Beyond Worst-case Analysis of Multicore Caching Strategies

Shahin Kamali

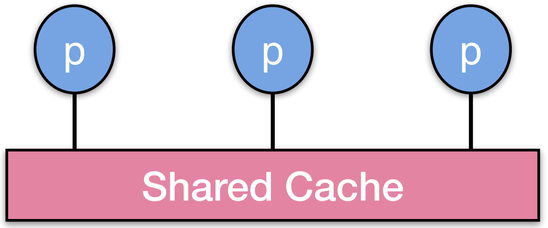
Helen Xu

University of Manitoba

APOCS 2021

Introduction

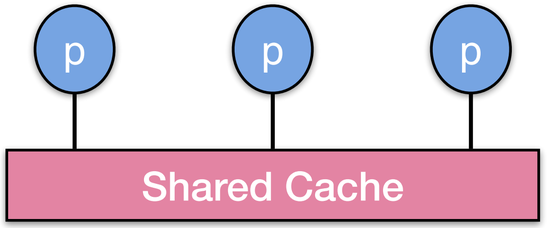
Caching for multicore architectures has been well-studied in practice [FSS06, QJPSE07, SRD04], but our theoretical understanding lags behind.



Introduction

Caching for multicore architectures has been well-studied in practice [FSS06, QJPSE07, SRD04], but our theoretical understanding lags behind.

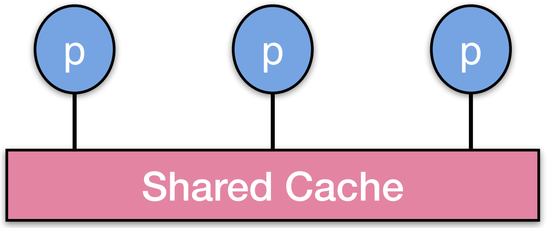
Multicore caching diverges from classical single-core caching because it has an added **scheduling aspect** due to fetch delay [LS12, KX20].



Introduction

Caching for multicore architectures has been well-studied in practice [FSS06, QJPSE07, SRD04], but our theoretical understanding lags behind.

Multicore caching diverges from classical single-core caching because it has an added **scheduling aspect** due to fetch delay [LS12, KX20].

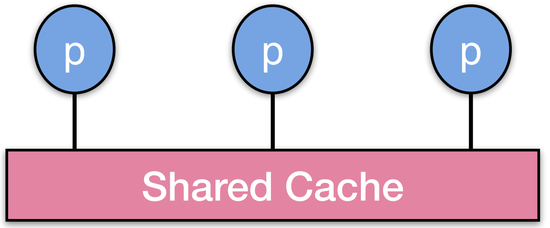


Furthest-In-Future (FIF) is the optimal algorithm [B66] and Least-Recently-Used (LRU) is k-competitive (where k is the size of the cache) [ST85] in **single-core**.

Introduction

Caching for multicore architectures has been well-studied in practice [FSS06, QJPSE07, SRD04], but our theoretical understanding lags behind.

Multicore caching diverges from classical single-core caching because it has an added **scheduling aspect** due to fetch delay [LS12, KX20].

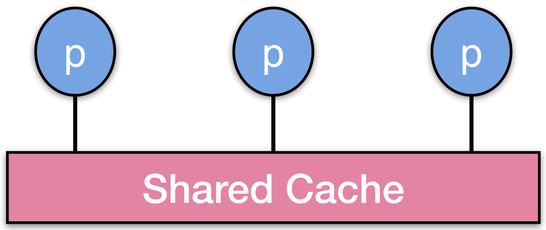


Furthest-In-Future (FIF) is the optimal algorithm [B66] and Least-Recently-Used (LRU) is k-competitive (where k is the size of the cache) [ST85] in **single-core**.

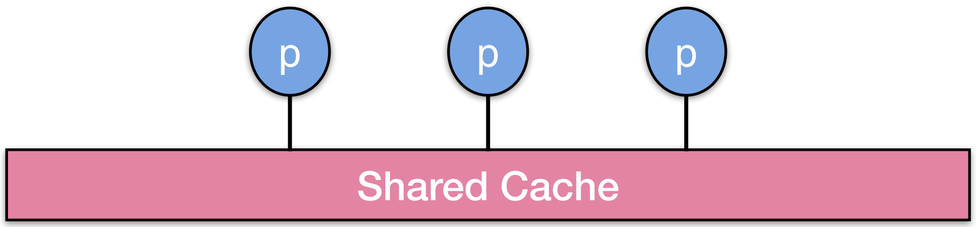
A large class of “lazy algorithms” (including LRU) **cannot be competitive** in multicore caching [KX20].

