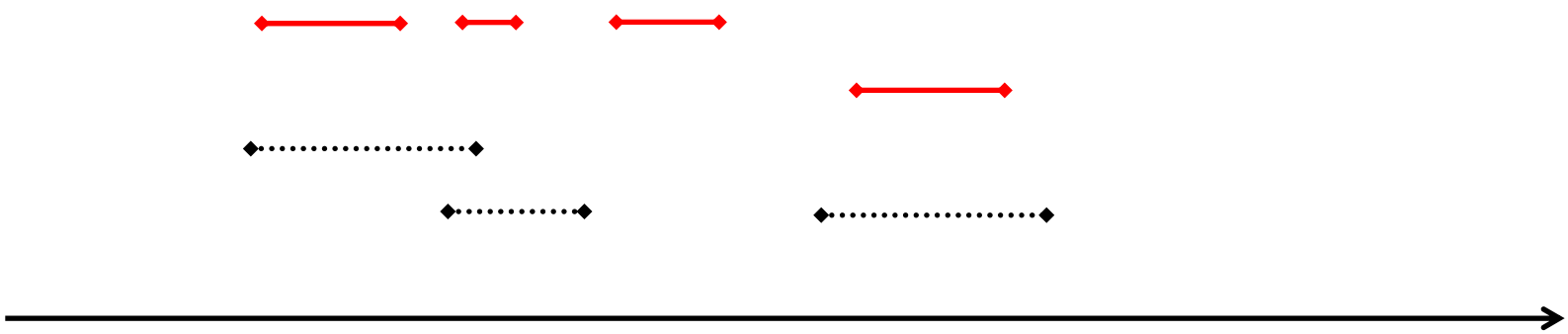
- Wähle immer das kürzeste Intervall



Intervall Scheduling

Strategie 2:

- Wähle immer das kürzeste Intervall

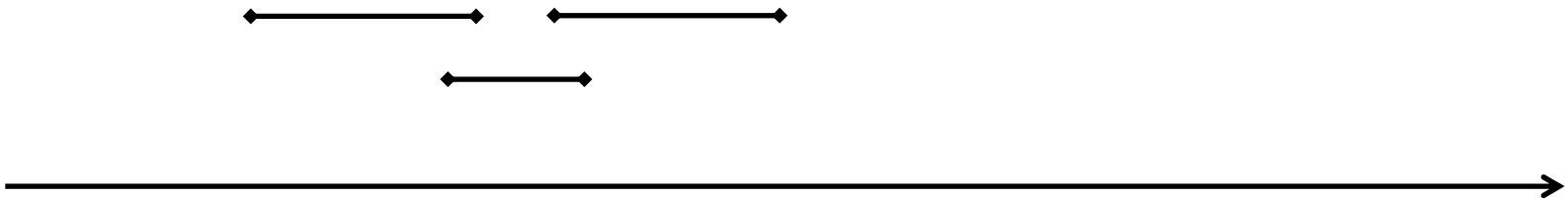


Intervall Scheduling

Strategie 2:

- Wähle immer das kürzeste Intervall

Optimalität?

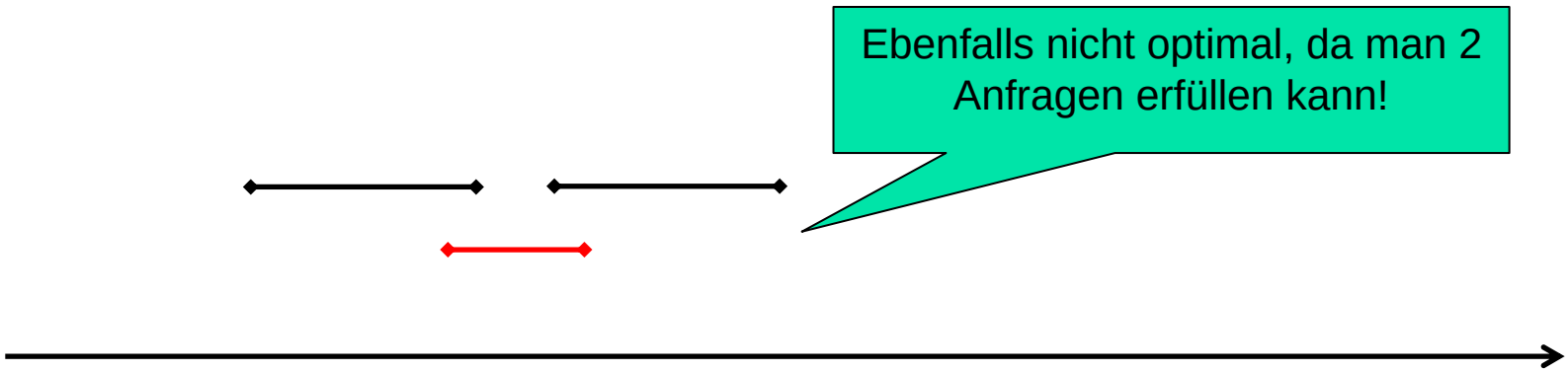


Intervall Scheduling

Strategie 2:

- Wähle immer das kürzeste Intervall

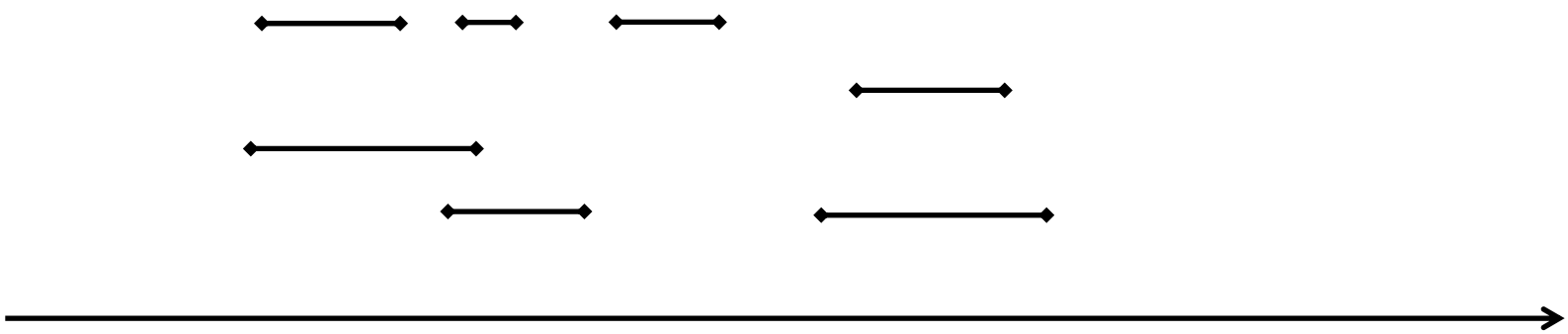
Optimalität?



Intervall Scheduling

Strategie 3:

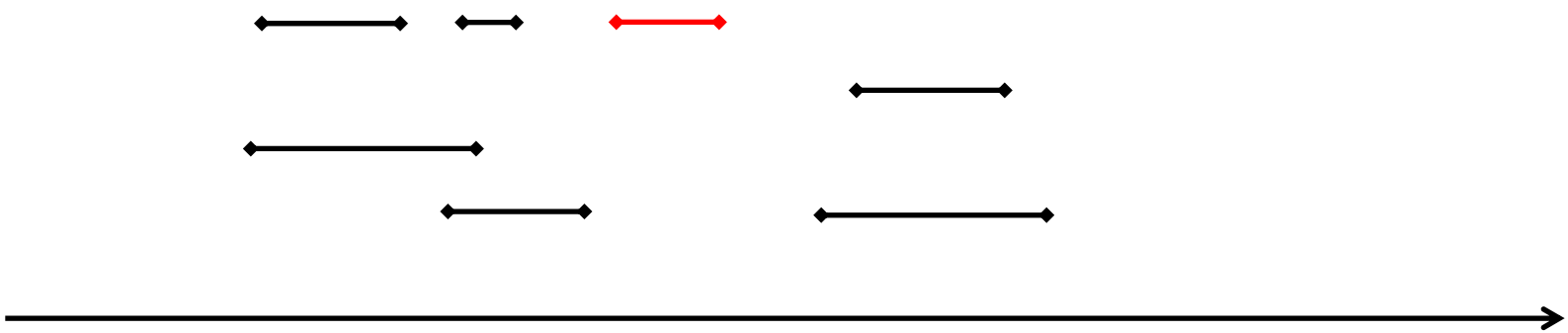
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

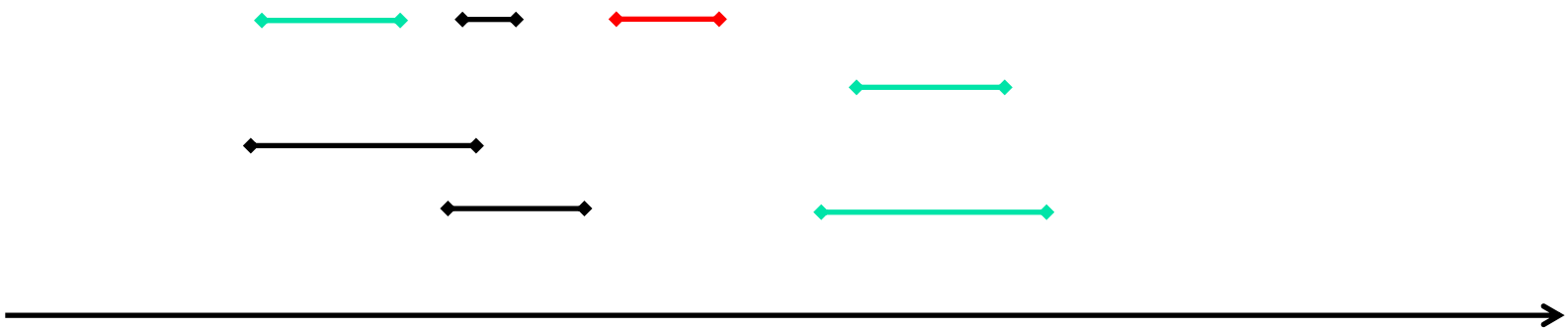


40

Intervall Scheduling

Strategie 3:

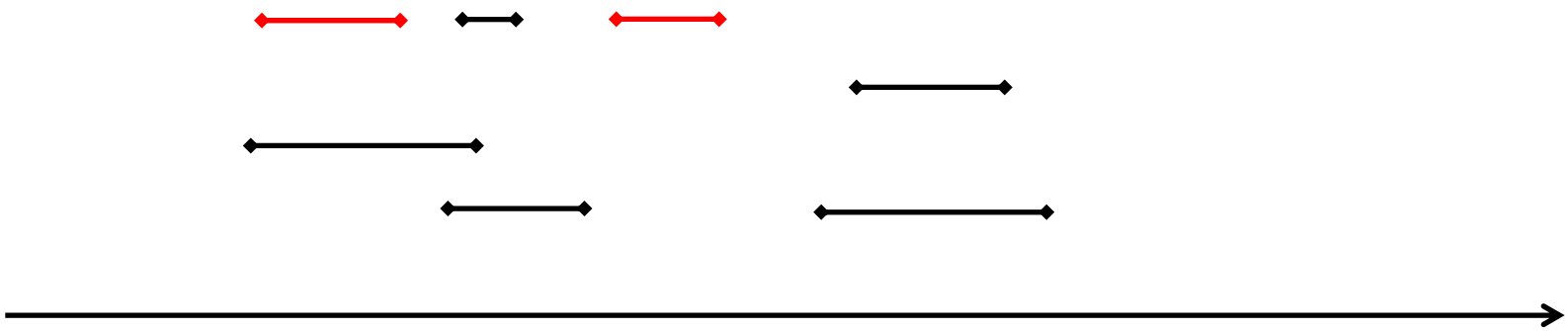
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

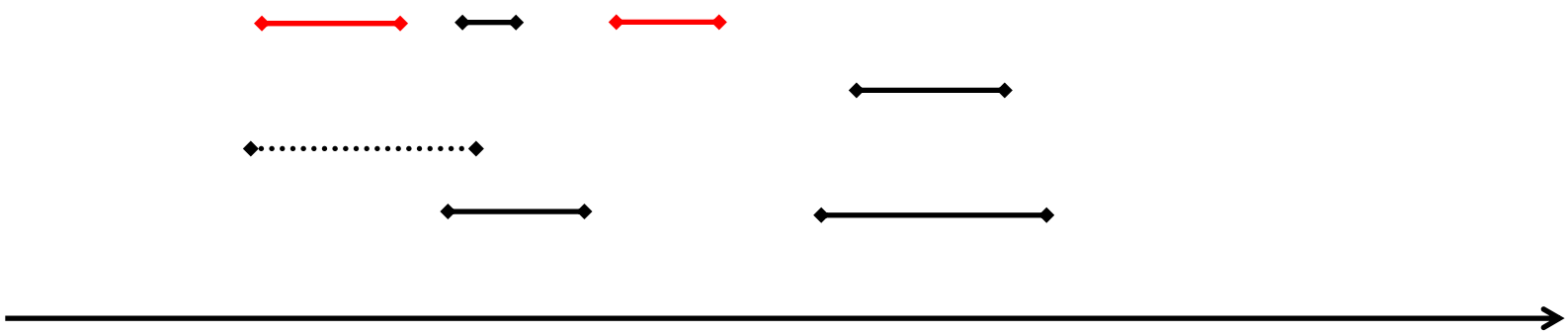
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

