Karlsruher Institut für Technologie

Algorithmen II
Vorlesung am 24.01.2013

Online Algorithmen

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

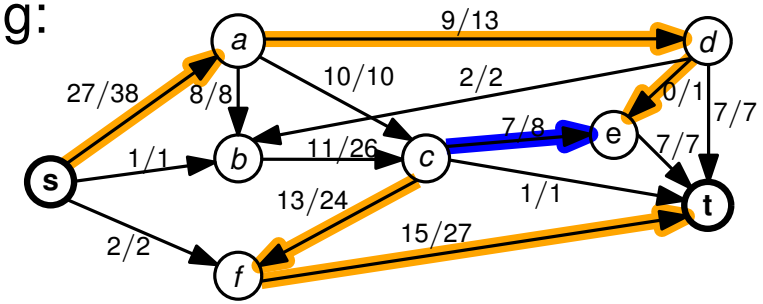
Einführung

Bisher in der Vorlesung betrachtet: **Offline-Algorithmen**

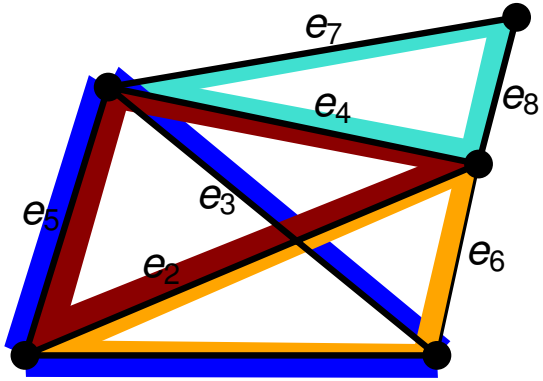
Charakteristische Merkmale:

- Eingabe muss vor Beginn der Ausführung bekannt sein.
- Berechnungen geschehen auf der gesamten Eingabe.

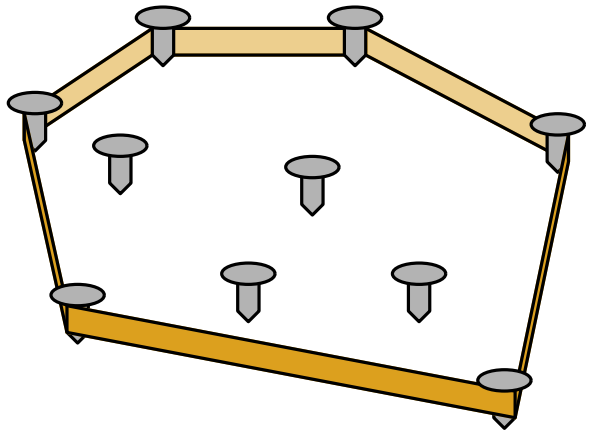
Beispiele aus der Vorlesung:



Flussalgorithmen



Algorithmen für Kreisbasen



Algorithmus für konvexe Hülle

Einführung

Online-Algorithmus: Ein Algorithmus heißt *Online-Algorithmus*, wenn er eine Eingabesequenz sequentiell stückweise abarbeitet, ohne die gesamte Eingabesequenz zu betrachten.

Charakteristische Merkmale:

- Liefern nach Abarbeitung einzelner Blöcke der Eingabesequenz bereits eine Lösung.
- Entscheidungen beruhen nur auf bereits vergangenen Ereignissen, aber nicht auf zukünftigen.

Beispiel: Job Scheudling eines Prozessors

Eingabesequenz:

atomare Jobs verschiedener Länge

Gesucht: Zuweisung für Prozessor, sodass durchschnittliche Wartezeit minimiert wird.

Kompetitive Analyse

Idee: Bestimme die Qualität eines Online-Algorithmus mithilfe eines optimalen Offline-Algorithmus.

Definition 35: Ein Online-Algorithmus ALG heißt *c-kompetitiv*, falls es eine Konstante α gibt, sodass für alle Instanzen I gilt

$$\mathrm{ALG}(I) \leq c \cdot \mathrm{OPT}(I) + \alpha.$$

Dabei bezeichnet ALG(I) den Wert der Lösung von ALG auf I angewendet und OPT(I) den Wert einer optimalen Lösung für I.
Falls $\alpha \leq 0$, dann heißt ALG *strikt c-kompetitiv*.

- ALG heißt *kompetitiv*, falls es Konstante c gibt, sodass ALG c-kompetitiv ist.
- c wird die *relative Güte* genannt.
- Bemerkung: c-kompetitiver Online-Algorithmus ist Approximationsalgorithmus mit relativer Güte c.

Beobachtung: Wenn ALG c-kompetitiv ist, dann ist er auch c'-kompetitiv für alle $c' \geq c$.

Ziel: Finde für ein gegebenes Problem einen Online-Algorithmus mit minimaler relativer Güte.

Beispiel: Ski-Verleih

Szenario: Man möchte mehrmals in die Alpen, um dort Ski zu fahren. Allerdings weiß man noch nicht wie häufig und man besitzt auch noch keine eigenen Ski.
Man steht nun vor der Entscheidung: Entweder man leiht sich im Skigebiet Ski für 50 € für den jeweiligen Ausflug, oder man kauft sich eigene Ski für 300 €.

Frage: Wie verhält man sich am besten?

Beispiel: Ski-Verleih

Szenario: Man möchte mehrmals in die Alpen, um dort Ski zu fahren. Allerdings weiß man noch nicht wie häufig und man besitzt auch noch keine eigenen Ski.
Man steht nun vor der Entscheidung: Entweder man leiht sich im Skigebiet Ski für 50 € für den jeweiligen Ausflug, oder man kauft sich eigene Ski für 300 €.

Frage: Wie verhält man sich am besten?

Einfacher Online-Algorithmus ALG**:** Leihe für die ersten fünf Ausflüge Ski und kaufe vor dem sechsten eigene.

Beispiel: Ski-Verleih

Szenario: Man möchte mehrmals in die Alpen, um dort Ski zu fahren. Allerdings weiß man noch nicht wie häufig und man besitzt auch noch keine eigenen Ski.
Man steht nun vor der Entscheidung: Entweder man leiht sich im Skigebiet Ski für 50 € für den jeweiligen Ausflug, oder man kauft sich eigene Ski für 300 €.

Frage: Wie verhält man sich am besten?

Einfacher Online-Algorithmus ALG**:** Leihe für die ersten fünf Ausflüge Ski und kaufe vor dem sechsten eigene.

Analyse: Sei k die Anzahl der Ausflüge, die man tatsächlich unternimmt.

Sei $k \leq 5$: Man zahlt $k \cdot 50$. Die Lösung eines optimalen Offline-Algorithmus ist ebenfalls $k \cdot 50$ für $k \leq 5$. Es ergibt sich eine relative Güte von $\frac{k \cdot 50}{k \cdot 50} = 1$.

Sei $k > 5$: Man zahlt $5 \cdot 50 + 300$. Die Lösung eines optimalen Offline-Algorithmus ist 300. Es ergibt sich eine relative Güte von $\frac{550}{300} = \frac{11}{6}$.

Algorithmus ist $\frac{11}{6}$-kompetitiv.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel: Ski-Verleih

Szenario: Man möchte mehrmals in die Alpen, um dort Ski zu fahren. Allerdings weiß man noch nicht wie häufig und man besitzt auch noch keine eigenen Ski.
Man steht nun vor der Entscheidung: Entweder man leiht sich im Skigebiet Ski für 50 € für den jeweiligen Ausflug, oder man kauft sich eigene Ski für 300 €.

Gibt es einen besseren Online-Algorithmus?

Annahme: Man kauft für den x-ten Ausflug eigene Ski und unternimmt tatsächlich k Ausflüge.

Beispiel: Ski-Verleih

Szenario: Man möchte mehrmals in die Alpen, um dort Ski zu fahren. Allerdings weiß man noch nicht wie häufig und man besitzt auch noch keine eigenen Ski.
Man steht nun vor der Entscheidung: Entweder man leiht sich im Skigebiet Ski für 50 € für den jeweiligen Ausflug, oder man kauft sich eigene Ski für 300 €.

Gibt es einen besseren Online-Algorithmus?

Annahme: Man kauft für den x-ten Ausflug eigene Ski und unternimmt tatsächlich k Ausflüge.

1. Fall: $x < 6$

Für $k < x$ bleibt die relative Güte 1.
Für $x = k$ liefert der optimale Algorithmus $50 \cdot x$ und der Online-Algorithmus $50 \cdot (x - 1) + 300$:

$$\frac{50 \cdot x + 250}{50 \cdot x} = 1 + \frac{5}{x} \geq 2$$

Beispiel: Ski-Verleih

Szenario: Man möchte mehrmals in die Alpen, um dort Ski zu fahren. Allerdings weiß man noch nicht wie häufig und man besitzt auch noch keine eigenen Ski.
Man steht nun vor der Entscheidung: Entweder man leiht sich im Skigebiet Ski für 50 € für den jeweiligen Ausflug, oder man kauft sich eigene Ski für 300 €.

Gibt es einen besseren Online-Algorithmus?

Annahme: Man kauft für den x-ten Ausflug eigene Ski und unternimmt tatsächlich k Ausflüge.

1. Fall: $x < 6$

Für $k < x$ bleibt die relative Güte 1.
Für $x = k$ liefert der optimale Algorithmus $50 \cdot x$ und der Online-Algorithmus $50 \cdot (x - 1) + 300$:

$$\frac{50 \cdot x + 250}{50 \cdot x} = 1 + \frac{5}{x} \geq 2$$

2. Fall: $x > 6$

Für $x = k$ liefert der optimale Algorithmus 300 während der Online-Algorithmus $50 \cdot (x - 1) + 300$:

$$\frac{50 \cdot x + 250}{300} = \frac{5 + x}{6}$$

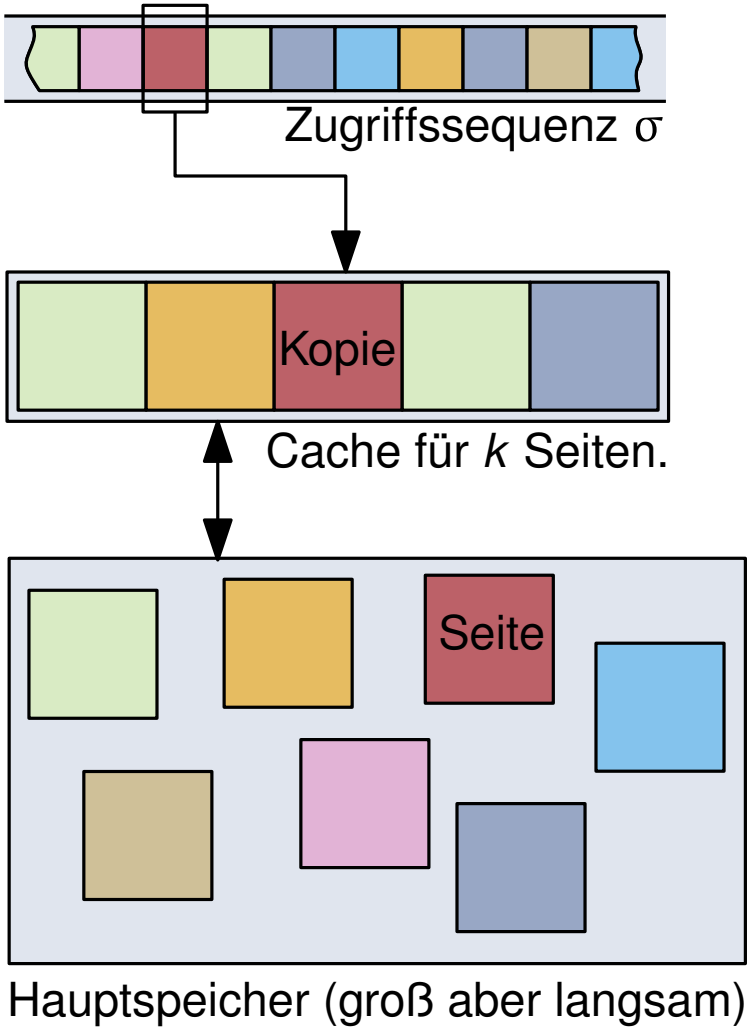
Beste Online-Algorithmus ist $\frac{11}{6}$-kompetitiv (mit $x = 6$).

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Online-Algorithmen für Paging

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Paging



- Hauptspeicher enthält N Seiten: $P = \{p_1, \ldots, p_N\}$
- Cache kann k Kopien von Seiten aus dem Hauptspeicher enthalten ($k < N$).
- n sequentielle Seitenzugriffe eines Programms werden beschrieben durch die Sequenz

$$\sigma\colon \{1, \ldots, n\} \to \{1, \ldots, N\}$$

Ablauf:

1. Programm fragt die i-te Seite $p_{\sigma(i)}$ an.
2. Falls $p_{\sigma(i)}$ noch nicht im Cache enthalten ist (**Fehlzugriff**), dann wird $p_{\sigma(i)}$ in den Cache geladen.
3. Programm greift auf $p_{\sigma(i)}$ im Cache zu.

Wie Seiten im Cache ersetzen, damit möglichst wenige Fehlzugriffe auftreten?