Relaxing Competitive Analysis

Resource augmentation [ST85] relaxes strict worst-case (competitive) analysis by giving online algorithms additional resources.



OPT

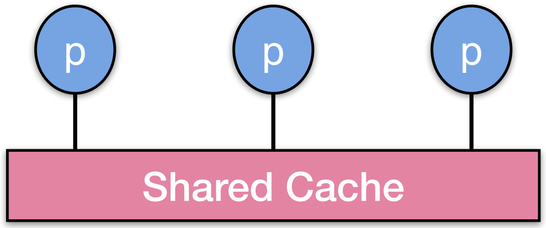


Online algorithm

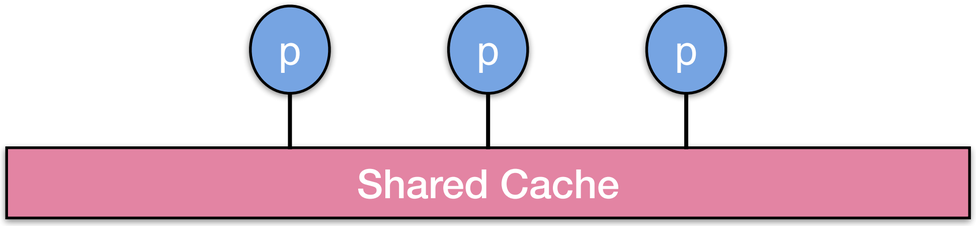
Relaxing Competitive Analysis

Resource augmentation [ST85] relaxes strict worst-case (competitive) analysis by giving online algorithms additional resources.

In single-core caching, doubling the space makes LRU 2-competitive. The same result **does not hold** in multicore [LS12].



OPT



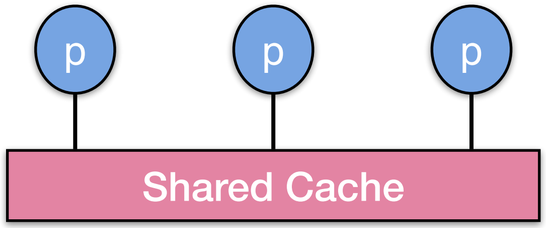
Online algorithm

Relaxing Competitive Analysis

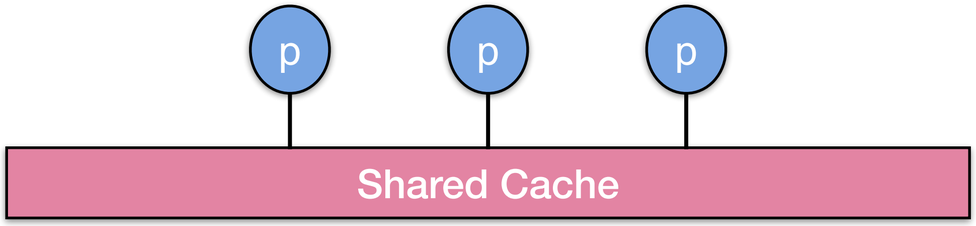
Resource augmentation [ST85] relaxes strict worst-case (competitive) analysis by giving online algorithms additional resources.

In single-core caching, doubling the space makes LRU 2-competitive. The same result **does not hold** in multicore [LS12].

In multicore caching, a constant factor in resource augmentation yields an **O(log p)-competitive** algorithm (where p is the number of cores) [ABDKPS20].



OPT



Online algorithm

Alternative Measures

Many alternative measures have been proposed for single-core caching but have not been studied in the **parallel setting** [Y94, Y98, KP00, KPR92, BLN01, BFL05, BB94].

Alternative Measures

Many alternative measures have been proposed for single-core caching but have not been studied in the **parallel setting** [Y94, Y98, KP00, KPR92, BLN01, BFL05, BB94].

Bijective analysis [ADL07, ADL08, AS09, AS13] directly compares online algorithms and has been used to separate LRU in the single-core setting from other online algorithms.