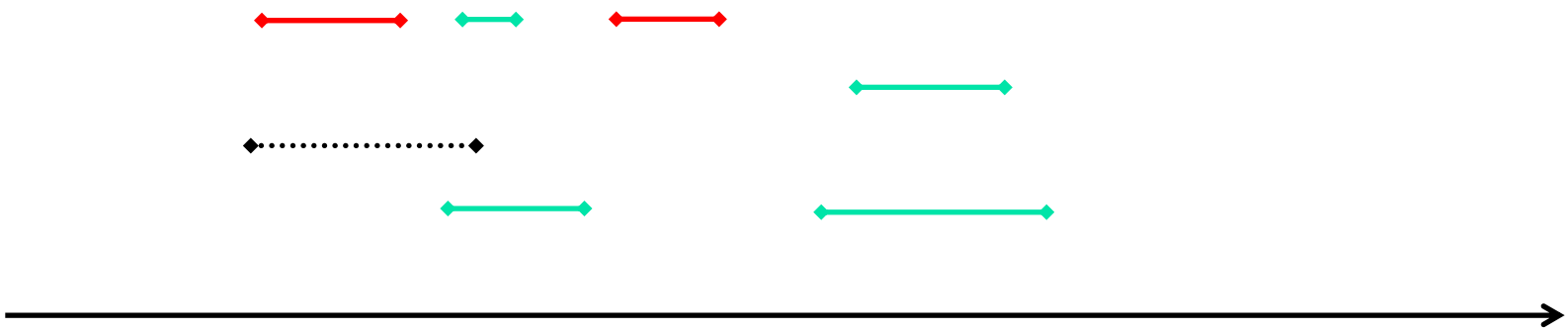
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

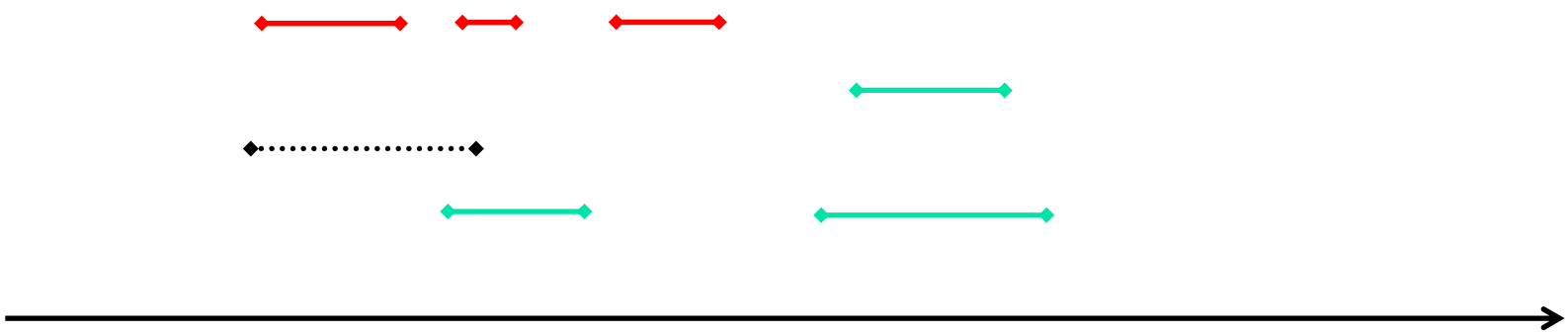
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

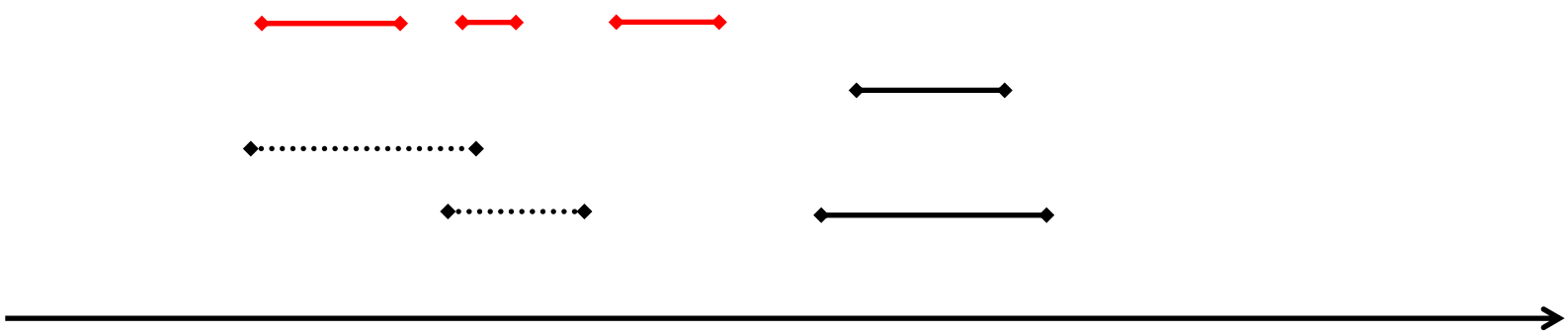
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

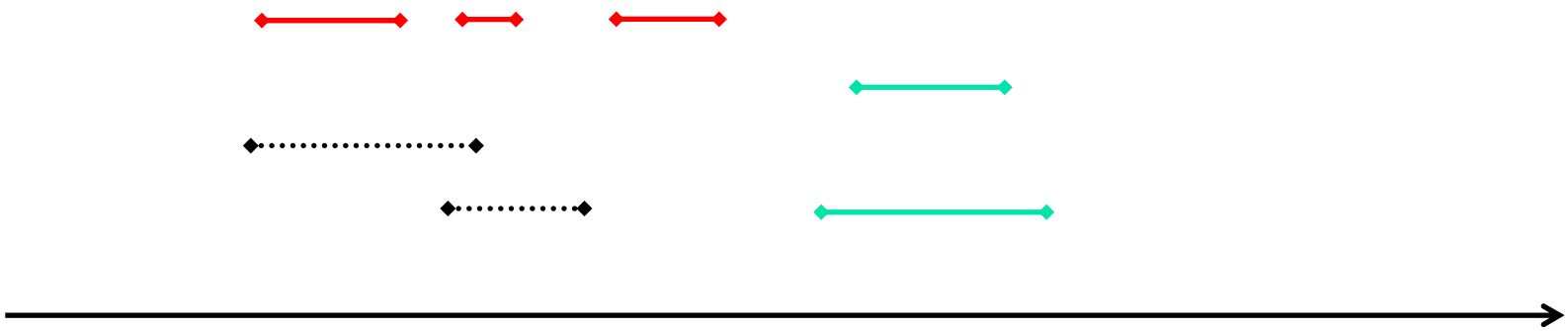
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

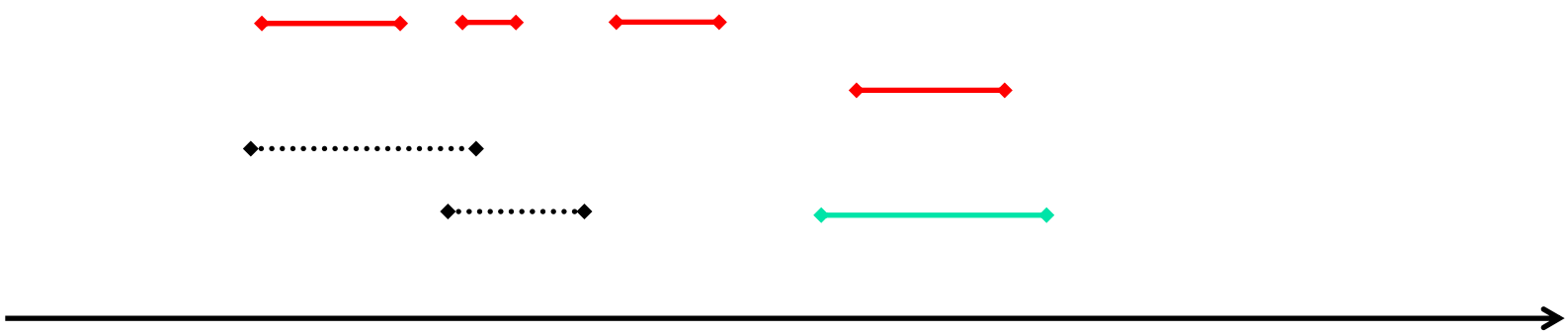
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

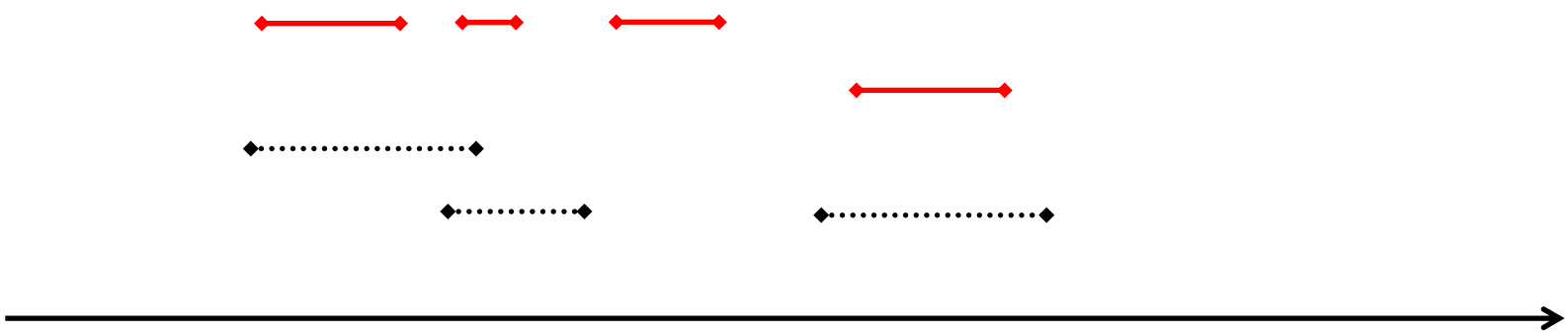
- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall



Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

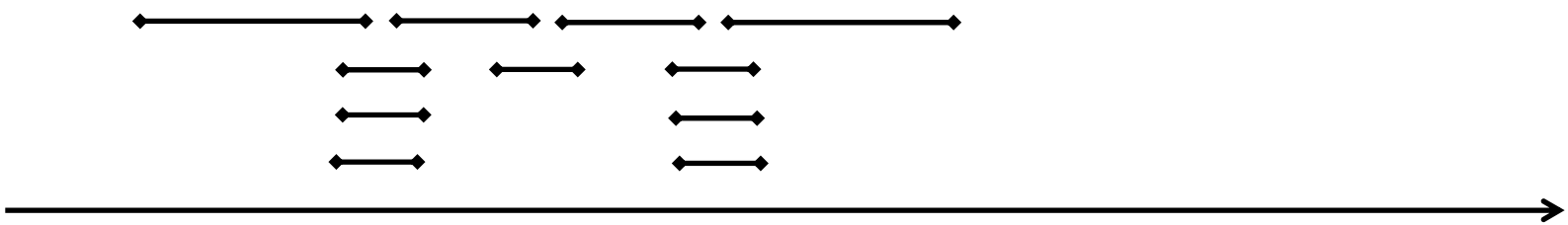


Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

Optimalität?

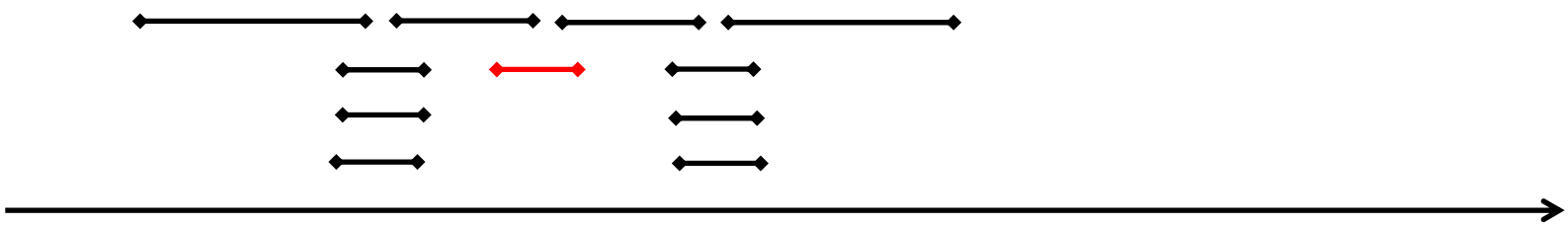


Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

Optimalität?