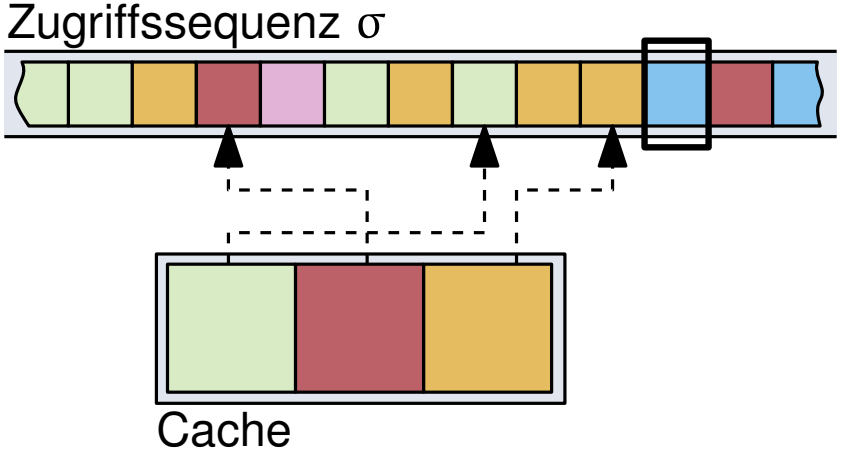
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Strategien

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)
LFD: Longest Forward Distance	deren Anfrage am weitesten in der Zukunft liegt.

Alle bis auf LFD sind Online-Algorithmen. LFD muss die komplette Anfragesequenz kennen und ist somit ein Offline-Algorithmus.

Beispiel LRU:



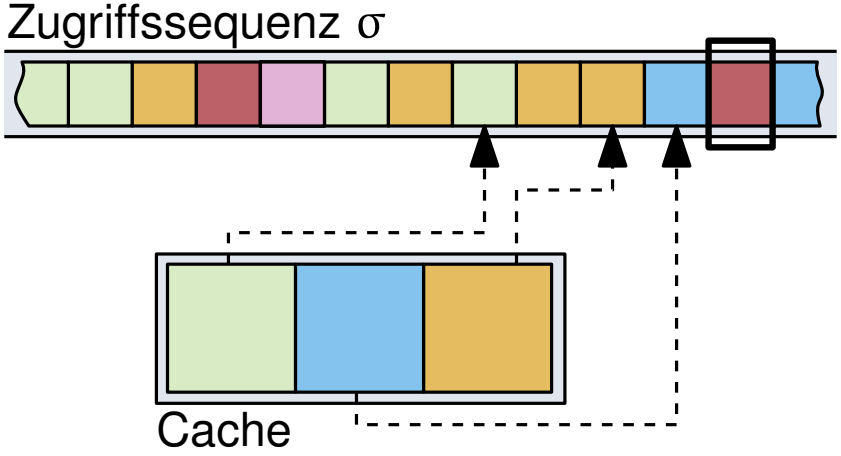
1. Speichere für jede Seite im Cache, wann sie zuletzt verwendet wurde.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Vergangenheit liegt.

Strategien

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)
LFD: Longest Forward Distance	deren Anfrage am weitesten in der Zukunft liegt.

Alle bis auf LFD sind Online-Algorithmen. LFD muss die komplette Anfragesequenz kennen und ist somit ein Offline-Algorithmus.

Beispiel LRU:



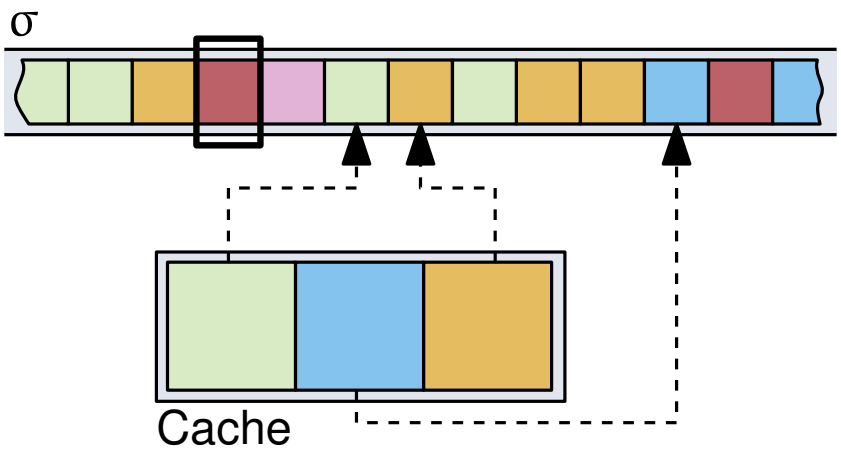
1. Speichere für jede Seite im Cache, wann sie zuletzt verwendet wurde.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Vergangenheit liegt.

Strategien

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)
LFD: Longest Forward Distance	deren Anfrage am weitesten in der Zukunft liegt.

Alle bis auf LFD sind Online-Algorithmen. LFD muss die komplette Anfragesequenz kennen und ist somit ein Offline-Algorithmus.

Beispiel LFD:



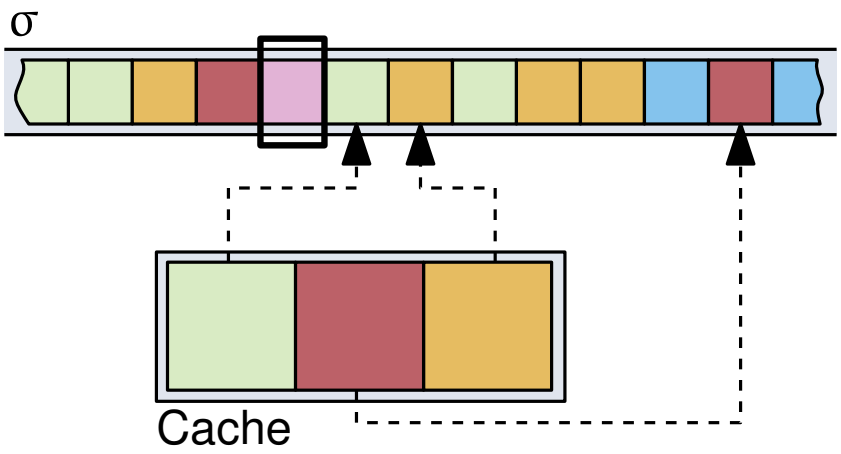
1. Speichere für jede Seite im Cache, wann sie als nächstes verwendet wird.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Zukunft liegt.

Strategien

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)
LFD: Longest Forward Distance	deren Anfrage am weitesten in der Zukunft liegt.

Alle bis auf LFD sind Online-Algorithmen. LFD muss die komplette Anfragesequenz kennen und ist somit ein Offline-Algorithmus.

Beispiel LFD:



1. Speichere für jede Seite im Cache, wann sie als nächstes verwendet wird.
2. Ersetze die Seite im Cache, deren Verwendung am weitesten in der Zukunft liegt.

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Beweisskizze:

Zeige, dass jeder optimale Algorithmus OPT so modifiziert werden kann, dass er auf einer beliebigen Sequenz σ wie LFD verfährt, ohne dass die Anzahl Fehlzugriffe zunimmt.

Induktion mithilfe folgender Behauptung:
Gegeben ein beliebiger Paging-Algorithmus ALG und eine Anfragesequenz σ: Es gibt Offline-Algorithmus ALG_i für $i = 1, \ldots, |\sigma|$, sodass folgende Eigenschaften gelten:

1. ALG_i bearbeitet die ersten $i - 1$ Anfragen auf dieselbe Weise wie ALG.
2. Falls die i-te Anfrage einen Fehlzugriff hat, dann wendet ALG_i Prinzip von LFD auf den Cache an.
3. Die Anzahl Fehlzugriffe steigt nicht an: $ALG_i(\sigma) \leq ALG(\sigma)$.

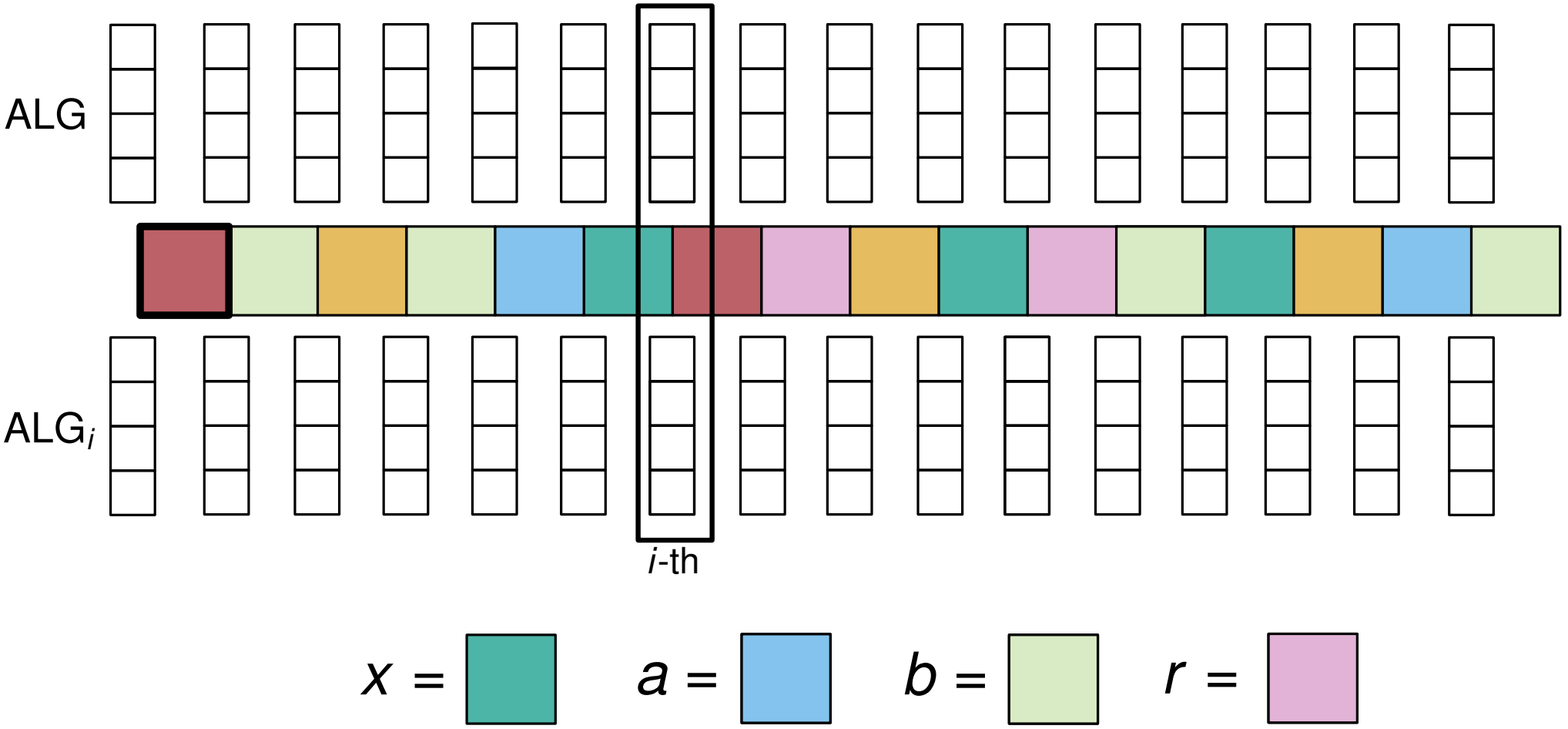
Wende Behauptung auf beliebigen optimalen Offline-Algorithmus OPT an:

1. Wende für $i = 1$ Behauptung auf OPT an, mit Ergebnis OPT_1.
2. Wende für $i = 2$ Behauptung auf OPT_1 an, mit Ergebnis OPT_2, usw.
3. OPT_n arbeitet genau wie LFD und Fehlzugriffe haben nicht zugenommen: LFD ist optimal.