Alternative Measures

Many alternative measures have been proposed for single-core caching but have not been studied in the **parallel setting** [Y94, Y98, KP00, KPR92, BLN01, BFL05, BB94].

Bijective analysis [ADL07, ADL08, AS09, AS13] directly compares online algorithms and has been used to separate LRU in the single-core setting from other online algorithms.

Overly pessimistic

Bijective analysis 💚

Competitive analysis 😡

Alternative Measures

Many alternative measures have been proposed for single-core caching but have not been studied in the **parallel setting** [Y94, Y98, KP00, KPR92, BLN01, BFL05, BB94].

Bijective analysis [ADL07, ADL08, AS09, AS13] directly compares online algorithms and has been used to separate LRU in the single-core setting from other online algorithms.

	Overly pessimistic	**Compares online algorithms directly**
Bijective analysis	💚	💚
Competitive analysis	😡	😡

Alternative Measures

Many alternative measures have been proposed for single-core caching but have not been studied in the **parallel setting** [Y94, Y98, KP00, KPR92, BLN01, BFL05, BB94].

Bijective analysis [ADL07, ADL08, AS09, AS13] directly compares online algorithms and has been used to separate LRU in the single-core setting from other online algorithms.

	Overly pessimistic	Compares online algorithms directly	Applicability
Bijective analysis	💚	💚	😡
Competitive analysis	😡	😡	💚

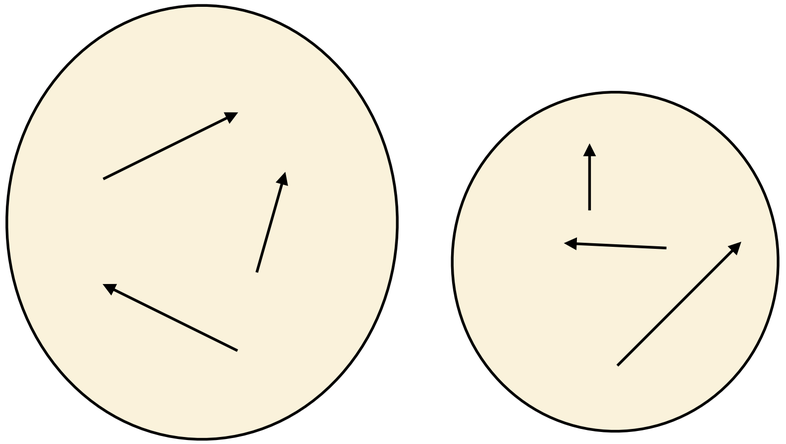
Cyclic Analysis for Multicore Caching

We introduce **cyclic analysis**, an alternative measure for online algorithms inspired by bijective analysis.

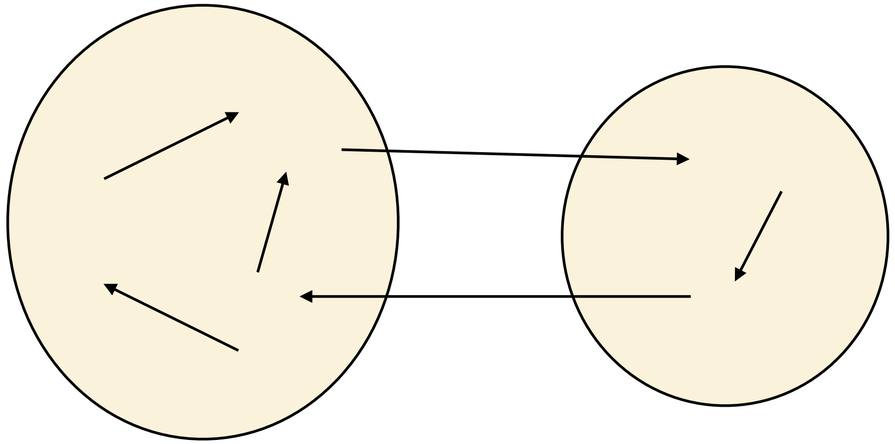
Cyclic Analysis for Multicore Caching

We introduce **cyclic analysis**, an alternative measure for online algorithms inspired by bijective analysis.

Bijective analysis compares inputs of the same “length”, while cyclic analysis compares algorithms **across all inputs** rather than within partitions.