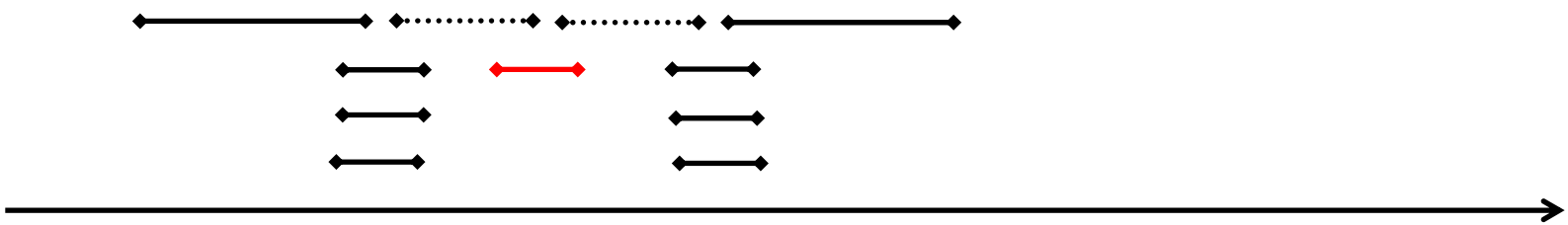


Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

Optimalität?

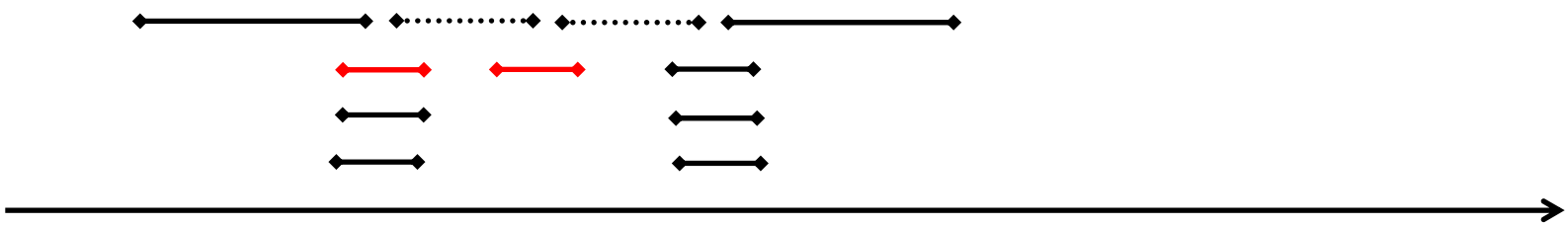


Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

Optimalität?

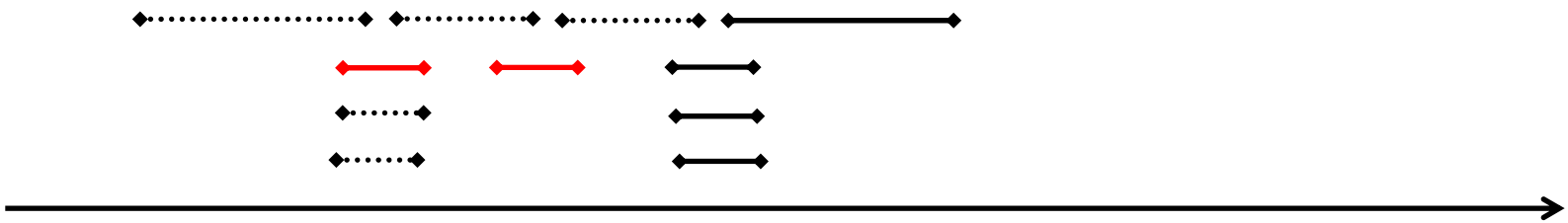


Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

Optimalität?

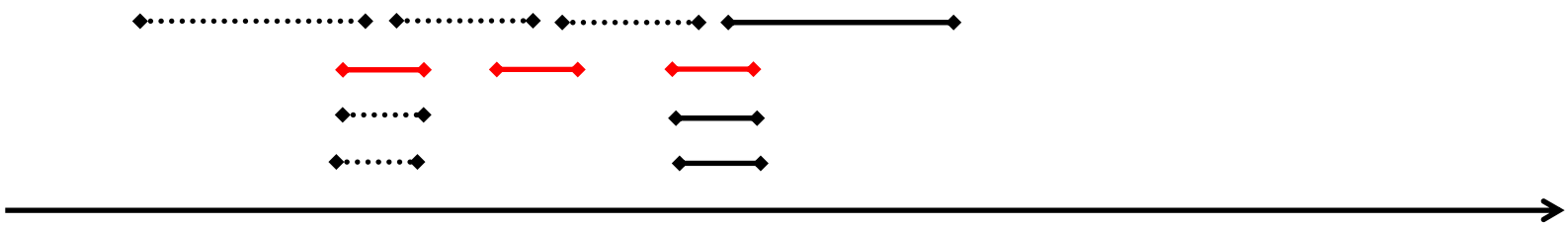


Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

Optimalität?

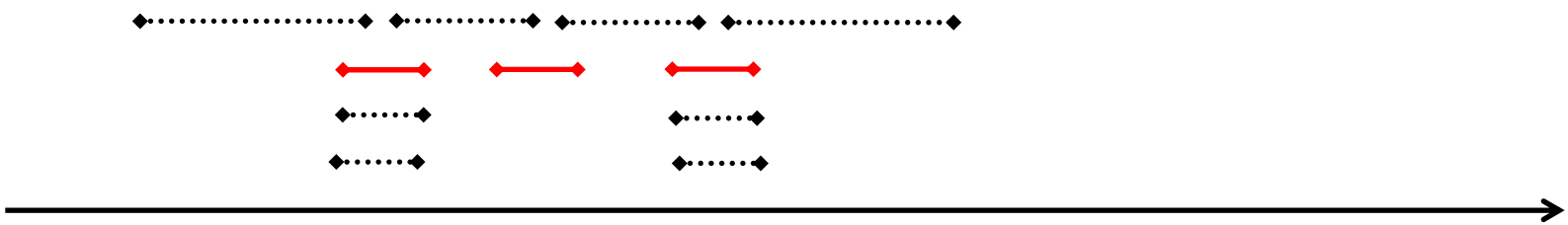


Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

Optimalität?

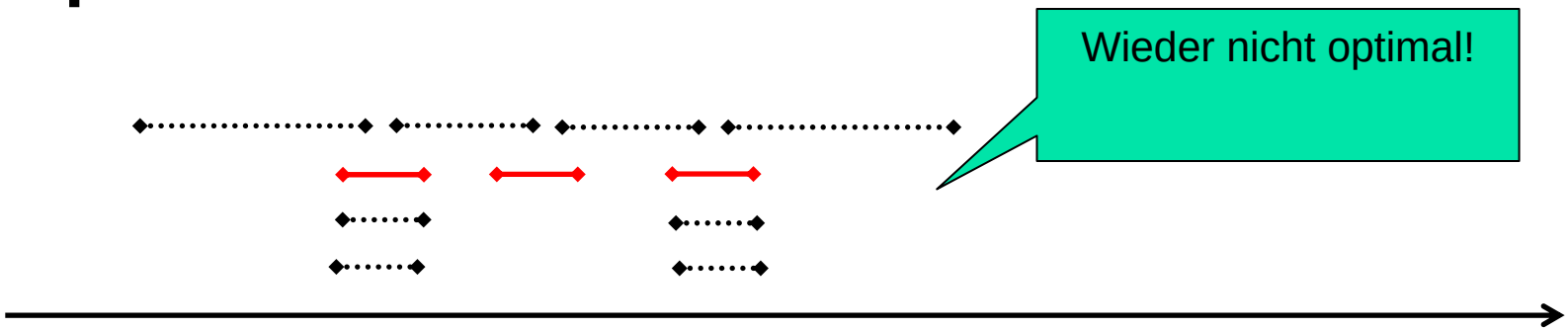


Intervall Scheduling

Strategie 3:

- Wähle immer das Intervall mit den wenigsten nicht kompatiblen Intervallen
- bei Gleichheit wähle das kürzeste Intervall

Optimalität?