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

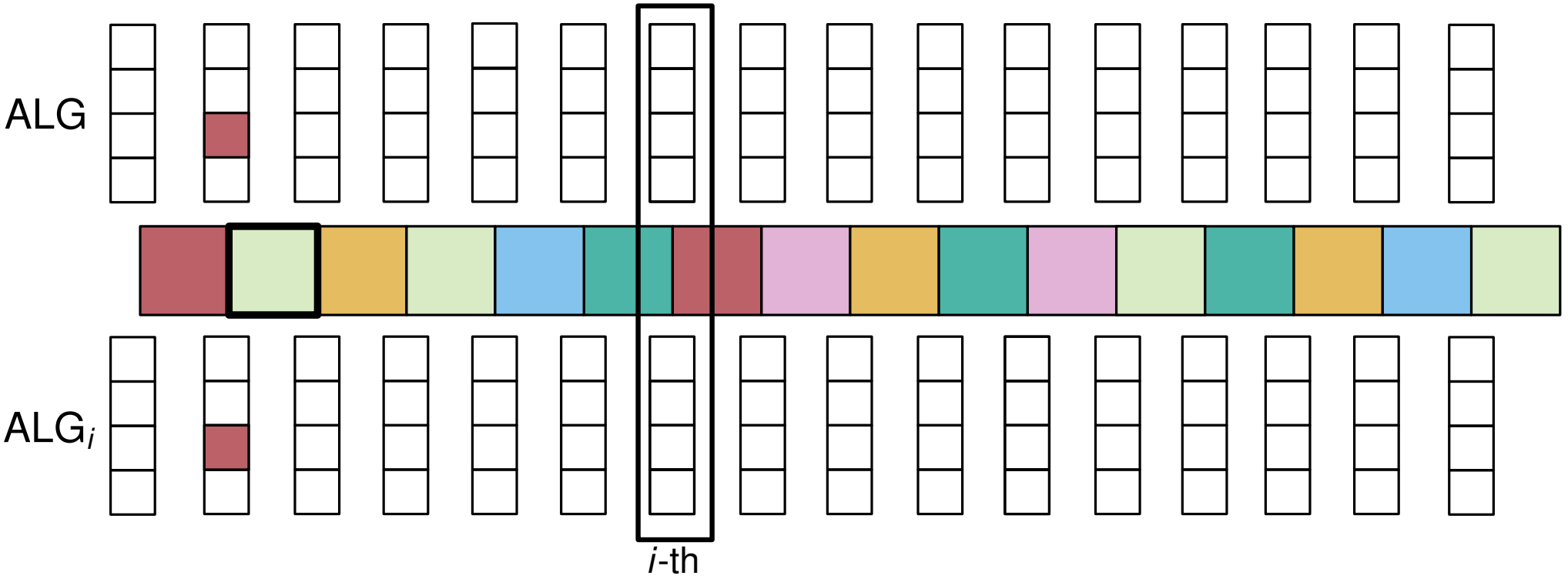


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

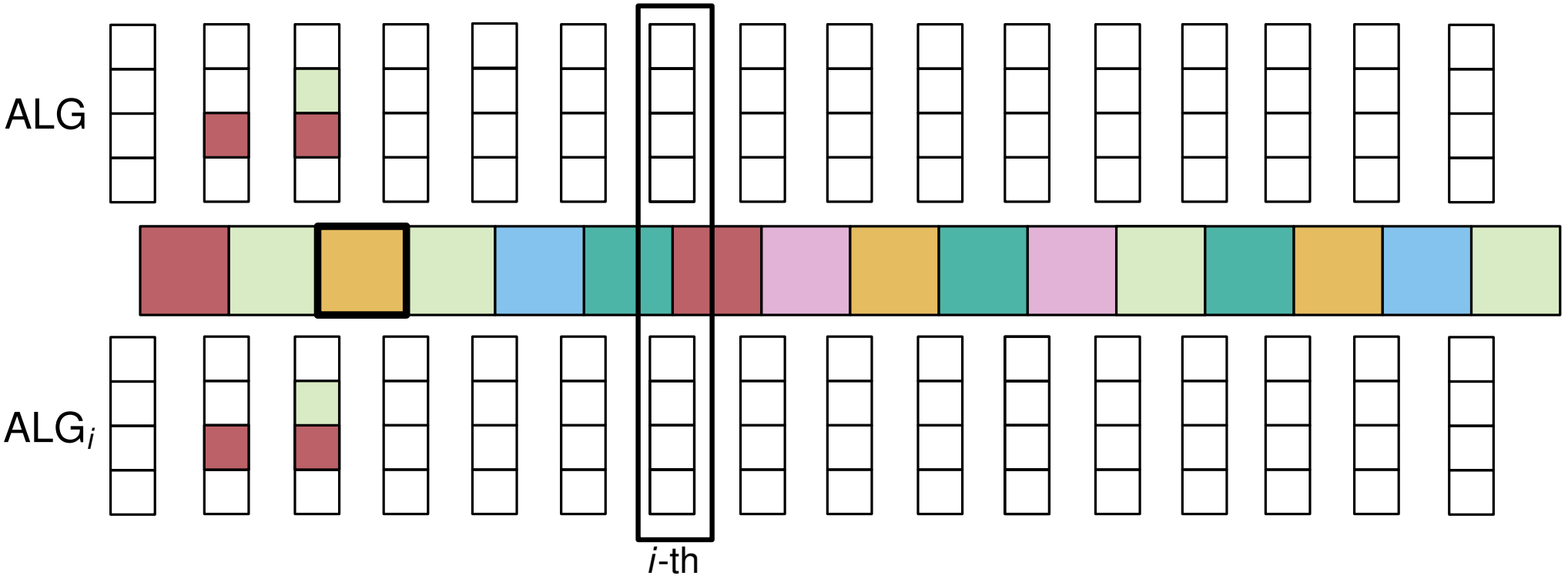


$x =$ $a =$ $b =$ $r =$

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

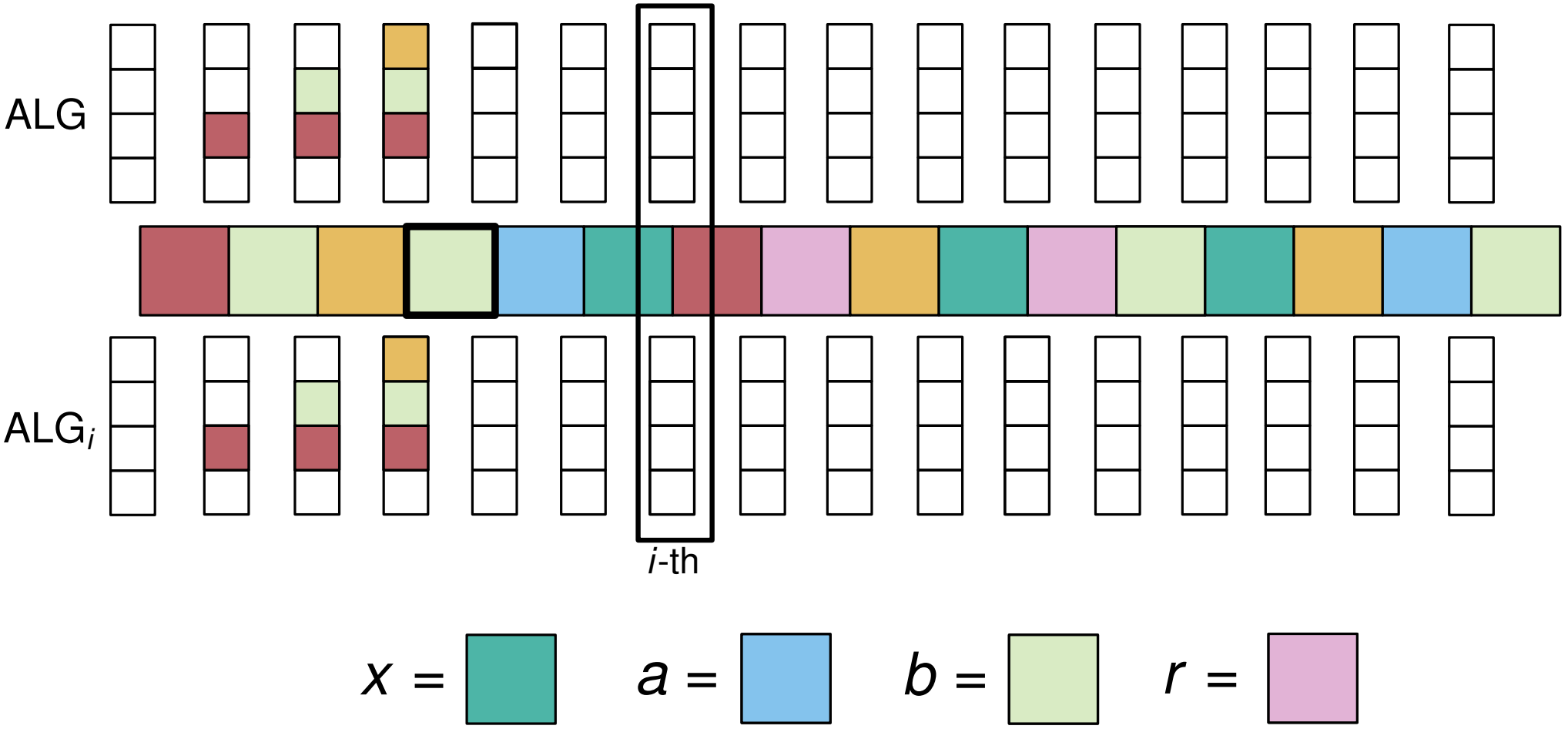


$x =$ $a =$ $b =$ $r =$

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

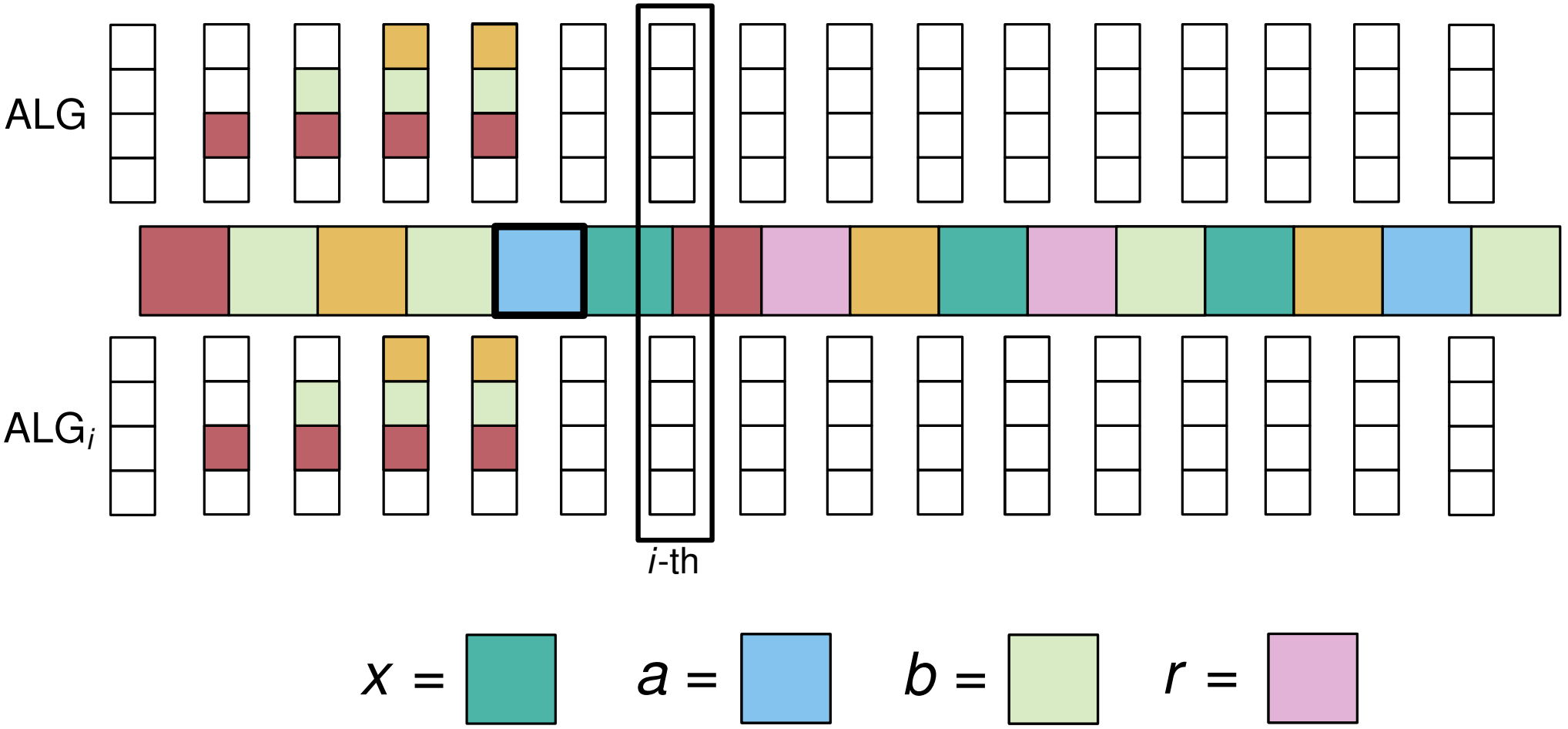


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

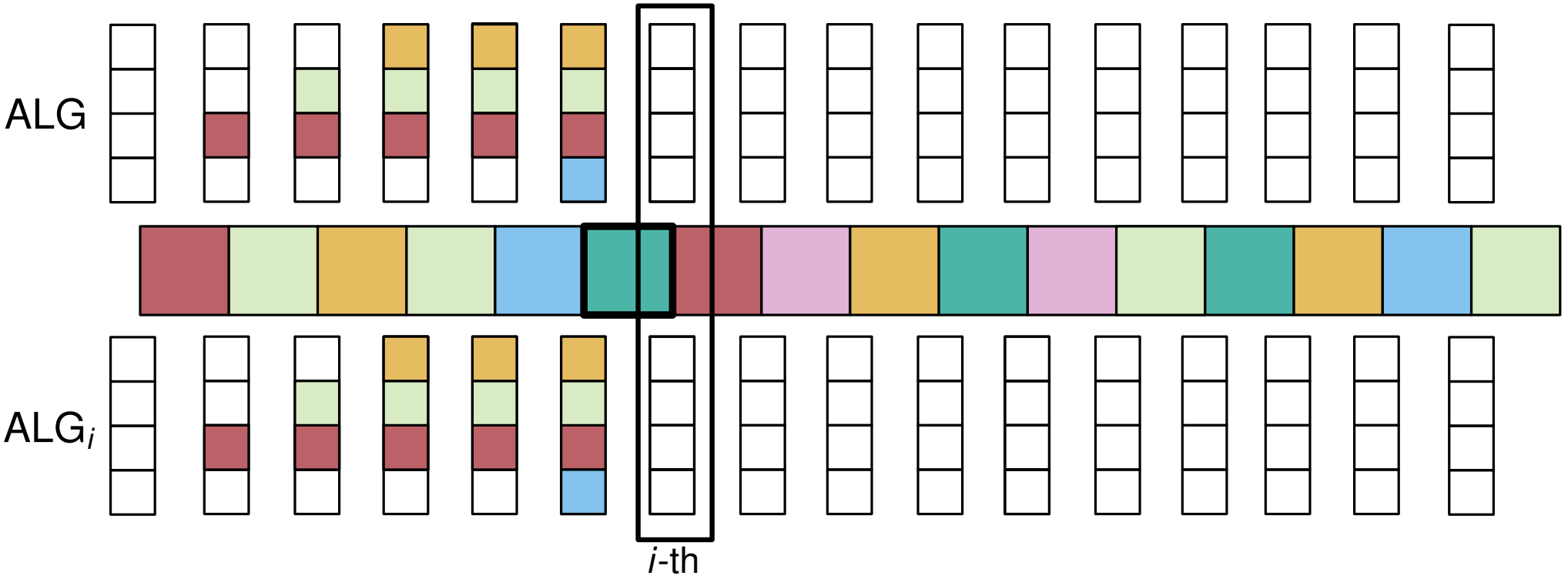


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

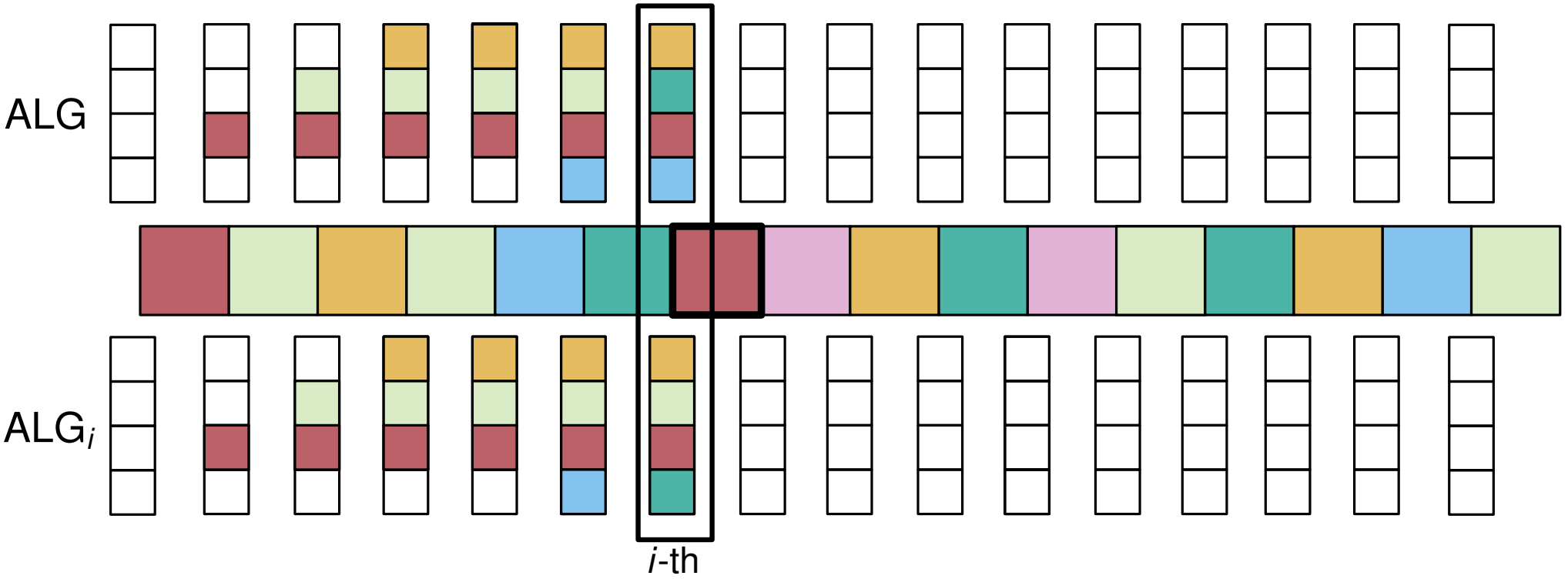


$x =$ $a =$ $b =$ $r =$

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

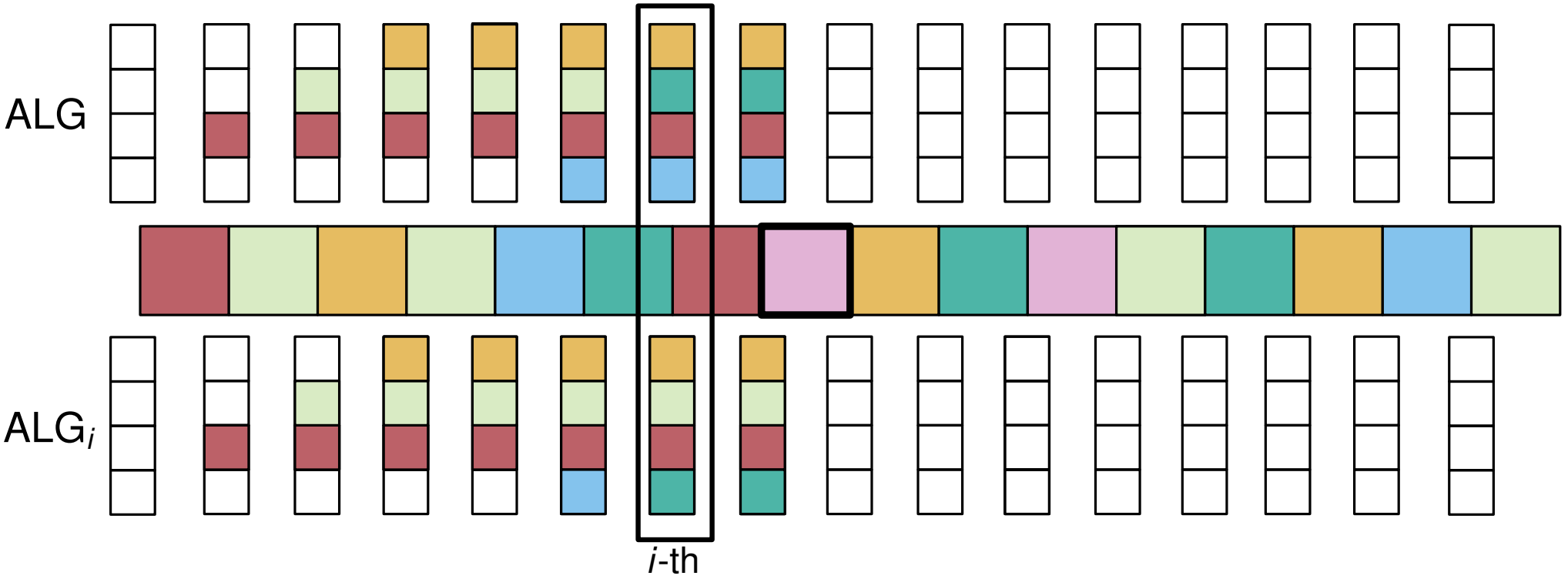