Bijective analysis



Cyclic analysis

Cyclic Analysis Measure

	Applicability
Bijective analysis	
Cyclic analysis	

Cyclic Analysis Measure

	Applicability	Compares online algorithms directly
Bijective analysis	😡	💚
Cyclic analysis	💚	💚

Cyclic Analysis Measure

Used to separate LRU

	Applicability	Compares online algorithms directly	Can incorporate restrictions on universe of inputs
Bijective analysis	😡	💚	💚
Cyclic analysis	💚	💚	💚

Separating LRU With Cyclic Analysis

An online algorithm A is **no worse** than online algorithm B under **cyclic analysis** on the universe of inputs I if there exists a bijection $\pi : I \leftrightarrow I$ satisfying $A(R) \leq B(\pi(R))$ for all $R \in I$. We denote this by $A \leq_c B$.

Algorithm cost

Separating LRU With Cyclic Analysis

An online algorithm A is **no worse** than online algorithm B under **cyclic analysis** on the universe of inputs I if there exists a bijection $\pi : I \leftrightarrow I$ satisfying $A(R) \leq B(\pi(R))$ for all $R \in I$. We denote this by $A \leq_c B$.

Algorithm cost

Main proof

Separating LRU With Cyclic Analysis

An online algorithm A is **no worse** than online algorithm B under **cyclic analysis** on the universe of inputs I if there exists a bijection $\pi : I \leftrightarrow I$ satisfying $A(R) \leq B(\pi(R))$ for all $R \in I$. We denote this by $A \leq_c B$.

Main proof

Algorithm cost

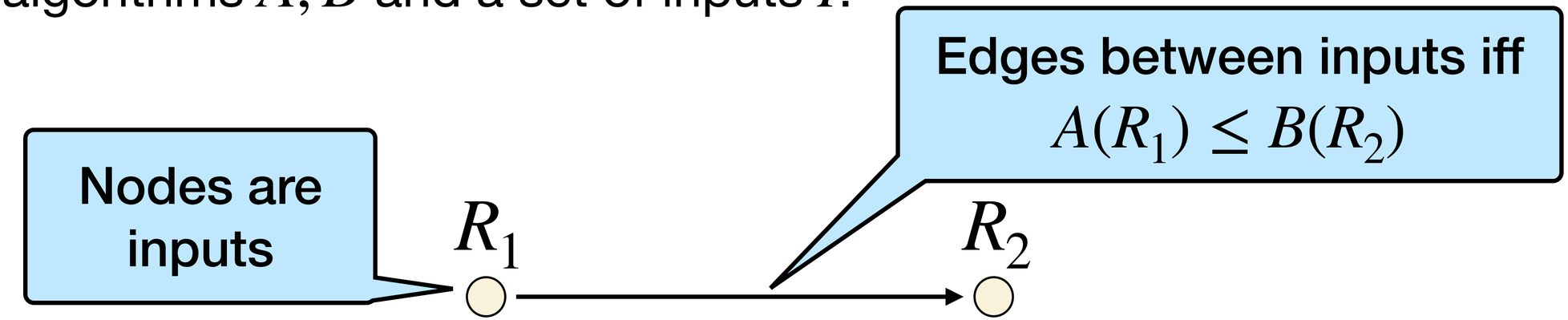
Main theorem: For any lazy caching algorithm A,

$LRU \leq_c A$ **on inputs with locality of reference.**

Cyclic Analysis: A Tutorial

Input-cost graph: modeling mappings between inputs with different costs.