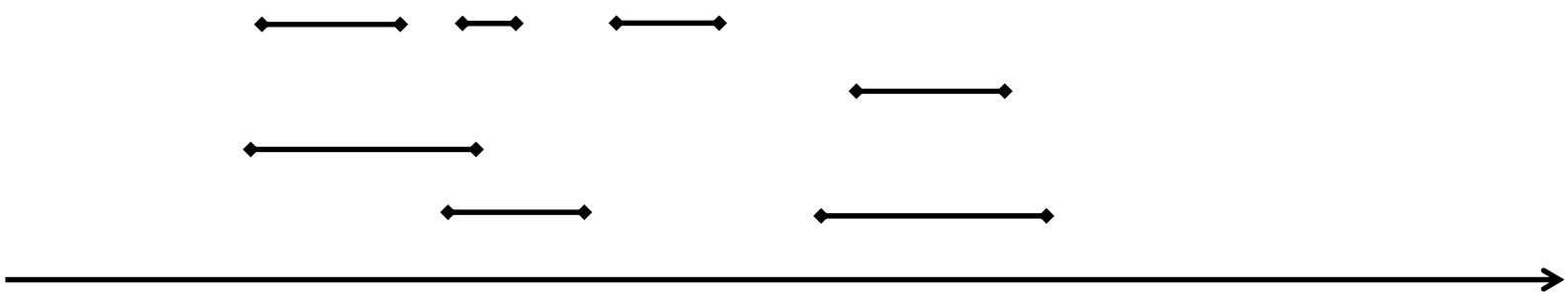
Intervall Scheduling

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



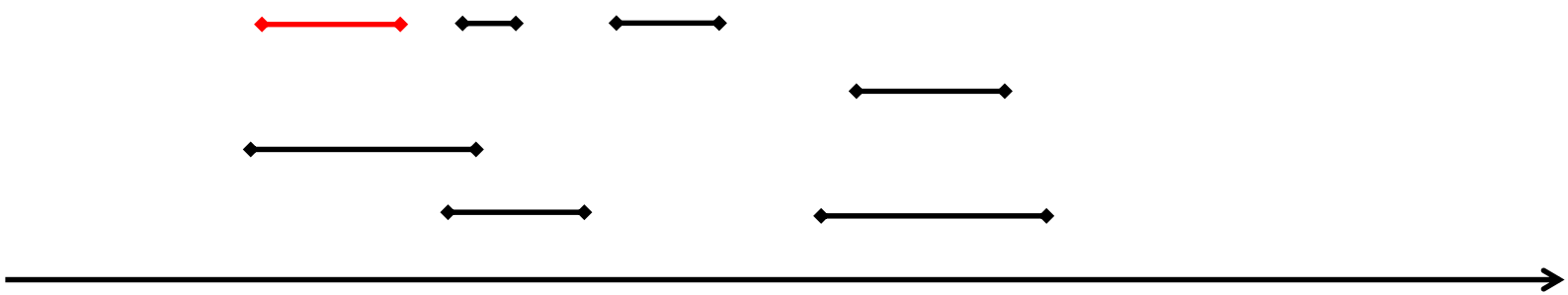
Intervall Scheduling

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



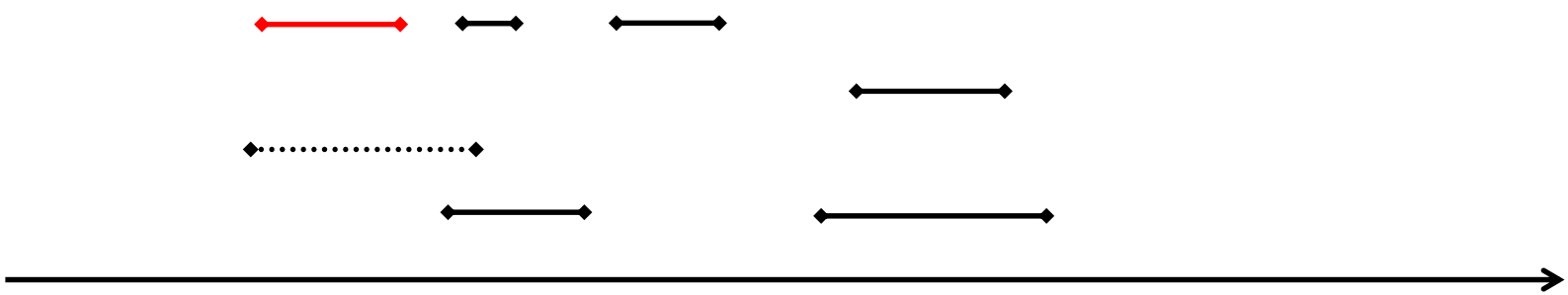
Intervall Scheduling

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



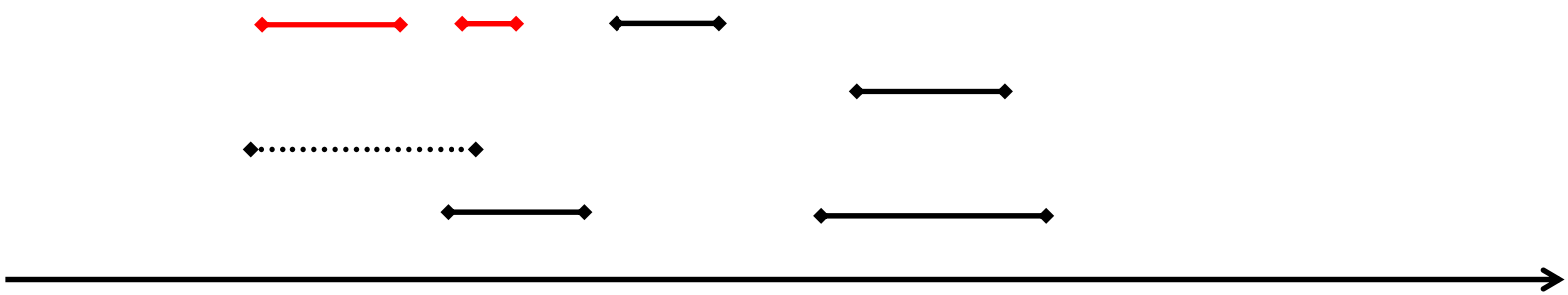
Intervall Scheduling

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



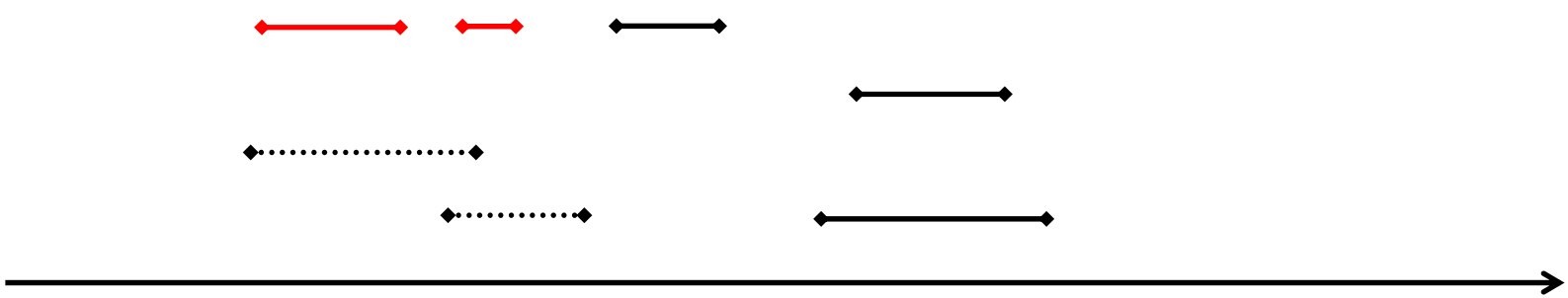
Intervall Scheduling

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



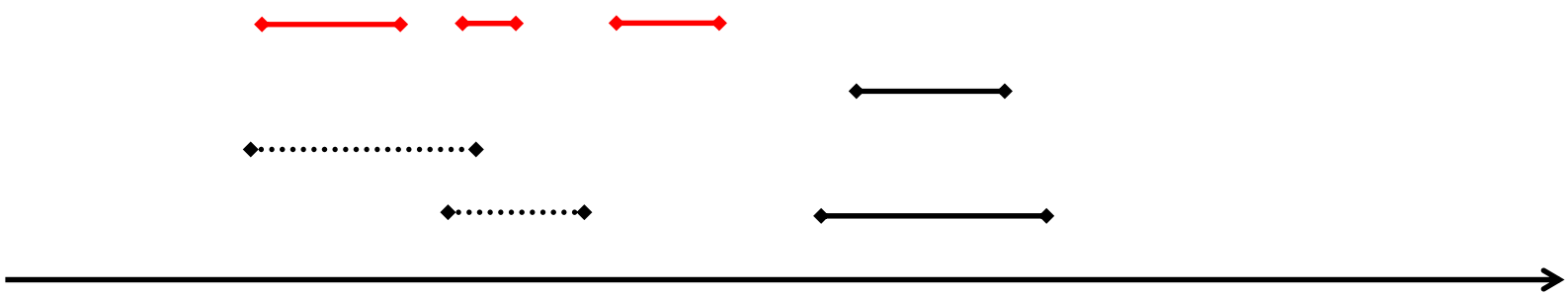
Intervall Scheduling

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



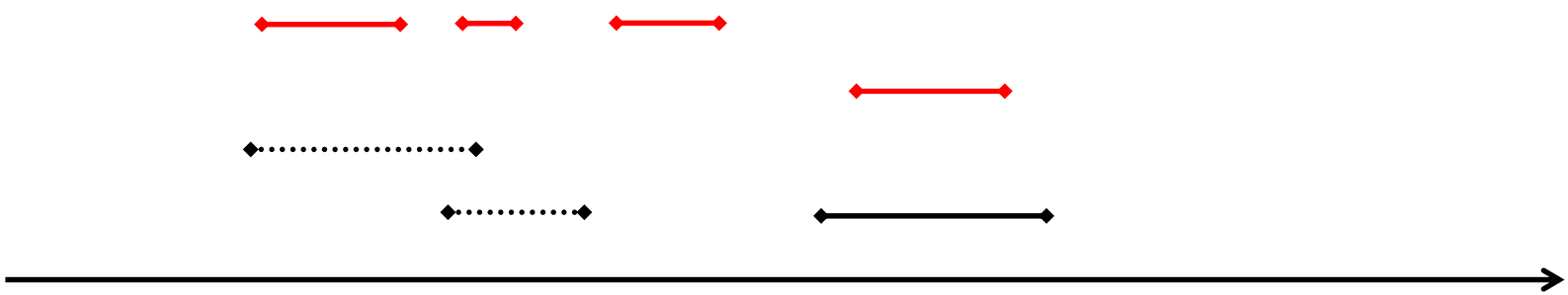
Intervall Scheduling

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



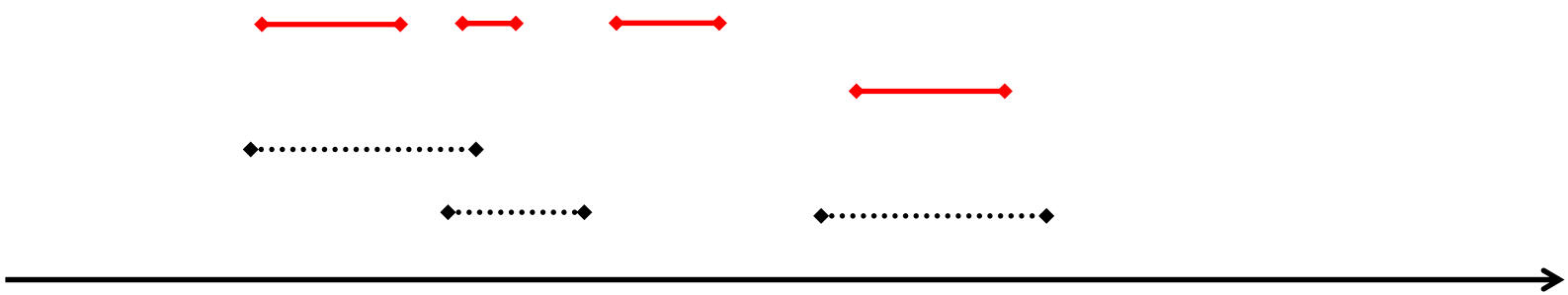
Intervall Scheduling

Worauf muss man achten?

- Resource muss möglichst früh wieder frei werden!

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



Intervall Scheduling

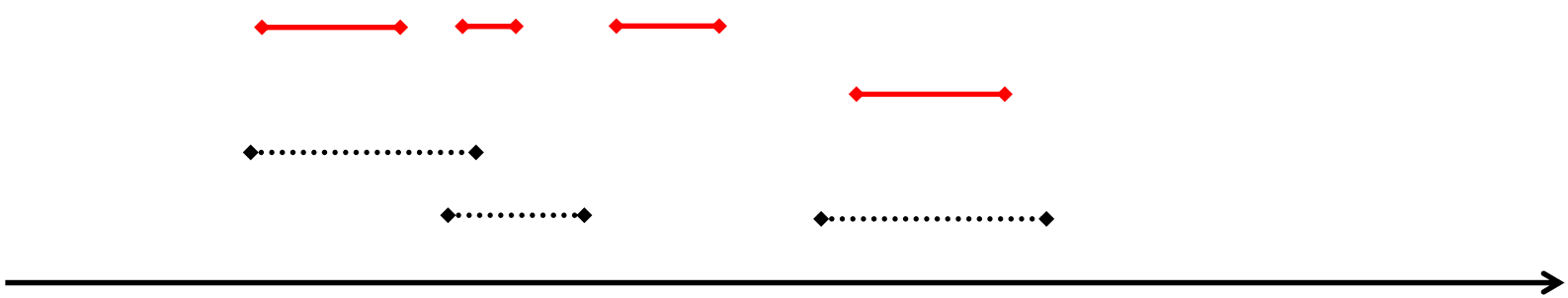
Worauf muss man achten?

- Resource muss möglichst früh wied

Diese Strategie ist optimal! Aber wie beweist man das?

Neue Strategie:

- Nimm die Anfrage, die am frühesten fertig ist.



Intervall Scheduling

Formale Problemformulierung:

- Problem: Intervall Scheduling
- Eingabe: Felder s und f, die die Intervalle (s[i], f[i]) beschreiben
- Ausgabe: Indizes der ausgewählten Intervalle

Wichtige Annahme:

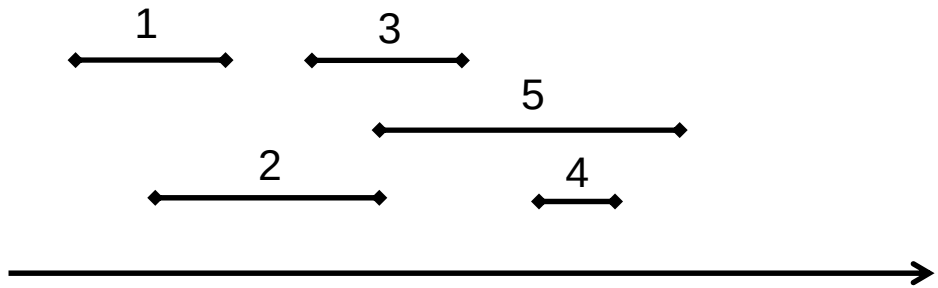
- Eingabe nach Intervallendpunkten sortiert, d.h.
- $f[1] \leq f[2] \leq \ldots \leq f[n]$

Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
7. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