$x =$ $a =$ $b =$ $r =$

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

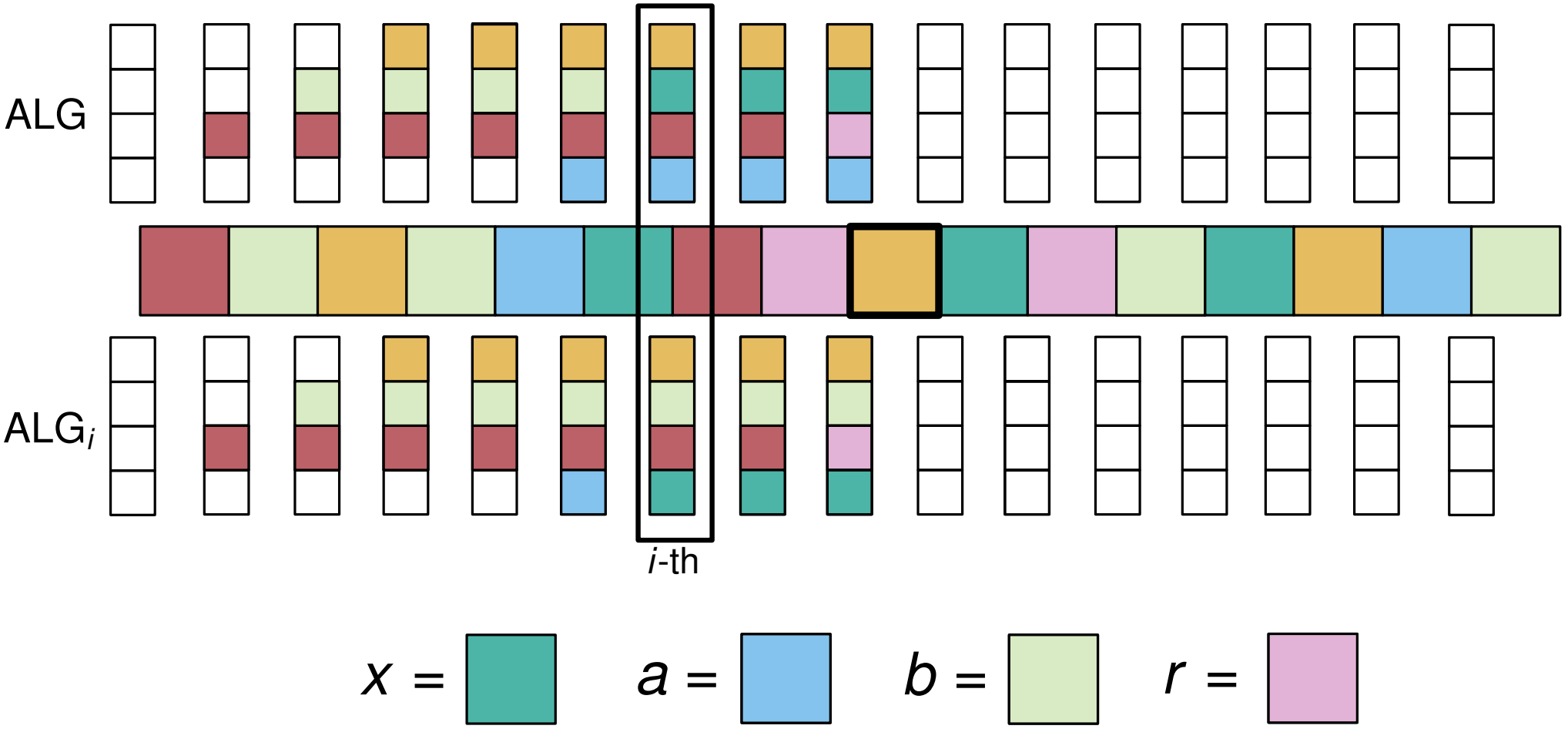


$x =$ $a =$ $b =$ $r =$

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

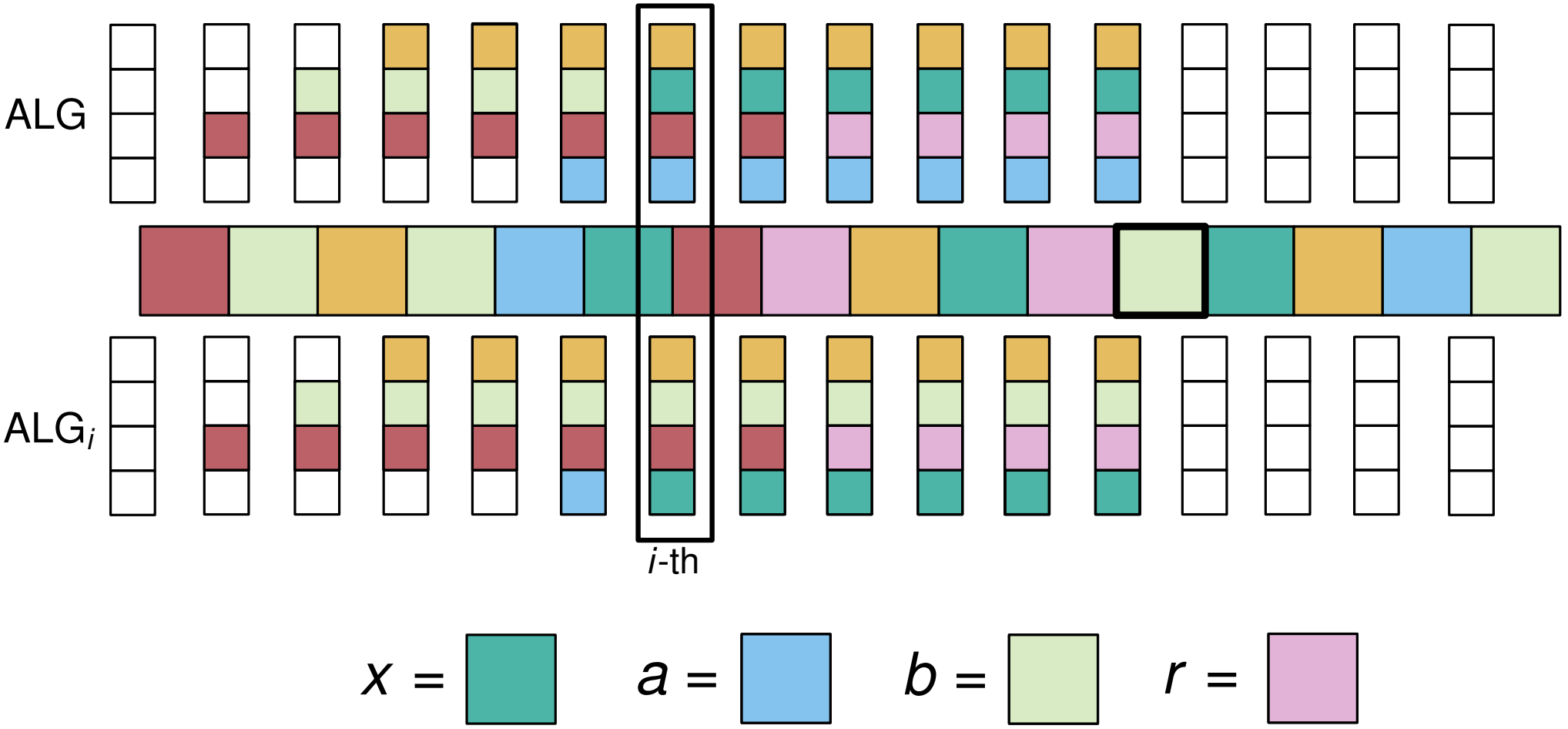


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

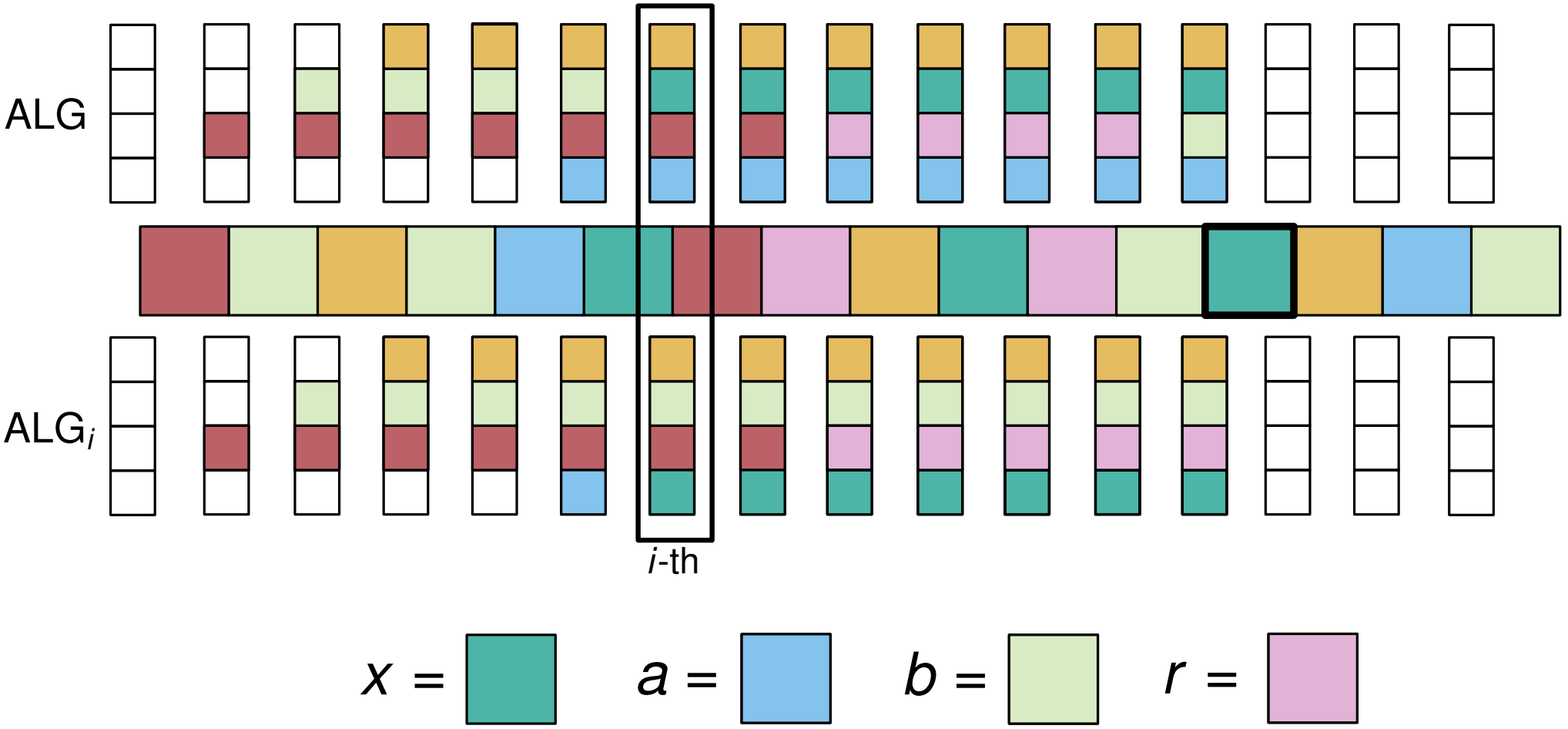


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

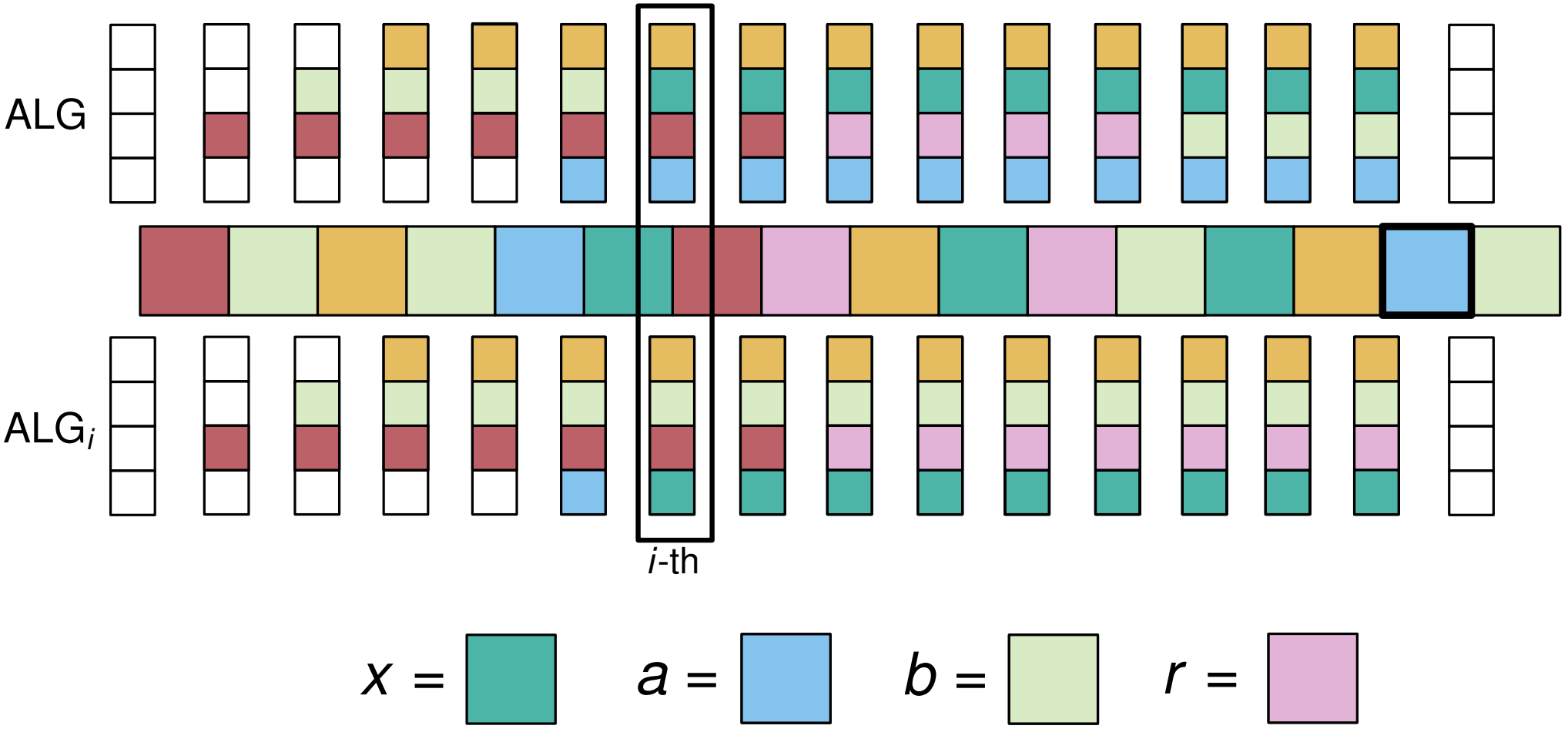


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:

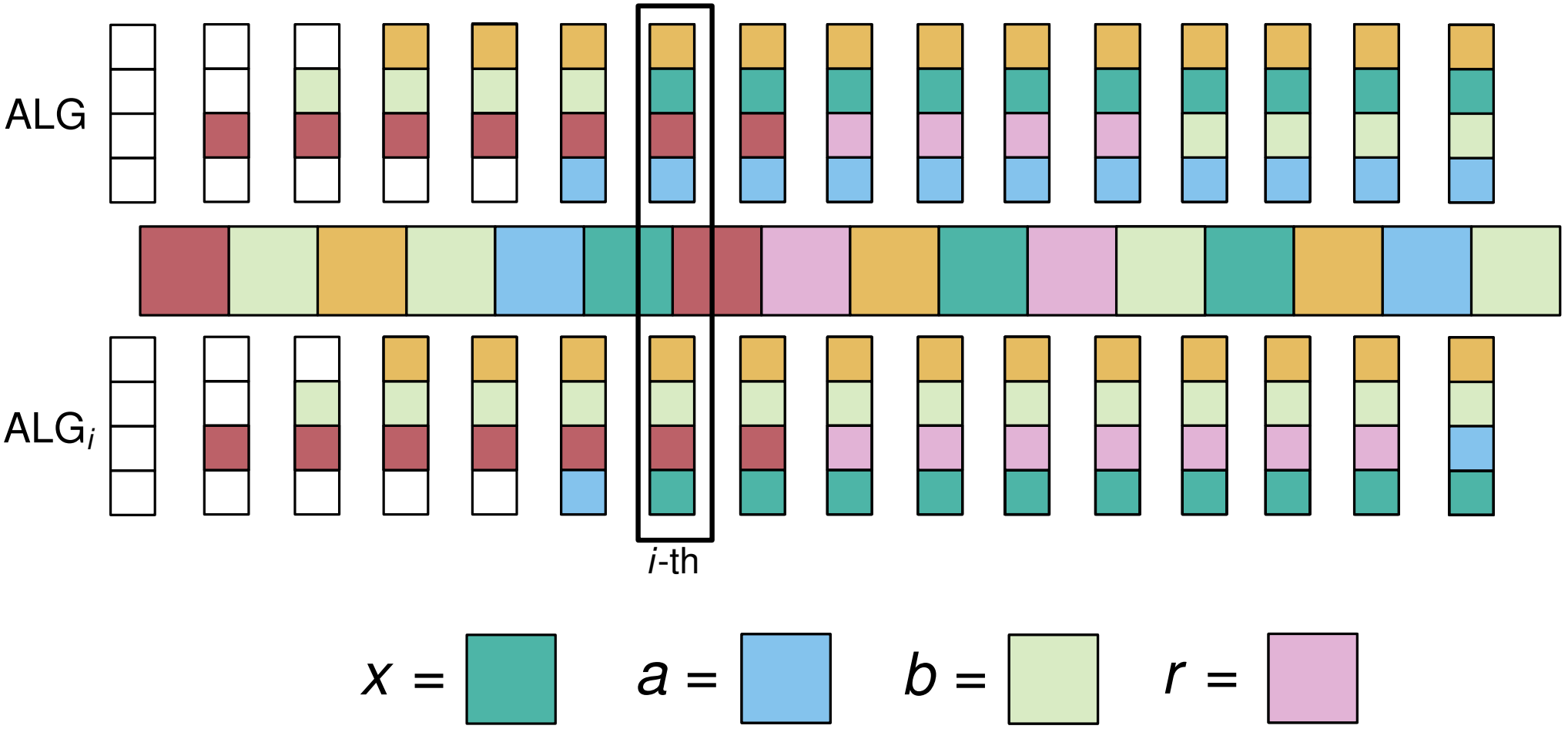


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Longest Forward Distance (LFD)

Theorem 50: LFD ist ein optimaler Offline-Algorithmus für Paging.