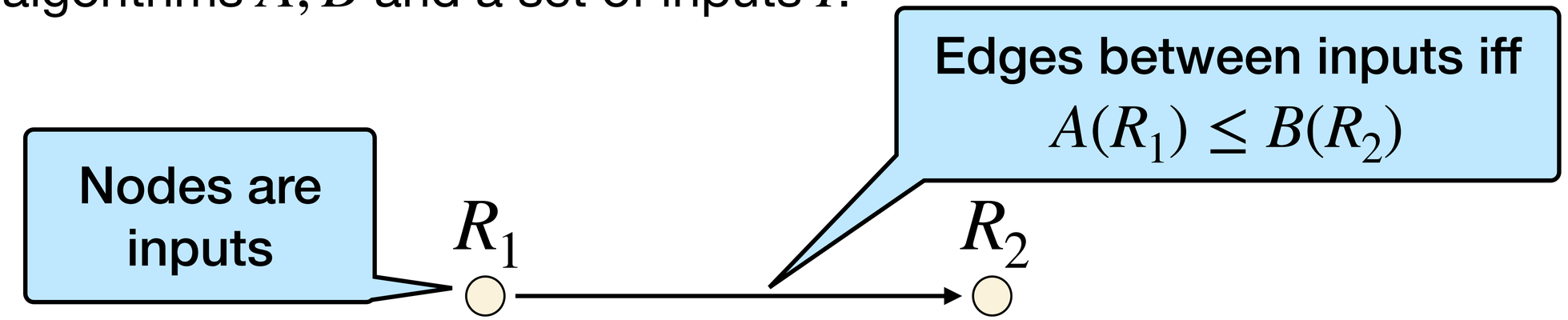
Given algorithms A, B and a set of inputs I:



Cyclic Analysis: A Tutorial

Input-cost graph: modeling mappings between inputs with different costs.

Given algorithms A, B and a set of inputs I:

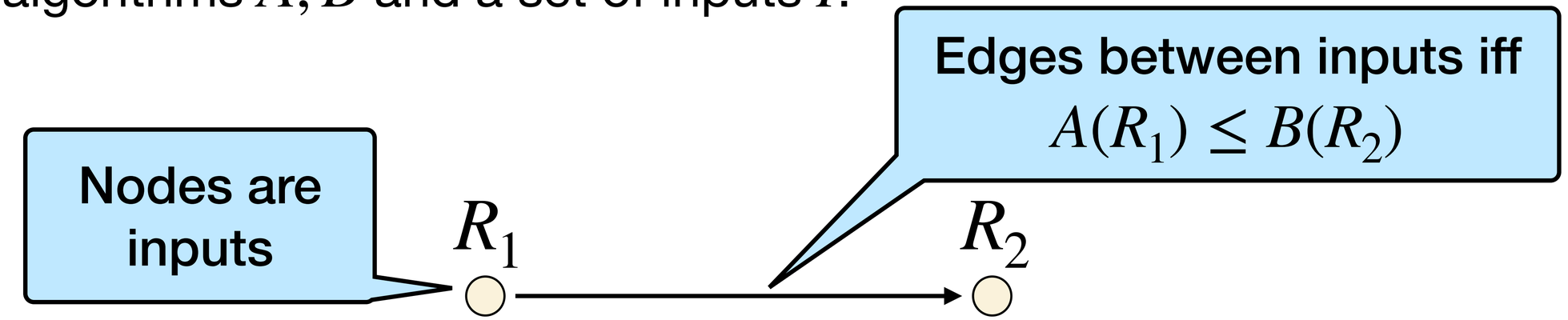


Goal: Define a bijection $\pi : I \leftrightarrow I$ such that for all inputs $R \in I, A(R) \leq B(\pi(R))$.

Cyclic Analysis: A Tutorial

Input-cost graph: modeling mappings between inputs with different costs.

Given algorithms A, B and a set of inputs I:



Goal: Define a bijection $\pi : I \leftrightarrow I$ such that for all inputs $R \in I, A(R) \leq B(\pi(R))$.

Analysis technique: Partition the input-cost graph and analyze cycles in the resulting subgraphs.