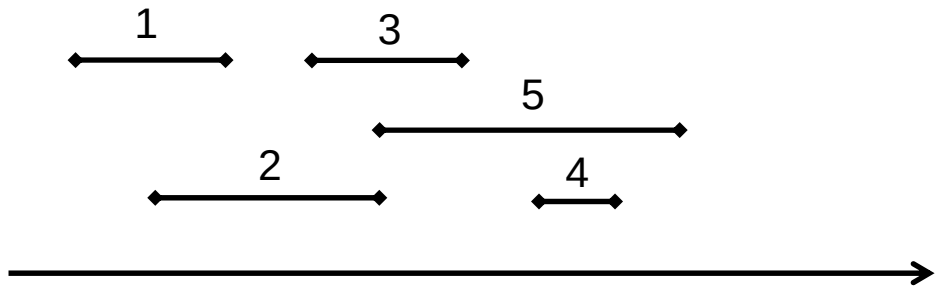


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

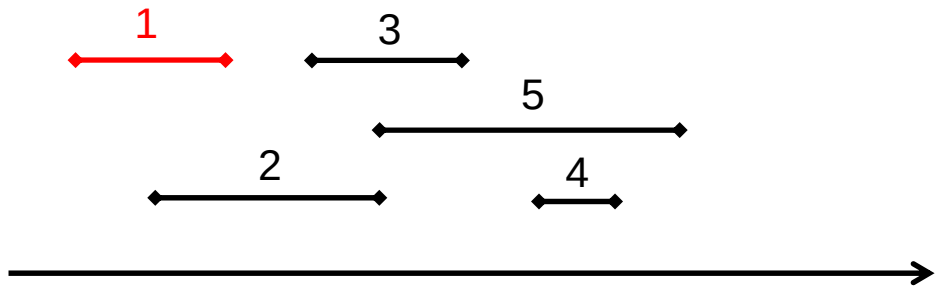


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

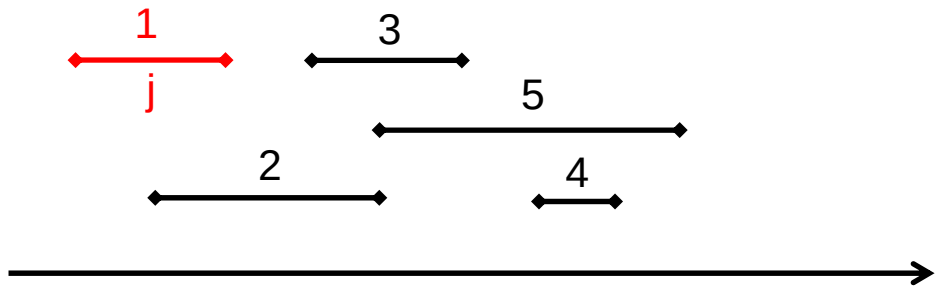


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

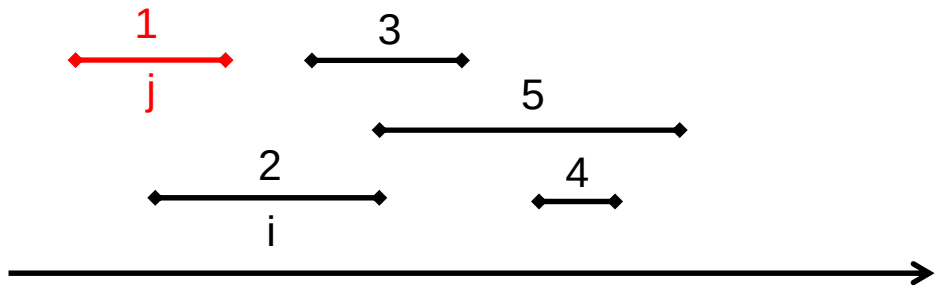


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

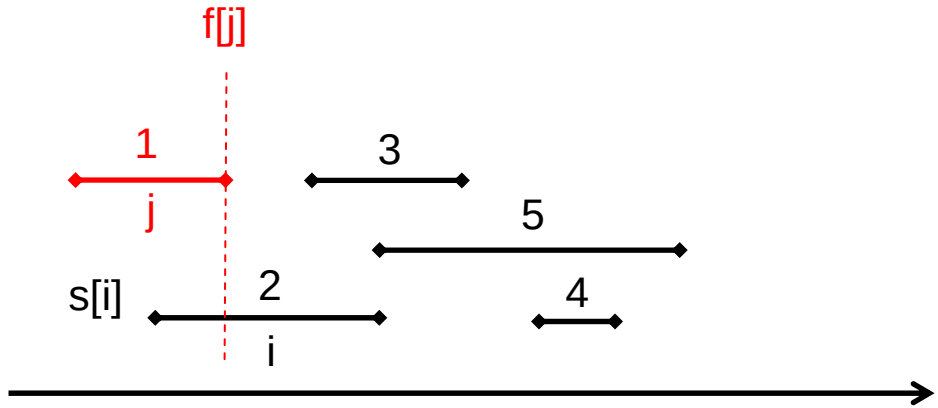


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

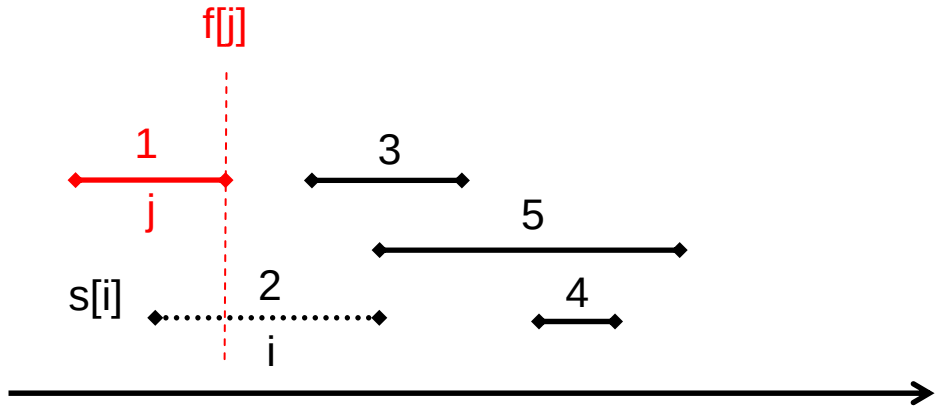


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

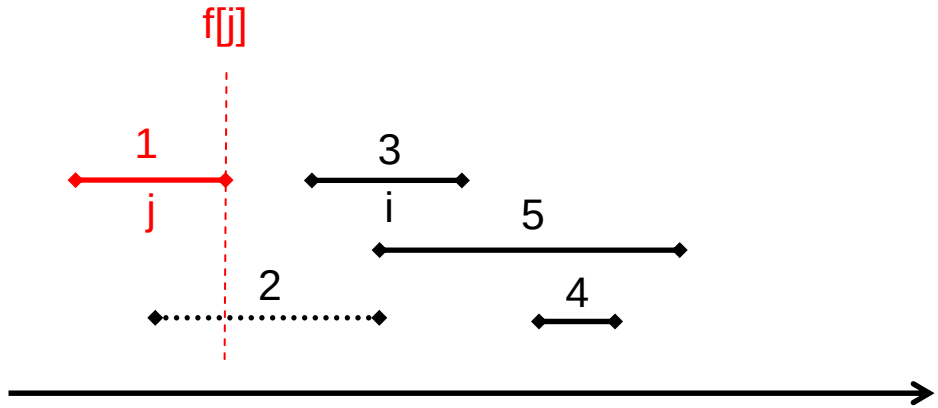


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

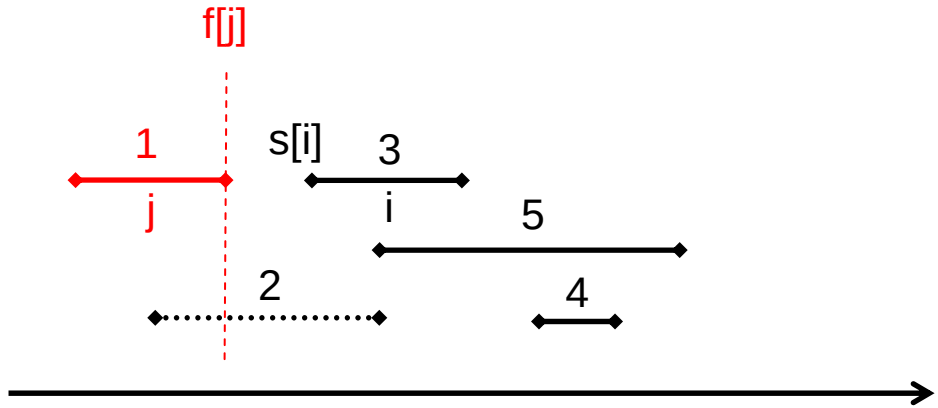


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

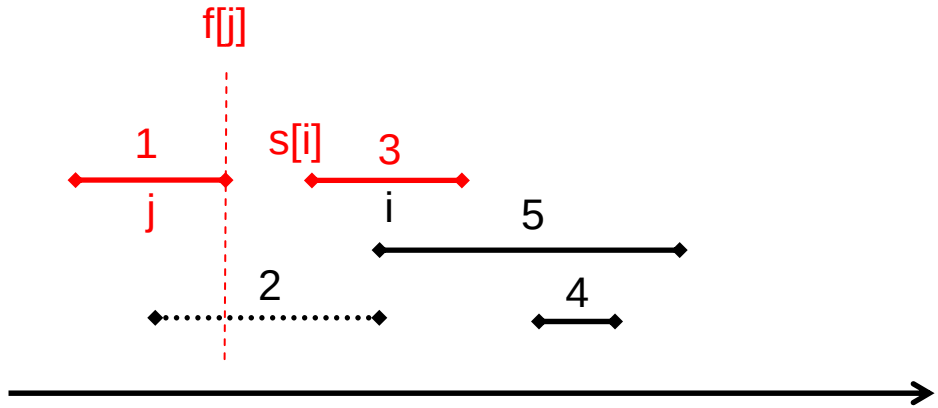


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

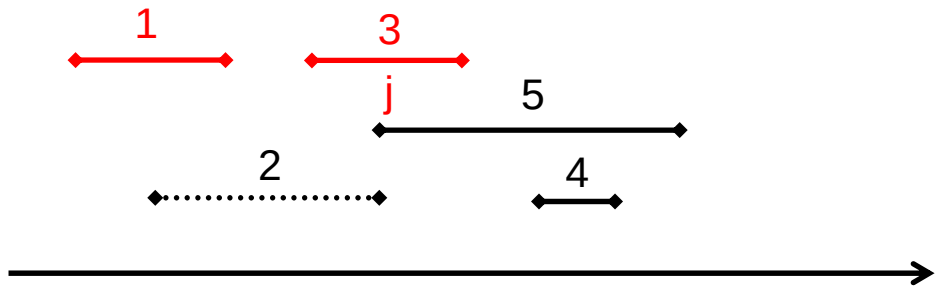


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

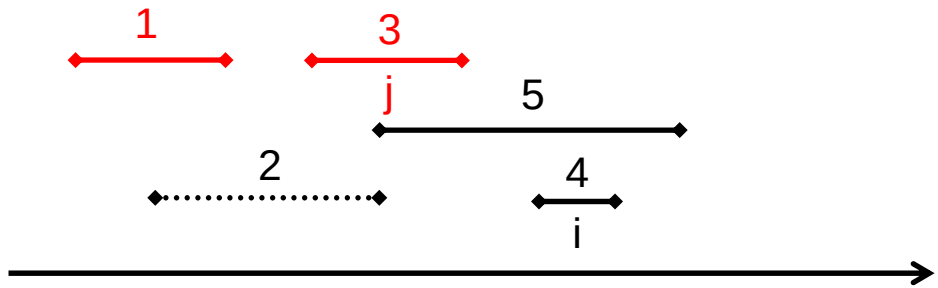


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

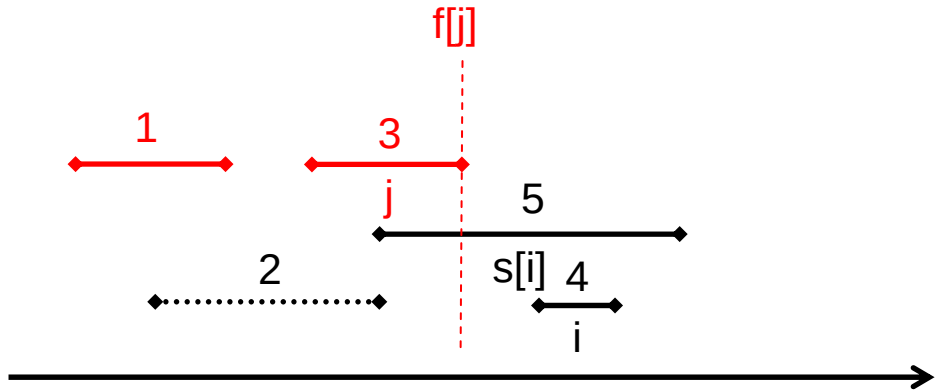


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

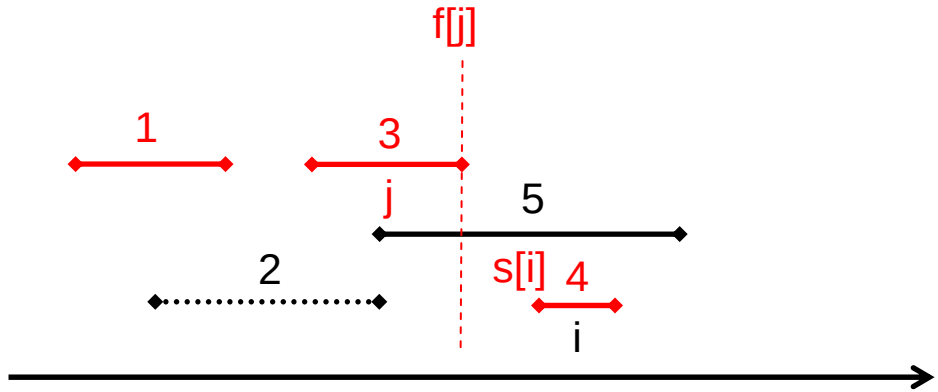


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

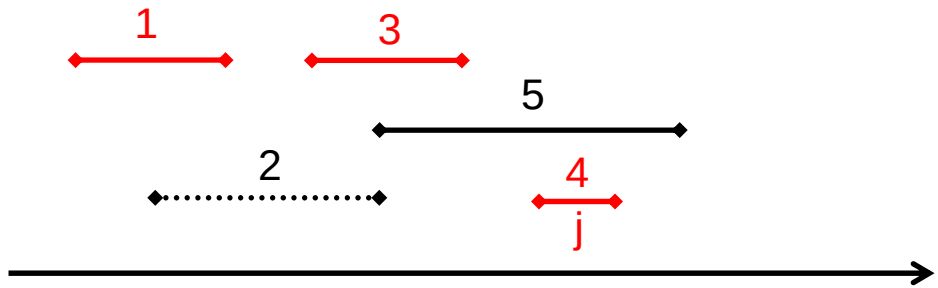


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

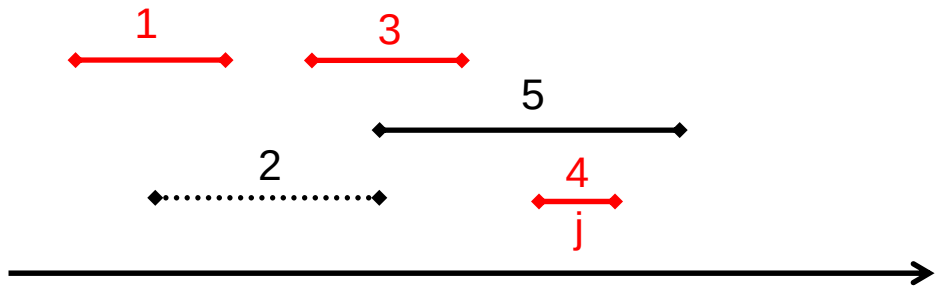