Illustration des Beweis:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

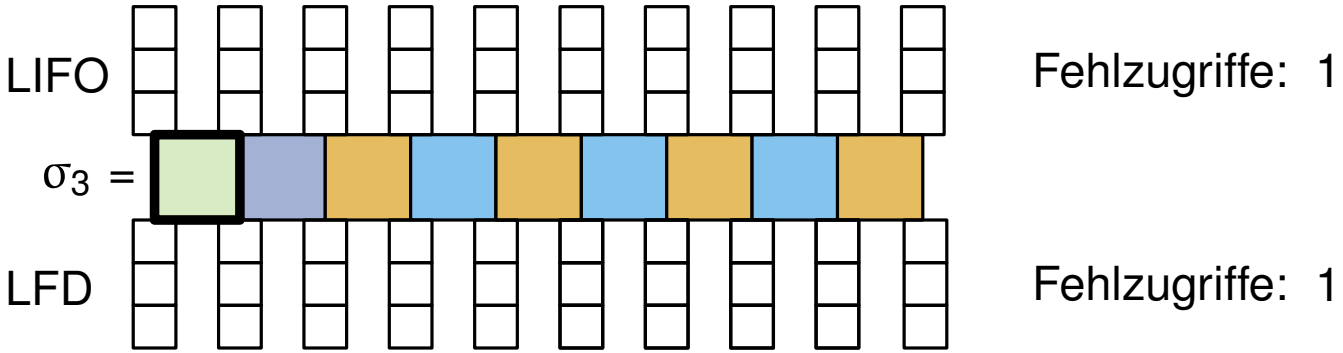
Kompetitive Paging-Algorithmen

LIFO (Last In/First out) ist nicht kompetitiv, d.h. die relative Güte c kann beliebig groß werden:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1, \ldots, p_k, (p_{k+1}, p_k)^m$$

- LIFO wird zuerst die Seiten $p_1, \ldots, p_k$ in den Cache aufnehmen (k Fehlzugriffe).
- Für die $k+1$-te Anfrage wird LIFO die Seite p_k durch die Seite p_{k+1} ersetzen.
- Für die $k+2$-te Anfrage wird LIFO die Seite p_{k+1} durch die Seite p_k ersetzen und usw.
- LIFO hat deshalb k+2m Fehlzugriffe.
- LFD hat nur k+1 Fehlzugriffe.
- Folglich: Für jedes c gibt es ein m, sodass $\mathrm{ALG}(\sigma_m) > c \cdot \mathrm{OPT}(\sigma_m)$