Cyclic Analysis: A Tutorial

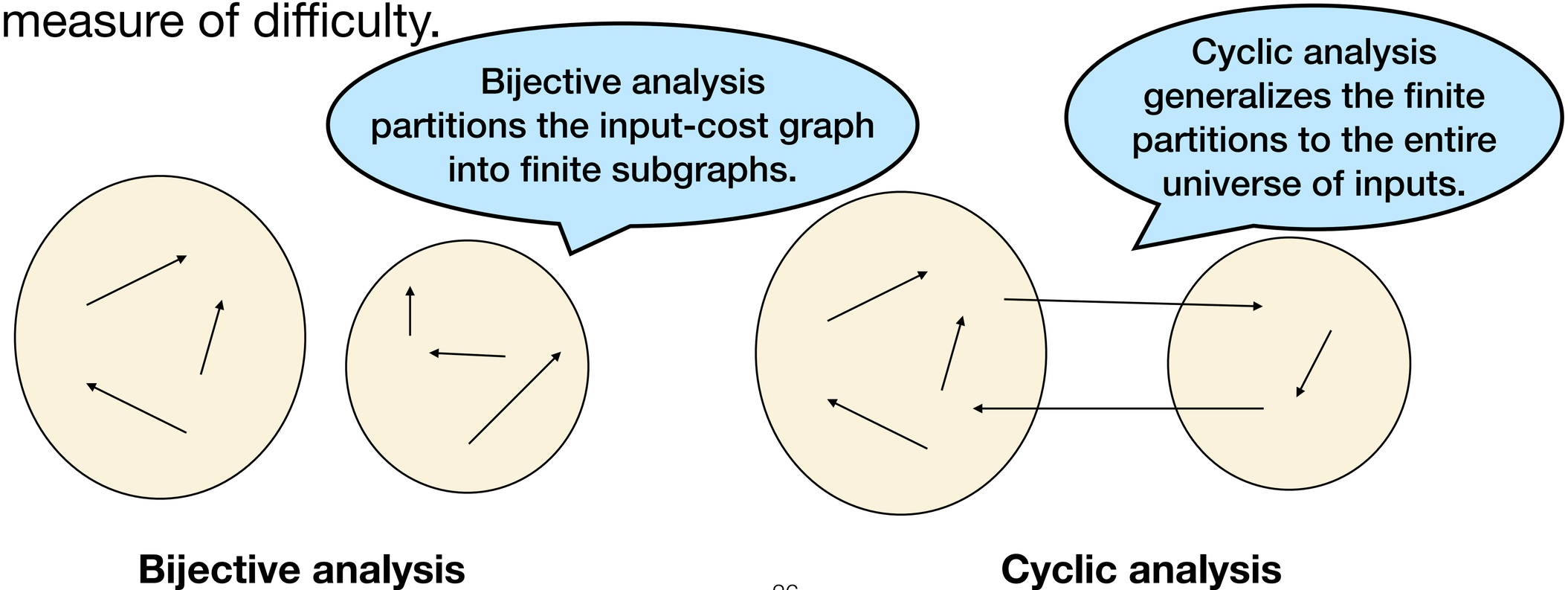
Partitioning the input-cost graph: For problems such as single-core caching, the length of inputs is well defined.

For general online problems (e.g. multicore caching), the length is not a measure of difficulty.

Cyclic Analysis: A Tutorial

Partitioning the input-cost graph: For problems such as single-core caching, the length of inputs is well defined.

For general online problems (e.g. multicore caching), the length is not a measure of difficulty.



Example: Bijective vs Cyclic Analysis

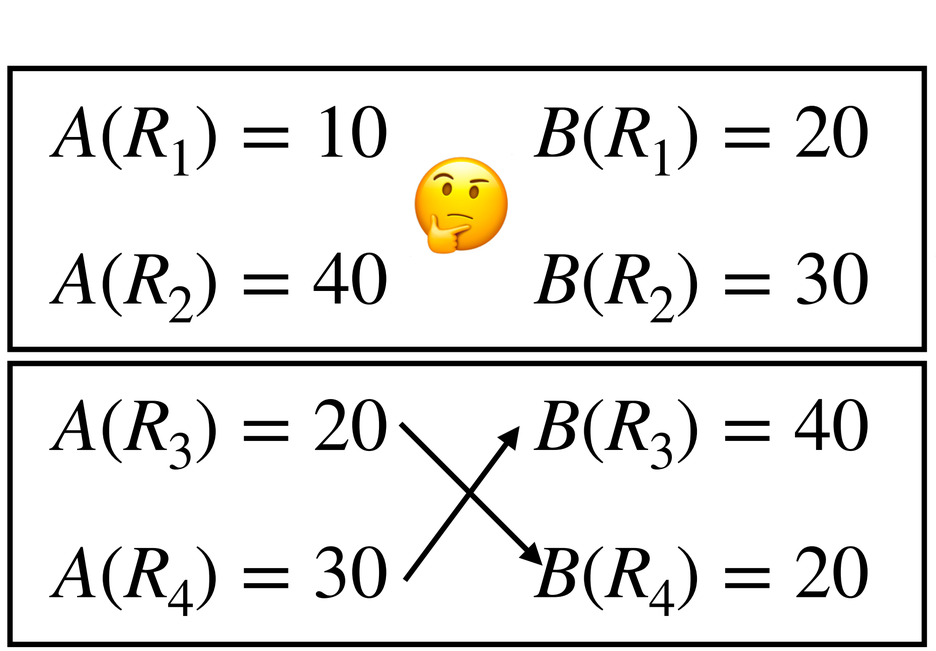
Given algorithms A, B for some online problem P.