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

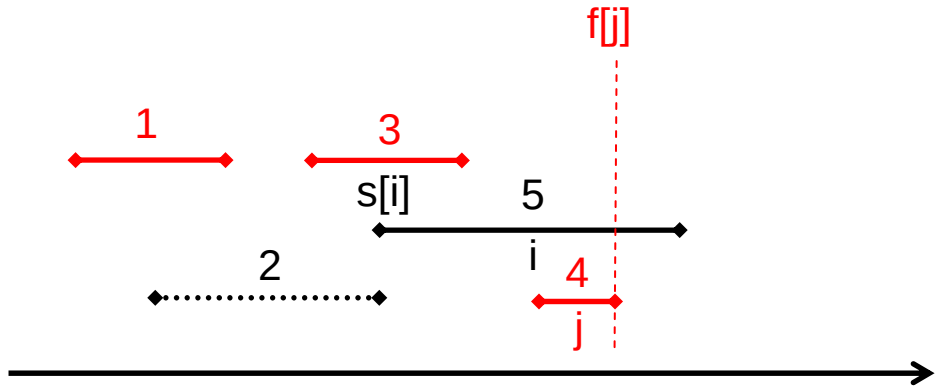


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

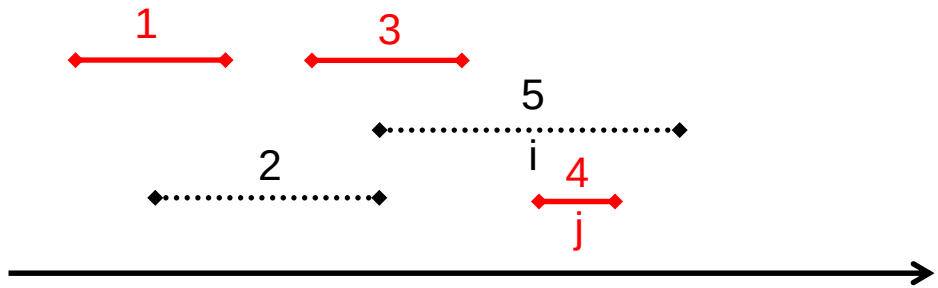


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9

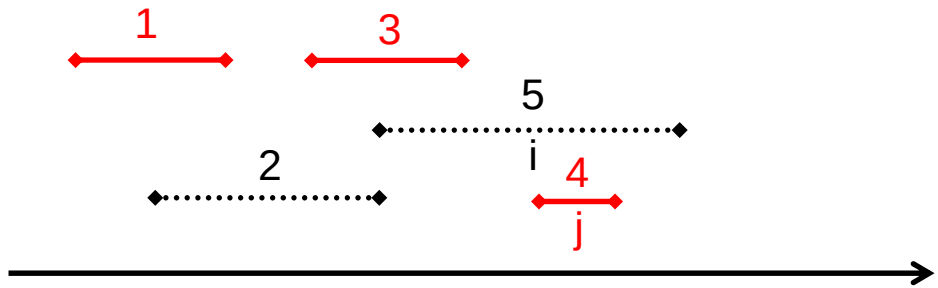


Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

s	1	2	4	7	5
f	3	5	6	8	9



Intervall Scheduling

Beweisidee: Der gierige Algorithmus „liegt vorn“

- Wir messen „Fortschritt“ des Algorithmus Schritt für Schritt
- Zeige: Der gierige Algorithmus macht mindestens genau so viel Fortschritt wie jeder beliebige andere Algorithmus

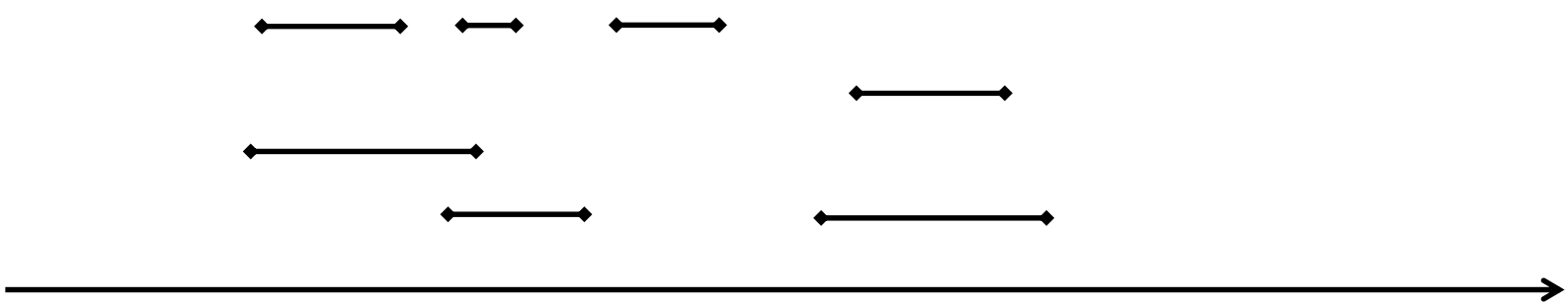
Beobachtung:

A ist eine Menge von kompatiblen Anfrage.

Intervall Scheduling

Wie können wir Optimalität zeigen?

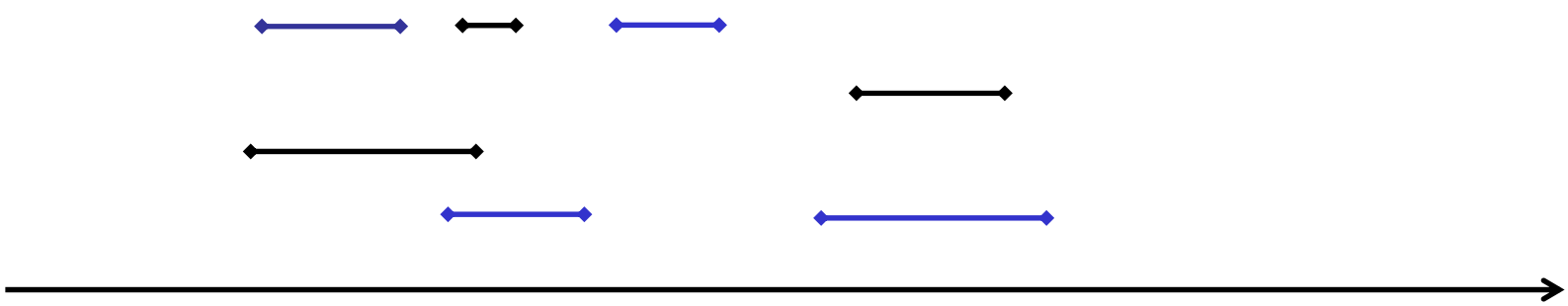
- Sei O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Intervall Scheduling

Wie können wir Optimalität zeigen?

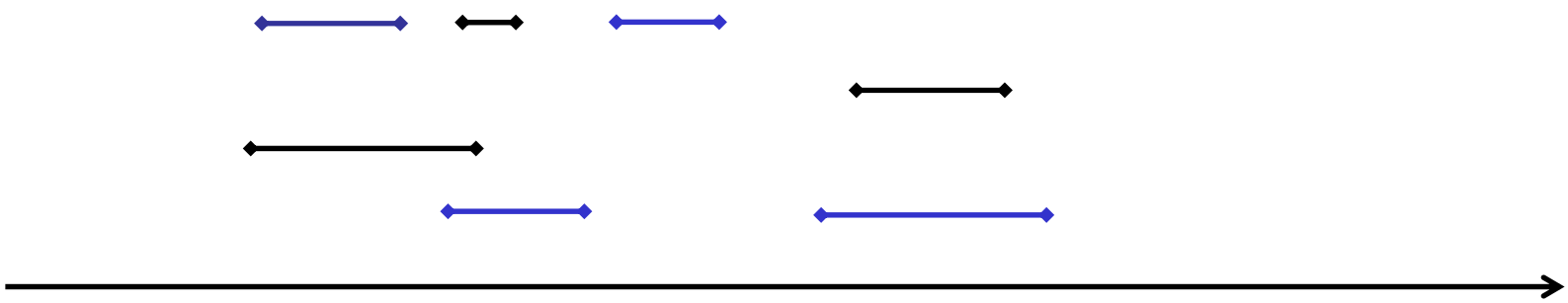
- Sei O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Intervall Scheduling

Wie können wir Optimalität zeigen?

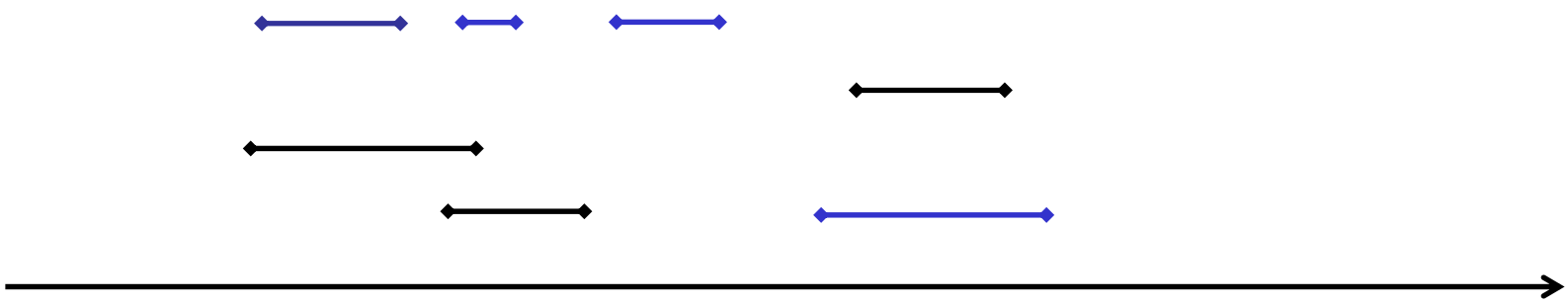
- Sei O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Intervall Scheduling

Wie können wir Optimalität zeigen?

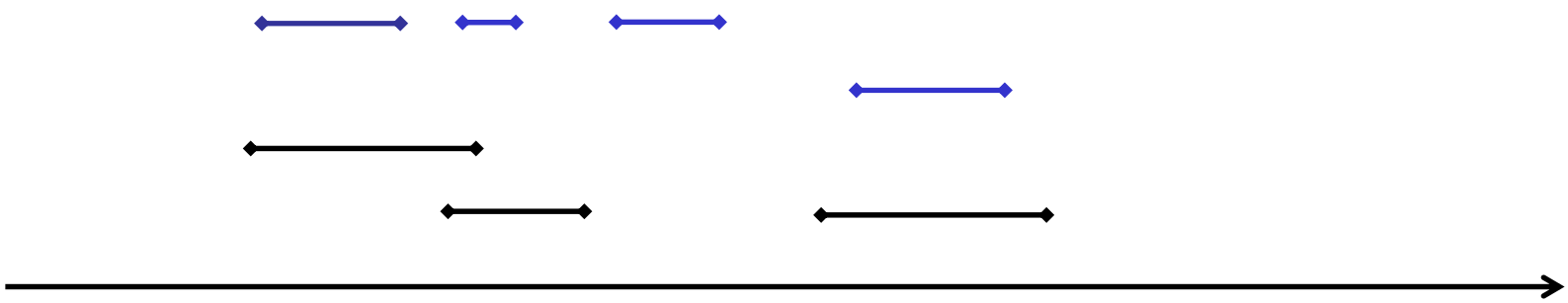
- Sei O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Intervall Scheduling

Wie können wir Optimalität zeigen?