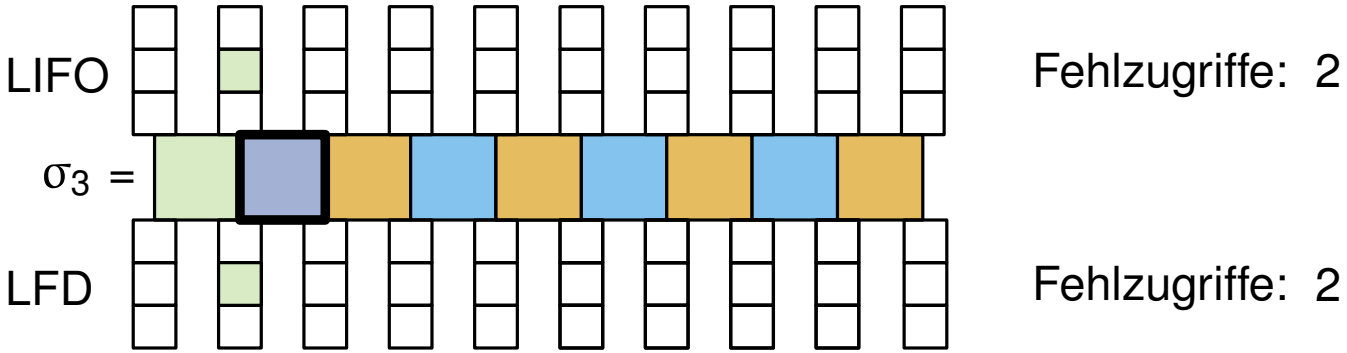
Kompetitive Paging-Algorithmen

LIFO (Last In/First out) ist nicht kompetitiv, d.h. die relative Güte c kann beliebig groß werden:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1, \ldots, p_k, (p_{k+1}, p_k)^m$$

- LIFO wird zuerst die Seiten $p_1, \ldots, p_k$ in den Cache aufnehmen (k Fehlzugriffe).
- Für die $k+1$-te Anfrage wird LIFO die Seite p_k durch die Seite p_{k+1} ersetzen.
- Für die $k+2$-te Anfrage wird LIFO die Seite p_{k+1} durch die Seite p_k ersetzen und usw.
- LIFO hat deshalb k+2m Fehlzugriffe.
- LFD hat nur k+1 Fehlzugriffe.
- Folglich: Für jedes c gibt es ein m, sodass $\mathrm{ALG}(\sigma_m) > c \cdot \mathrm{OPT}(\sigma_m)$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

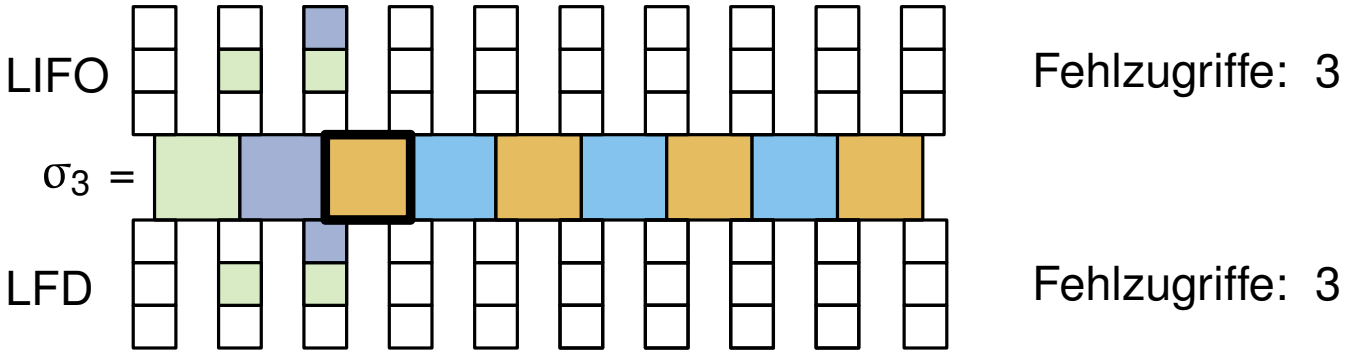
Kompetitive Paging-Algorithmen

LIFO (Last In/First out) ist nicht kompetitiv, d.h. die relative Güte c kann beliebig groß werden:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1, \ldots, p_k, (p_{k+1}, p_k)^m$$

- LIFO wird zuerst die Seiten $p_1, \ldots, p_k$ in den Cache aufnehmen (k Fehlzugriffe).
- Für die $k+1$-te Anfrage wird LIFO die Seite p_k durch die Seite p_{k+1} ersetzen.
- Für die $k+2$-te Anfrage wird LIFO die Seite p_{k+1} durch die Seite p_k ersetzen und usw.
- LIFO hat deshalb k+2m Fehlzugriffe.
- LFD hat nur k+1 Fehlzugriffe.
- Folglich: Für jedes c gibt es ein m, sodass $\mathrm{ALG}(\sigma_m) > c \cdot \mathrm{OPT}(\sigma_m)$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

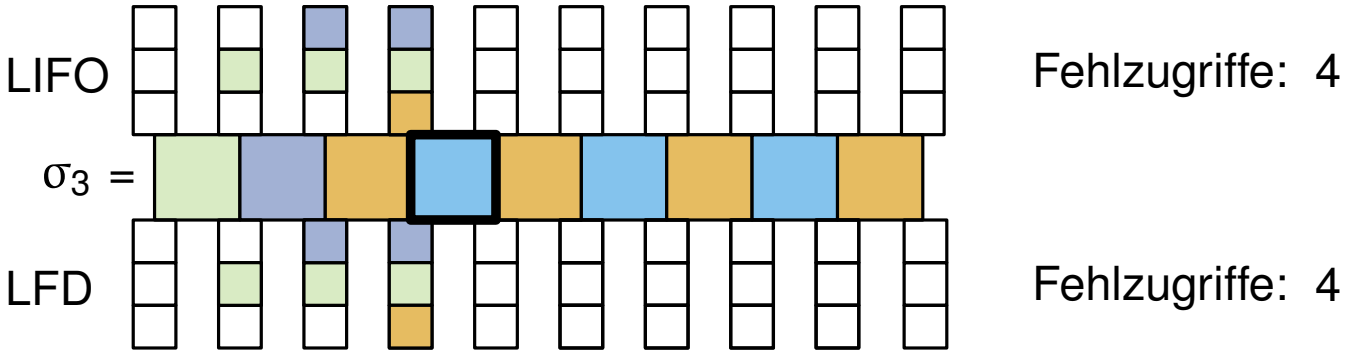
Kompetitive Paging-Algorithmen

LIFO (Last In/First out) ist nicht kompetitiv, d.h. die relative Güte c kann beliebig groß werden:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1, \ldots, p_k, (p_{k+1}, p_k)^m$$

- LIFO wird zuerst die Seiten $p_1, \ldots, p_k$ in den Cache aufnehmen (k Fehlzugriffe).
- Für die $k+1$-te Anfrage wird LIFO die Seite p_k durch die Seite p_{k+1} ersetzen.
- Für die $k+2$-te Anfrage wird LIFO die Seite p_{k+1} durch die Seite p_k ersetzen und usw.
- LIFO hat deshalb k+2m Fehlzugriffe.
- LFD hat nur k+1 Fehlzugriffe.
- Folglich: Für jedes c gibt es ein m, sodass $\mathrm{ALG}(\sigma_m) > c \cdot \mathrm{OPT}(\sigma_m)$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

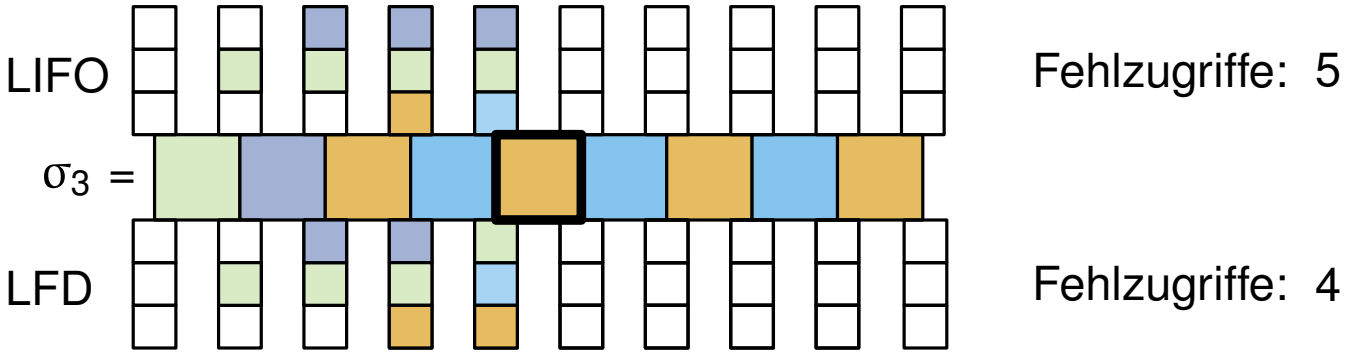
Kompetitive Paging-Algorithmen

LIFO (Last In/First out) ist nicht kompetitiv, d.h. die relative Güte c kann beliebig groß werden:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1, \ldots, p_k, (p_{k+1}, p_k)^m$$

- LIFO wird zuerst die Seiten $p_1, \ldots, p_k$ in den Cache aufnehmen (k Fehlzugriffe).
- Für die $k+1$-te Anfrage wird LIFO die Seite p_k durch die Seite p_{k+1} ersetzen.
- Für die $k+2$-te Anfrage wird LIFO die Seite p_{k+1} durch die Seite p_k ersetzen und usw.
- LIFO hat deshalb k+2m Fehlzugriffe.
- LFD hat nur k+1 Fehlzugriffe.
- Folglich: Für jedes c gibt es ein m, sodass $\mathrm{ALG}(\sigma_m) > c \cdot \mathrm{OPT}(\sigma_m)$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

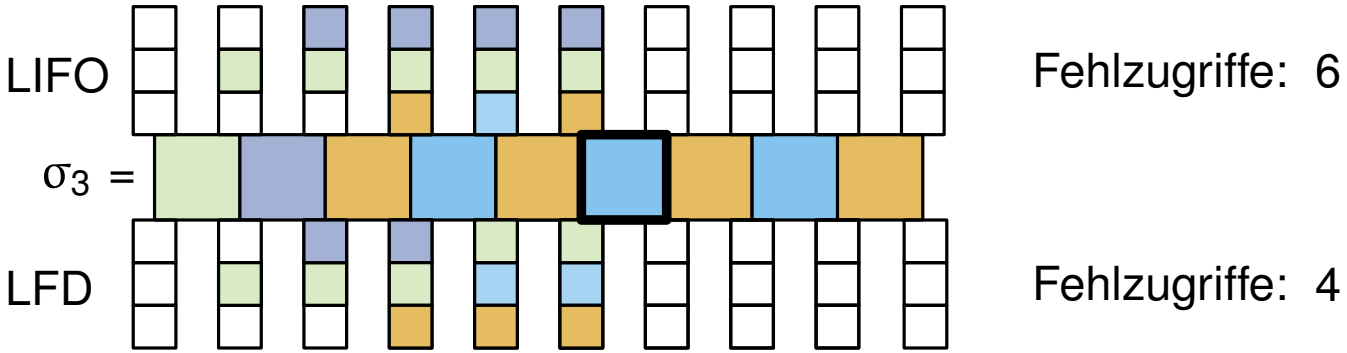
Kompetitive Paging-Algorithmen

LIFO (Last In/First out) ist nicht kompetitiv, d.h. die relative Güte c kann beliebig groß werden:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1, \ldots, p_k, (p_{k+1}, p_k)^m$$

- LIFO wird zuerst die Seiten $p_1, \ldots, p_k$ in den Cache aufnehmen (k Fehlzugriffe).
- Für die $k+1$-te Anfrage wird LIFO die Seite p_k durch die Seite p_{k+1} ersetzen.
- Für die $k+2$-te Anfrage wird LIFO die Seite p_{k+1} durch die Seite p_k ersetzen und usw.
- LIFO hat deshalb k+2m Fehlzugriffe.
- LFD hat nur k+1 Fehlzugriffe.
- Folglich: Für jedes c gibt es ein m, sodass $\mathrm{ALG}(\sigma_m) > c \cdot \mathrm{OPT}(\sigma_m)$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

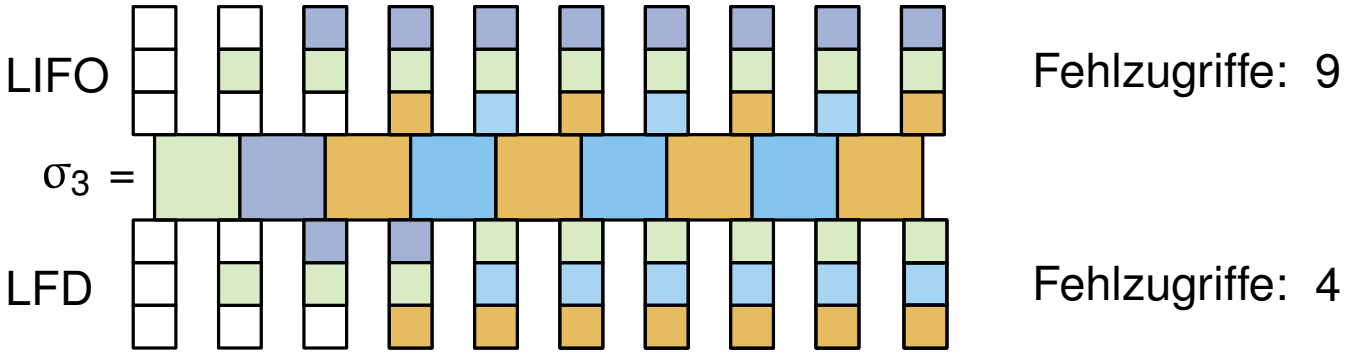
Kompetitive Paging-Algorithmen

LIFO (Last In/First out) ist nicht kompetitiv, d.h. die relative Güte c kann beliebig groß werden:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1, \ldots, p_k, (p_{k+1}, p_k)^m$$

- LIFO wird zuerst die Seiten $p_1, \ldots, p_k$ in den Cache aufnehmen (k Fehlzugriffe).
- Für die $k+1$-te Anfrage wird LIFO die Seite p_k durch die Seite p_{k+1} ersetzen.
- Für die $k+2$-te Anfrage wird LIFO die Seite p_{k+1} durch die Seite p_k ersetzen und usw.
- LIFO hat deshalb k+2m Fehlzugriffe.
- LFD hat nur k+1 Fehlzugriffe.
- Folglich: Für jedes c gibt es ein m, sodass $\mathrm{ALG}(\sigma_m) > c \cdot \mathrm{OPT}(\sigma_m)$



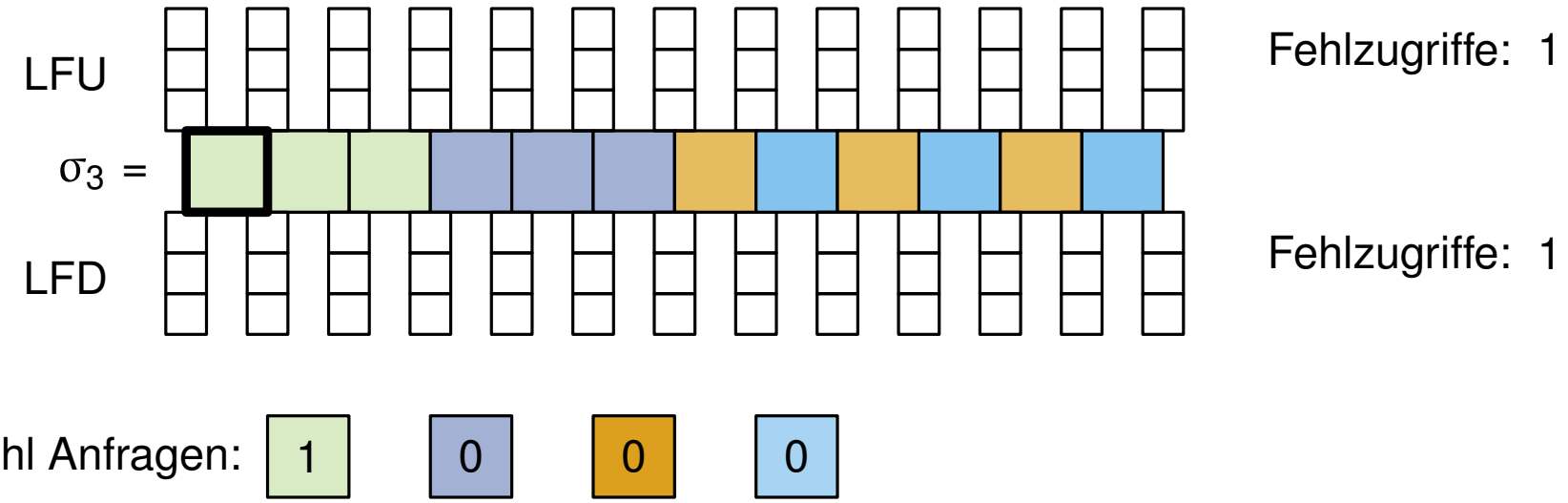
Kompetitive Paging-Algorithmen

LFU (Least Frequently Used) ist nicht kompetitiv, d.h. relative Güte c ist nicht beschränkt:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1^m, \ldots, p_{k-1}^m, (p_k, p_{k+1})^m$$

- LFU wird zuerst die Seiten $p_1, \ldots, p_{k-1}$ in den Cache aufnehmen (k-1 Fehlzugriffe).
- Nach $m \cdot (k-1)$ Anfragen, wird LFU für jedes Anfragepaar (p_k, p_{k+1}) einen Fehlzugriff haben. Folglich m weitere Fehlzugriffe.
- Da LFD nur k+1 Fehlzugriffe hat, ist die relative Güte von LFU nicht beschränkt.



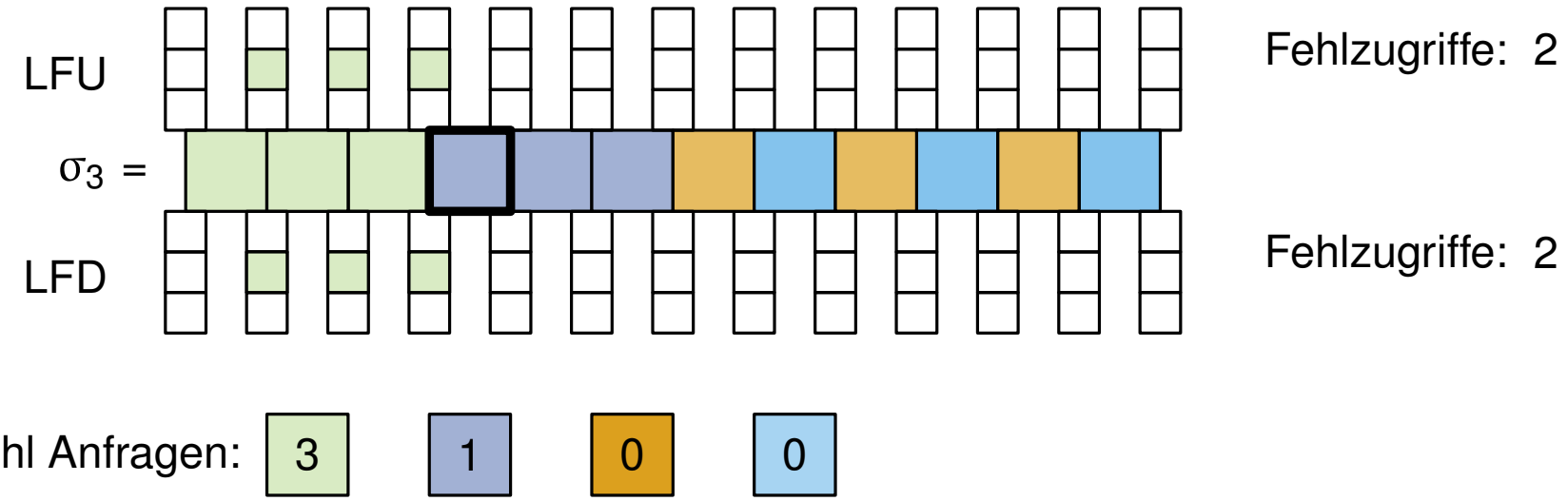
Kompetitive Paging-Algorithmen

LFU (Least Frequently Used) ist nicht kompetitiv, d.h. relative Güte c ist nicht beschränkt:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1^m, \ldots, p_{k-1}^m, (p_k, p_{k+1})^m$$

- LFU wird zuerst die Seiten $p_1, \ldots, p_{k-1}$ in den Cache aufnehmen (k-1 Fehlzugriffe).
- Nach $m \cdot (k-1)$ Anfragen, wird LFU für jedes Anfragepaar (p_k, p_{k+1}) einen Fehlzugriff haben. Folglich m weitere Fehlzugriffe.
- Da LFD nur k+1 Fehlzugriffe hat, ist die relative Güte von LFU nicht beschränkt.



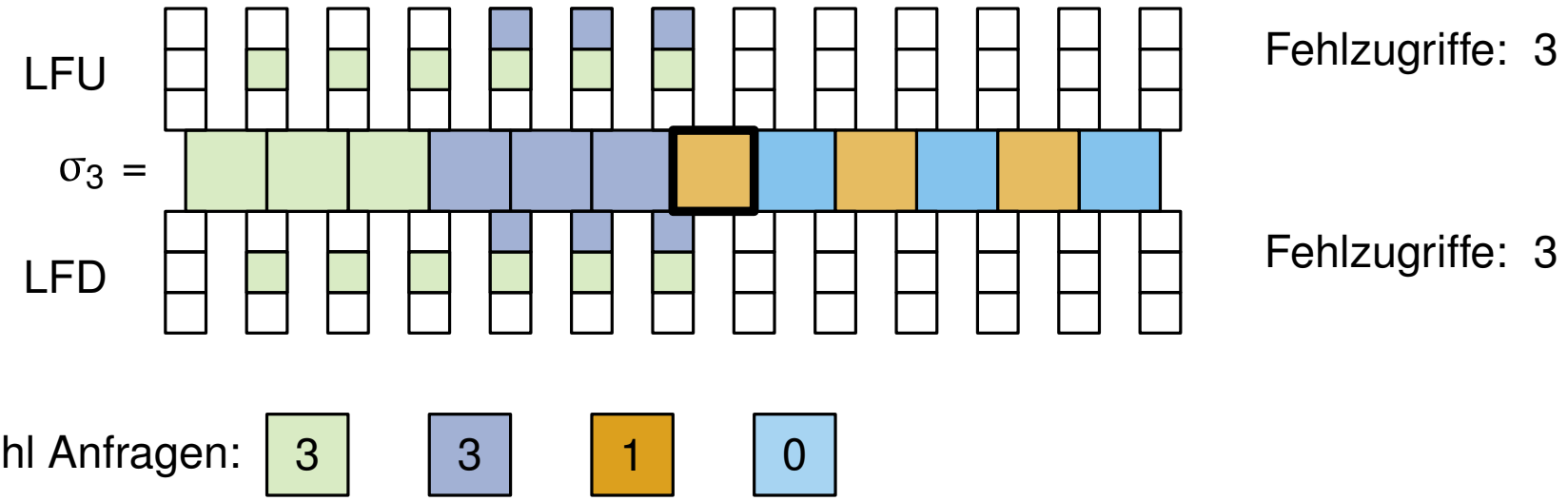
Kompetitive Paging-Algorithmen

LFU (Least Frequently Used) ist nicht kompetitiv, d.h. relative Güte c ist nicht beschränkt:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1^m, \ldots, p_{k-1}^m, (p_k, p_{k+1})^m$$

- LFU wird zuerst die Seiten $p_1, \ldots, p_{k-1}$ in den Cache aufnehmen (k-1 Fehlzugriffe).
- Nach $m \cdot (k-1)$ Anfragen, wird LFU für jedes Anfragepaar (p_k, p_{k+1}) einen Fehlzugriff haben. Folglich m weitere Fehlzugriffe.
- Da LFD nur k+1 Fehlzugriffe hat, ist die relative Güte von LFU nicht beschränkt.



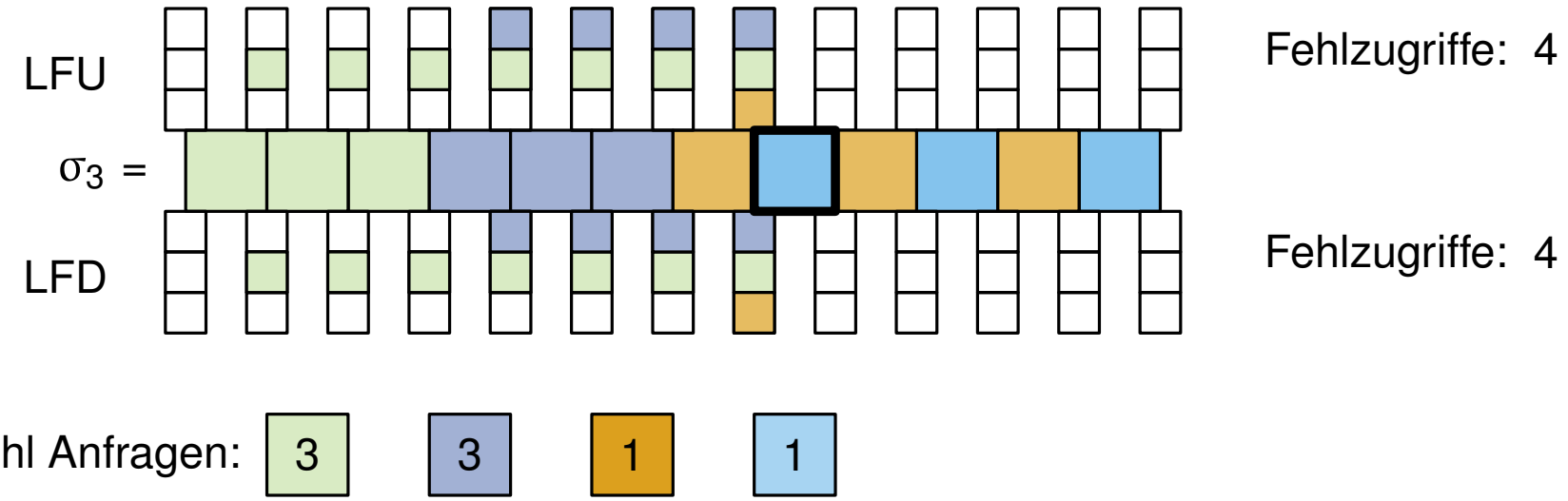
Kompetitive Paging-Algorithmen

LFU (Least Frequently Used) ist nicht kompetitiv, d.h. relative Güte c ist nicht beschränkt:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1^m, \ldots, p_{k-1}^m, (p_k, p_{k+1})^m$$

- LFU wird zuerst die Seiten $p_1, \ldots, p_{k-1}$ in den Cache aufnehmen (k-1 Fehlzugriffe).
- Nach $m \cdot (k-1)$ Anfragen, wird LFU für jedes Anfragepaar (p_k, p_{k+1}) einen Fehlzugriff haben. Folglich m weitere Fehlzugriffe.
- Da LFD nur k+1 Fehlzugriffe hat, ist die relative Güte von LFU nicht beschränkt.