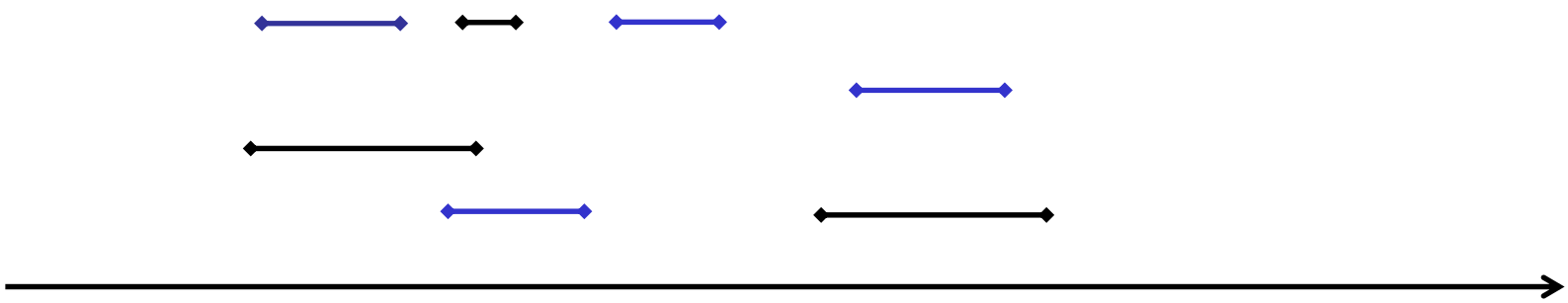
- Sei O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Intervall Scheduling

Wie können wir Optimalität zeigen?

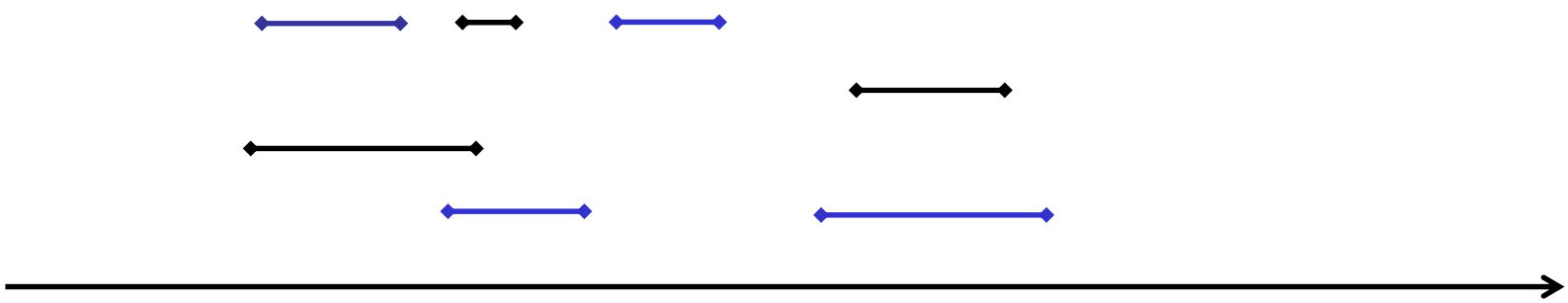
- Sei O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Intervall Scheduling

Wie können wir Optimalität zeigen?

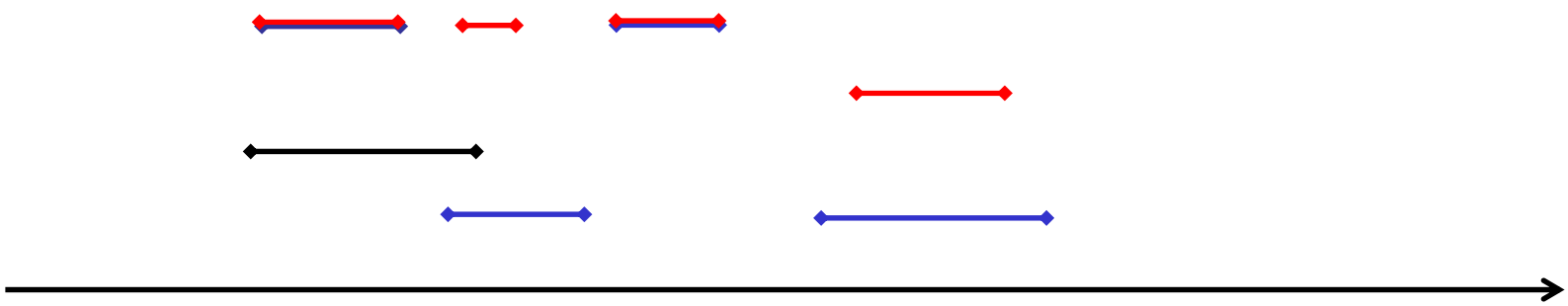
- Sei O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Intervall Scheduling

Wie können wir Optimalität zeigen?

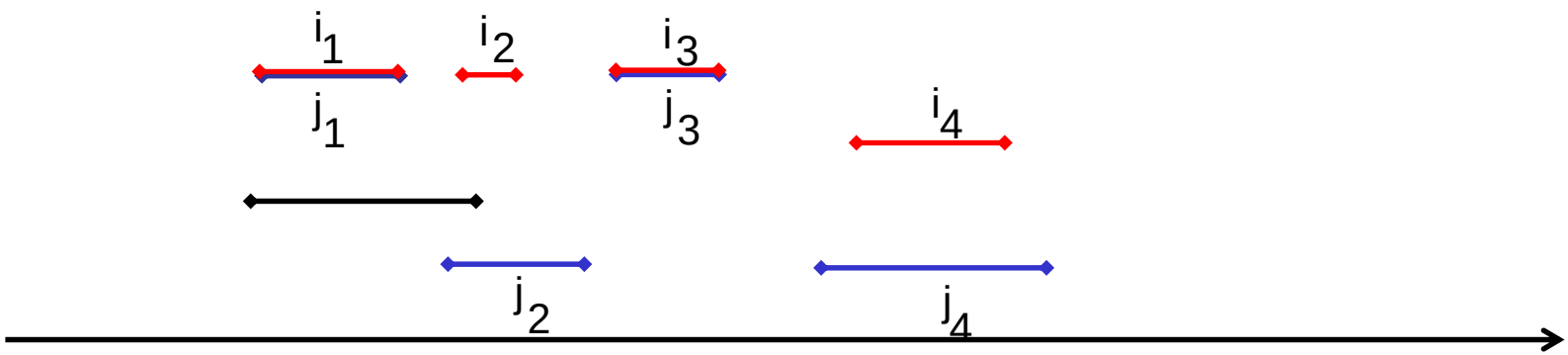
- Sei O optimale Menge von Intervallen
- u. U. viele optimale Lösungen
- Wir zeigen: |A| = |O|



Intervall Scheduling

Notation:

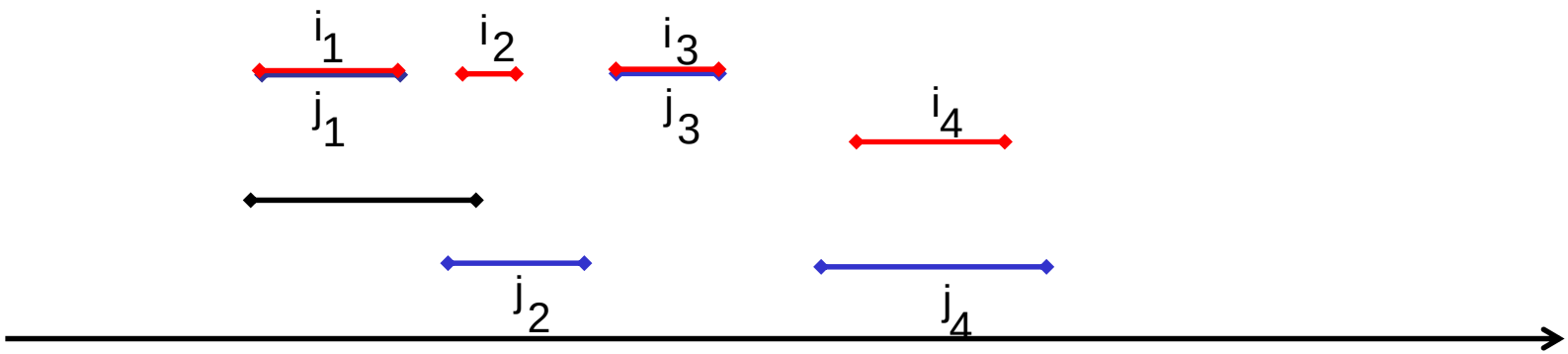
- $i_1, \ldots, i_k$ Intervalle von A in Ordnung des Hinzufügen
- $j_1, \ldots, j_m$ Intervalle von O sortiert nach Endpunkt
- Zu zeigen: k = m



Intervall Scheduling

Der gierige Algorithmus liegt vorn:

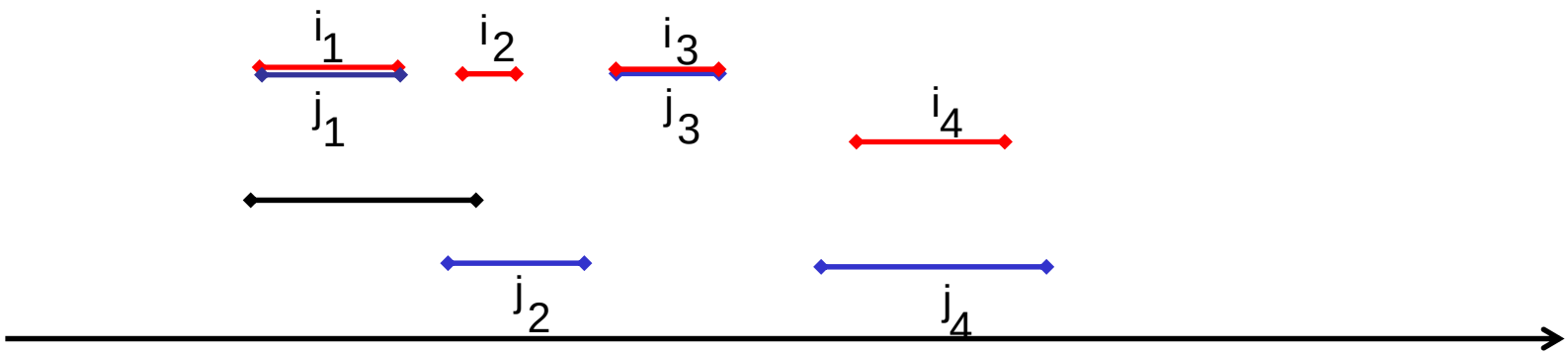
- Idee des Algorithmus: Die Resource soll so früh wie möglich wieder frei werden
- Dies ist war für das erste Interval: $f[i_1] \leq f[j_1]$
- Zu zeigen: Gilt für alle Intervalle



Intervall Scheduling

Lemma:

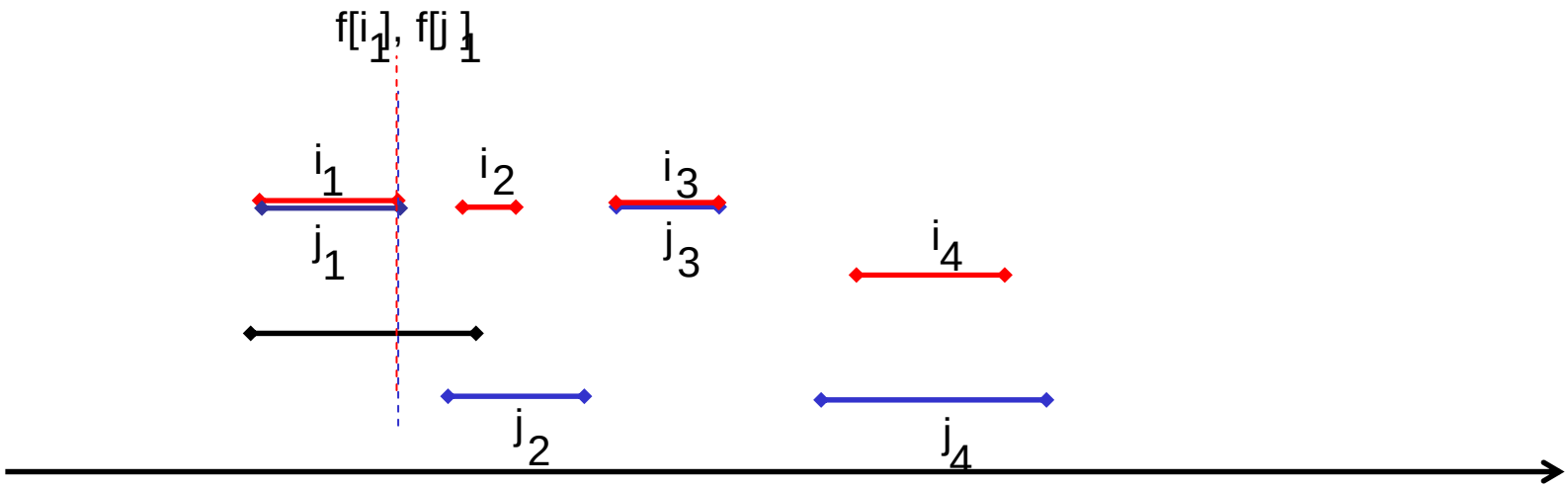
Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.