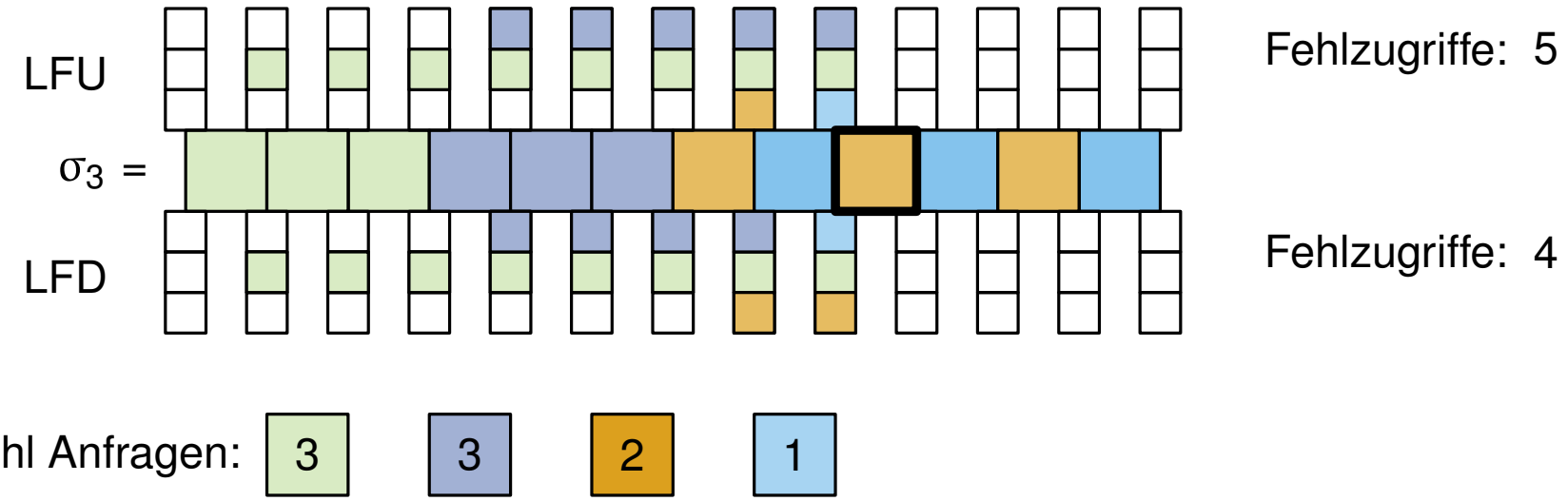
Kompetitive Paging-Algorithmen

LFU (Least Frequently Used) ist nicht kompetitiv, d.h. relative Güte c ist nicht beschränkt:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1^m, \ldots, p_{k-1}^m, (p_k, p_{k+1})^m$$

- LFU wird zuerst die Seiten $p_1, \ldots, p_{k-1}$ in den Cache aufnehmen (k-1 Fehlzugriffe).
- Nach $m \cdot (k-1)$ Anfragen, wird LFU für jedes Anfragepaar (p_k, p_{k+1}) einen Fehlzugriff haben. Folglich m weitere Fehlzugriffe.
- Da LFD nur k+1 Fehlzugriffe hat, ist die relative Güte von LFU nicht beschränkt.



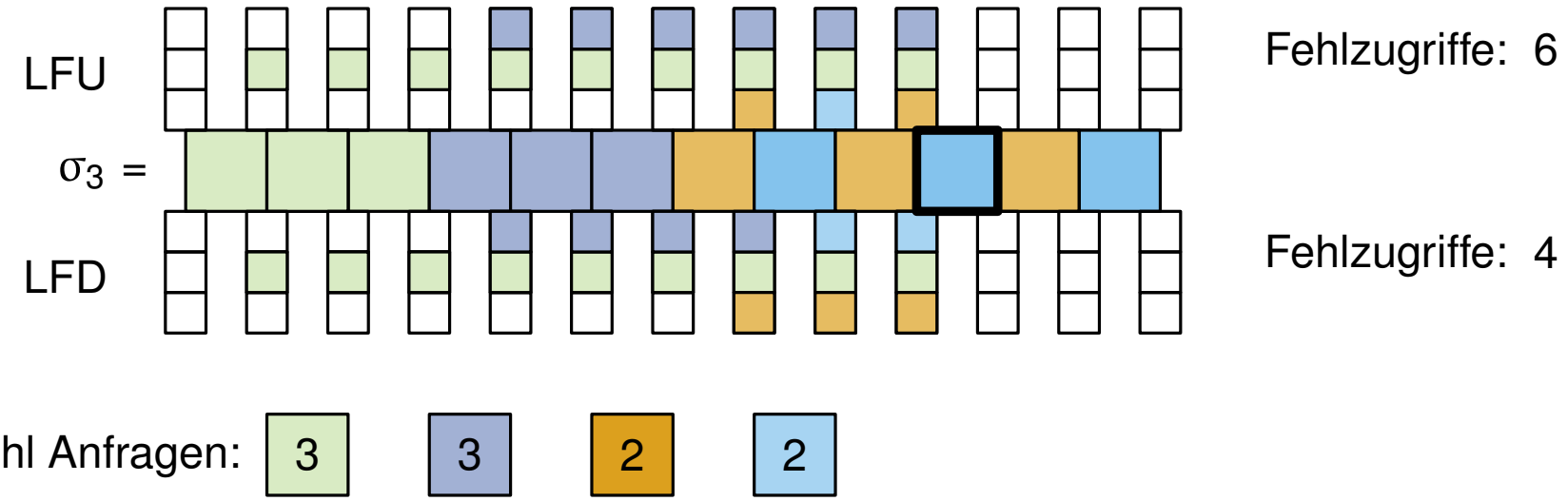
Kompetitive Paging-Algorithmen

LFU (Least Frequently Used) ist nicht kompetitiv, d.h. relative Güte c ist nicht beschränkt:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1^m, \ldots, p_{k-1}^m, (p_k, p_{k+1})^m$$

- LFU wird zuerst die Seiten $p_1, \ldots, p_{k-1}$ in den Cache aufnehmen (k-1 Fehlzugriffe).
- Nach $m \cdot (k-1)$ Anfragen, wird LFU für jedes Anfragepaar (p_k, p_{k+1}) einen Fehlzugriff haben. Folglich m weitere Fehlzugriffe.
- Da LFD nur k+1 Fehlzugriffe hat, ist die relative Güte von LFU nicht beschränkt.



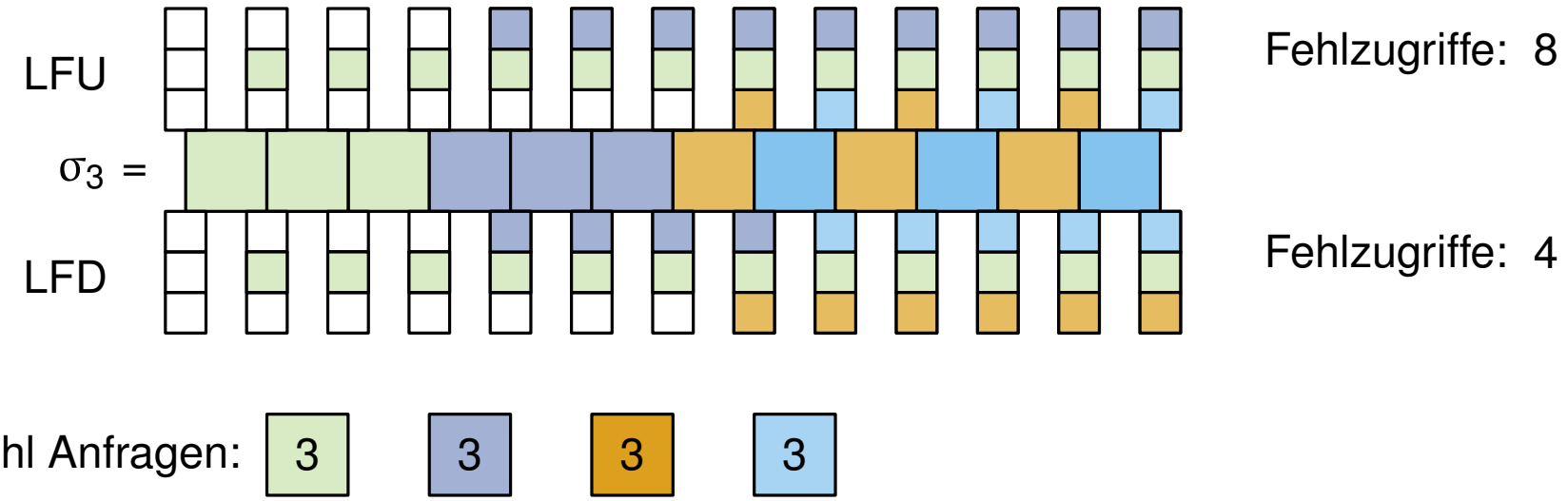
Kompetitive Paging-Algorithmen

LFU (Least Frequently Used) ist nicht kompetitiv, d.h. relative Güte c ist nicht beschränkt:

Sei k Cache-Größe und $P = \{p_1, \ldots, p_k, p_{k+1}\}$ Seiten im Hauptspeicher. Betrachte für beliebiges $m \in \mathbb{N}$ die Sequenz

$$\sigma_m = p_1^m, \ldots, p_{k-1}^m, (p_k, p_{k+1})^m$$

- LFU wird zuerst die Seiten $p_1, \ldots, p_{k-1}$ in den Cache aufnehmen (k-1 Fehlzugriffe).
- Nach $m \cdot (k-1)$ Anfragen, wird LFU für jedes Anfragepaar (p_k, p_{k+1}) einen Fehlzugriff haben. Folglich m weitere Fehlzugriffe.
- Da LFD nur k+1 Fehlzugriffe hat, ist die relative Güte von LFU nicht beschränkt.



Kompetitive Online-Algorithmen

Theorem 51: Es gibt keinen deterministischen Online-Algorithmus für Paging, der eine bessere relative Güte als k erreicht. Dabei gibt k die Größe des Caches an.

Konservative Paging-Algorithmen

(h,k)-Paging-Problem:

- Optimaler Offline-Algorithmus für Paging arbeitet auf Cache der Größe h.
- Online-Algorithmus für Paging arbeitet auf Cache der Größe k mit $k \geq h$.
- Online-Algorithmus bekommt größeren Cache um dessen Unwissenheit auszugleichen.

Béládys Anomalie: Für manche der Paging-Algorithmen kann man Zugriffssequenzen finden, sodass sie mit einem kleineren Cache weniger Fehlzugriffe liefern als mit einem größeren. FIFO ist ein solcher Algorithmus (siehe Übung).

Konservative Paging-Algorithmen

(h,k)-Paging-Problem:

- Optimaler Offline-Algorithmus für Paging arbeitet auf Cache der Größe h.
- Online-Algorithmus für Paging arbeitet auf Cache der Größe k mit $k \geq h$.
- Online-Algorithmus bekommt größeren Cache um dessen Unwissenheit auszugleichen.

Definition 36: Ein Paging-Algorithmus ALG mit Cache der Größe k heißt *konservativ*, falls für jede Anfragesequenz σ folgende Aussage gilt:
Jede Teilsequenz σ' von σ, die maximal k verschiedene Seiten enthält, erzeugt maximal k Fehlzugriffe während ALG die Sequenz σ abarbeitet.

Name	Ersetze Seite im Cache,...
FIFO: First In/First Out	die bereits am längsten im Cache ist.
LIFO: Last In/First Out	die am neusten im Cache ist.
LFU: Least Frequently Used	die bisher am wenigsten häufig angefordert wurde.
LRU: Least Recently Used	deren Anfrage am weitesten in der Vergangenheit liegt.
FWF: Flush When Full	(Gebe alle Seiten frei, wenn Cache voll ist.)

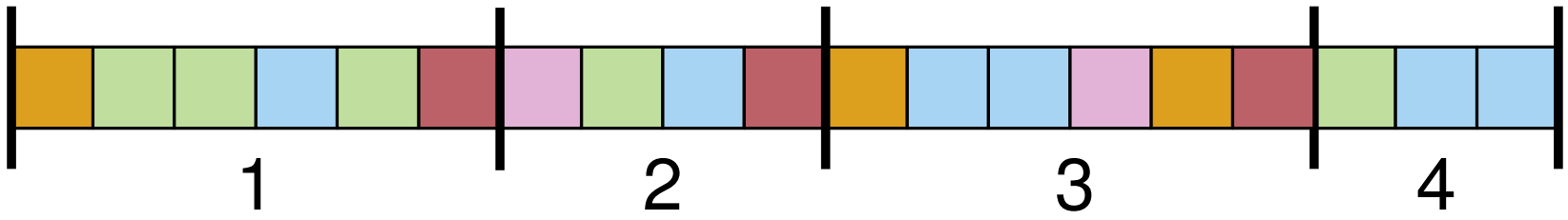
konservative Paging-Algorithmen

Konservative Paging-Algorithmen

Theorem 52: Jeder konservative Paging-Algorithmus mit Cache-Größe k ist $\frac{k}{k-h+1}$-kompetitiv bezüglich jedes optimalen Offline-Algorithmus mit Cache-Größe h.

k-Phasen-Partition einer Sequenz σ:

- Phase 0 ist die leere Sequenz.
- Für $i > 0$ sei Phase i längste Teilsequenz von σ, die sich direkt an Phase $i-1$ anschließt und maximal k verschiedene Seiten enthält.



4-Phasen-Partition

Konservative Paging-Algorithmen

Theorem 52: Jeder konservative Paging-Algorithmus mit Cache-Größe k ist $\frac{k}{k-h+1}$-kompetitiv bezüglich jedes optimalen Offline-Algorithmus mit Cache-Größe h.

Anwendung:

- Für (k, k)-Paging ist jeder konservative Paging-Algorithmus k-kompetitiv, was nach Theorem 51 der unteren Schranke entspricht.
- Für $h = \frac{k}{2}$ sind konservative Online-Algorithmen für Paging 2-kompetitiv.