Intervall Scheduling

Lemma:

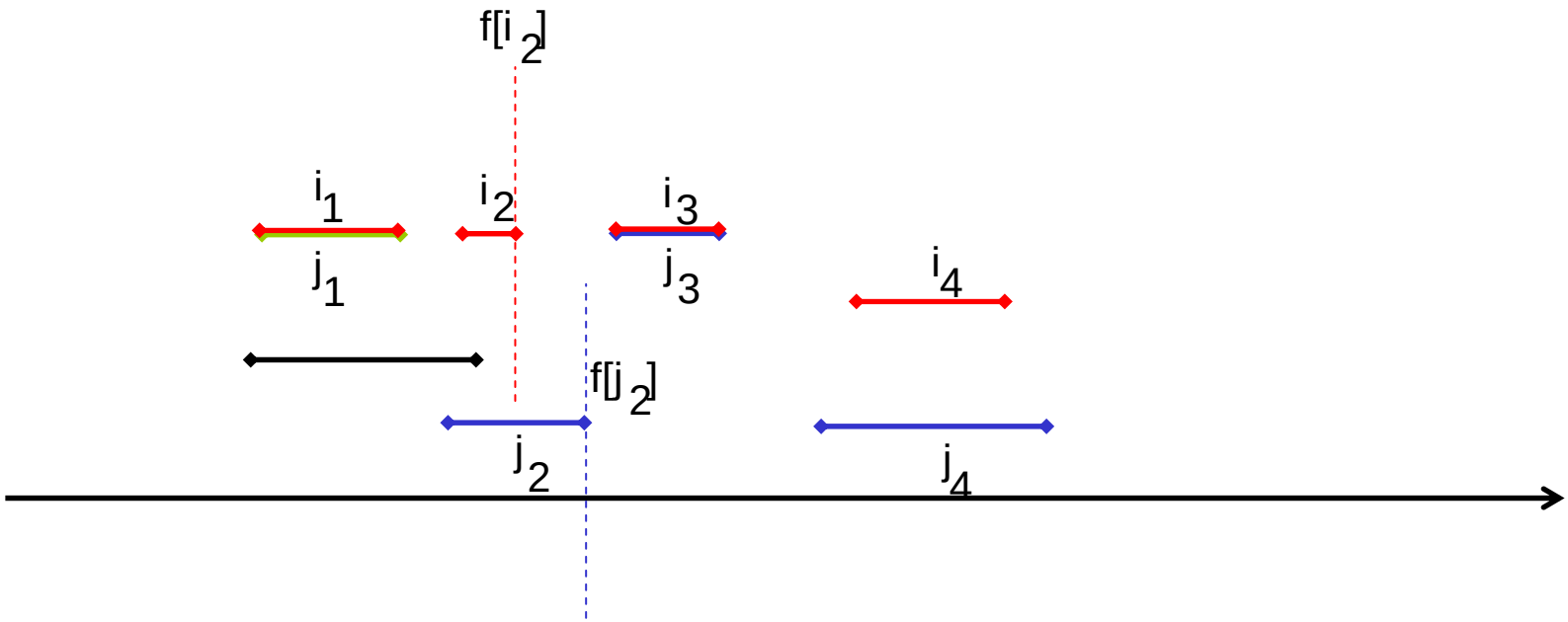
Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.



Intervall Scheduling

Lemma:

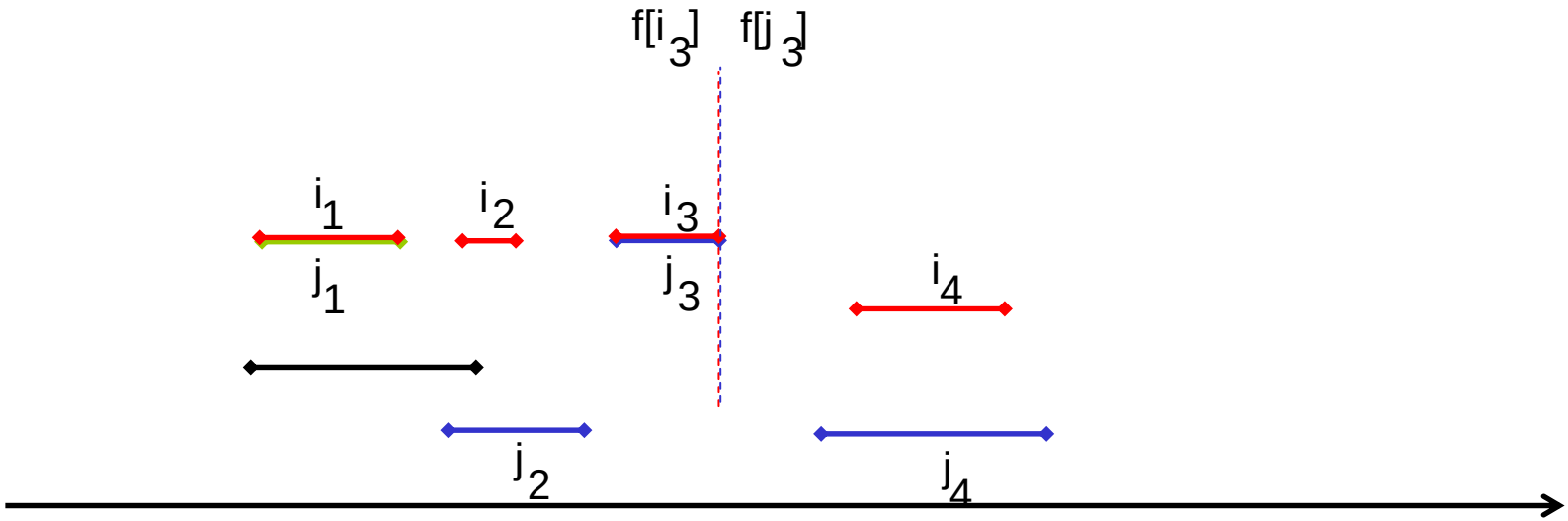
Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.



Intervall Scheduling

Lemma:

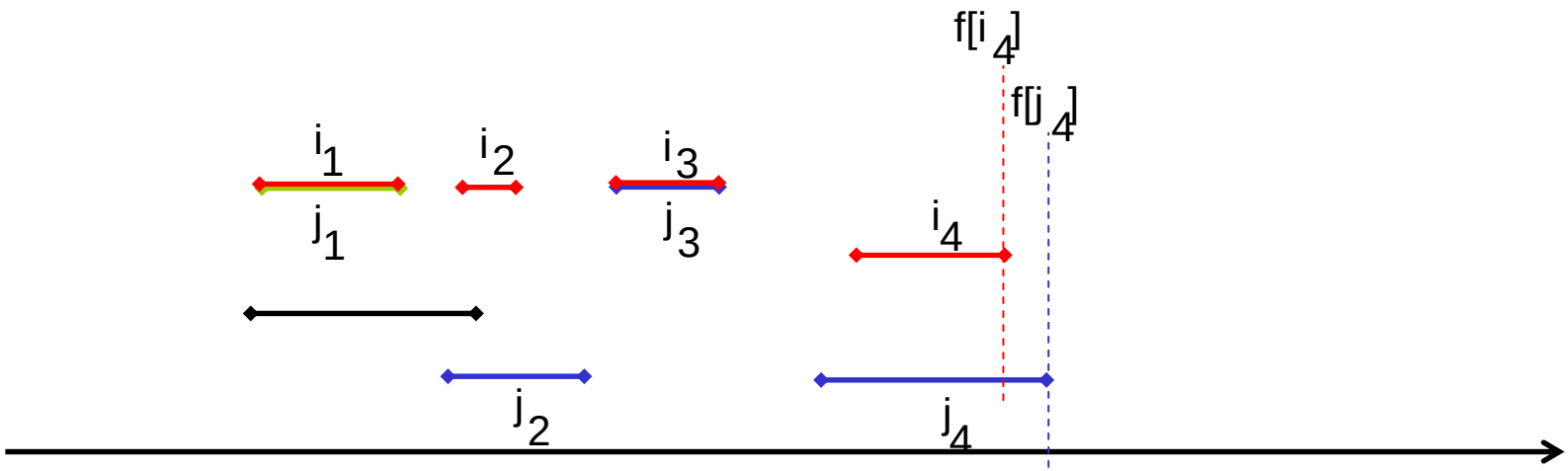
Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.



Intervall Scheduling

Lemma:

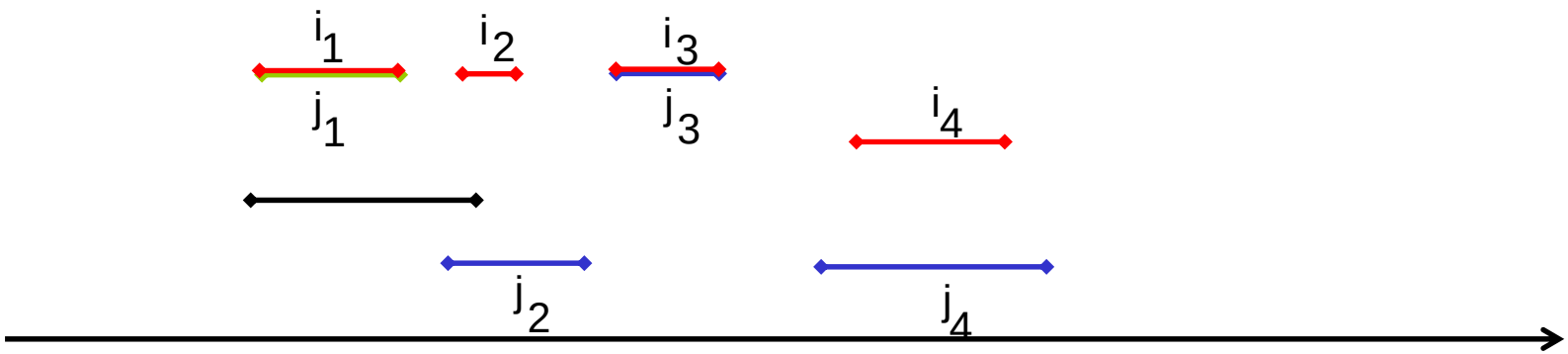
Für alle $r \leq k$ gilt $f[i_r] \leq f[j_r]$.



Intervall Scheduling

Satz:

Die von Algorithmus IntervalScheduling berechnete Lösung A ist optimal.



Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

$\Theta(1)$

Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

Θ(1)

Θ(n)

Intervall Scheduling

IntervalScheduling(s,f)

1. n ← length[s]
2. A ← {1}
3. j ← 1
4. **for** i ← 2 **to** n **do**
5. **if** s[i] ≥ f[j] **then**
6. A ← A ∪ {i}
7. j ← i
8. **return** A

Θ(1)

Θ(n)

Θ(1)

Intervall Scheduling

```
IntervalScheduling(s,f)
1.  n ← length[s]                          } Θ(1)
2.  A ← {1}
3.  j ← 1
4.  for i ← 2 to n do                      } Θ(n)
5.      if s[i] ≥ f[j] then
6.          A ←  A ∪ {i}
7.          j ←  i
8.  return A                               } Θ(1)
                                           ------
                                           Θ(n)
```

Intervall Scheduling

Satz:

Algorithmus IntervalScheduling berechnet in $\Theta(n)$ Zeit eine optimale Lösung, wenn die Eingabe nach Endzeit der Intervalle (rechter Endpunkt) sortiert ist. Die Sortierung kann in $\Theta(n \log n)$ Zeit berechnet werden.

Greedy-Verfahren

Greedy-Wahl-Eigenschaften:

Wenn man optimale Teillösung hat und man trifft eine lokal optimale Wahl, dann gibt es eine global optimale Lösung, die diese Wahl enthält.

Optimalität von Teillösungen:

Eine Teillösung einer optimalen Lösung ist eine optimale Lösung des Teilproblems.

→ nach jeder lokal optimalen Wahl erhalten wir ein zur Ausgangs-situation analoges Problem