$A(R_1) = 10$	$B(R_1) = 20$
$A(R_2) = 40$	$B(R_2) = 30$
$A(R_3) = 20$	$B(R_3) = 40$
$A(R_4) = 30$	$B(R_4) = 20$

Same lengths

Example: Bijective vs Cyclic Analysis

Given algorithms A, B for some online problem P.

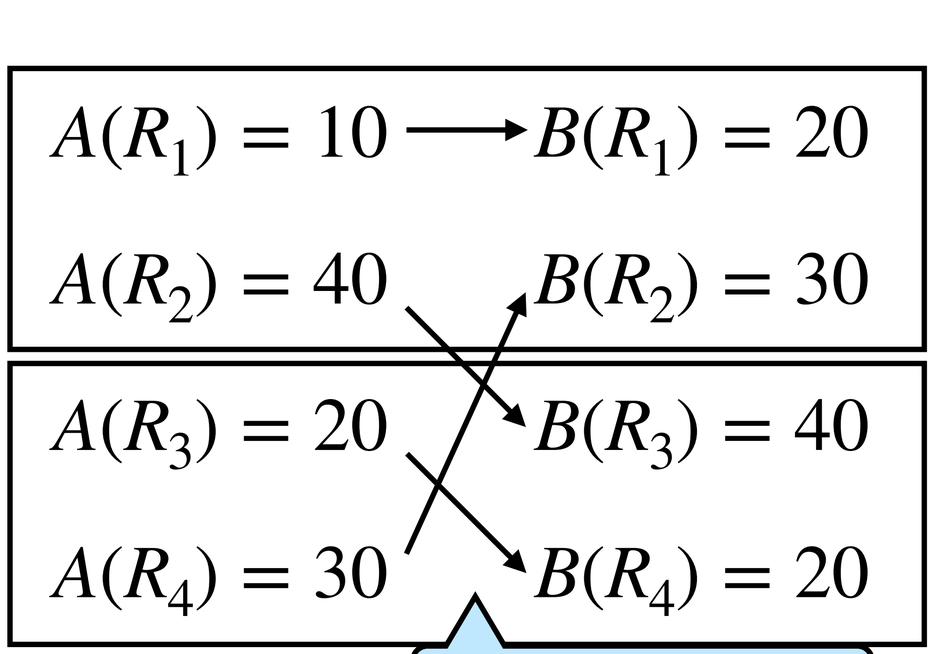


Same lengths

Under bijective analysis: no way to establish the advantage of A over B

Example: Bijective vs Cyclic Analysis

Given algorithms A, B for some online problem P.



Same lengths

Under bijective analysis: no way to establish the advantage of A over B

Under cyclic analysis:

$A <_C B$

Properties of Cyclic Analysis

The cyclic analysis measure has several **natural properties**:

Transitivity ($A \leq_c B, B \leq_c C \rightarrow A \leq_c C$).

Directedness ($A <_c B$ implies $B \not\leq_c A$).

Properties of Cyclic Analysis

The cyclic analysis measure has several **natural properties**:

Transitivity ($A \leq_c B, B \leq_c C \rightarrow A \leq_c C$).

Directedness ($A <_c B$ implies $B \not\leq_c A$).

Cyclic analysis can be used to show **“to-be-expected” relationships** between multicore caching algorithms:

All **lazy algorithms are equivalent**.

Any lazy algorithm A is **strictly better than Flush-When-Full** (for p=2).

Free-Interleaving Model [LS12]

Problem setup: given a multicore processor with p cores and k pages.

An input R is formed by p request sequences $R = R_1, \ldots, R_p$ composed of page requests, where each core must serve its corresponding request sequence.

Page requests arrive at discrete timesteps.

Free-Interleaving Model [LS12]

Problem setup: given a multicore processor with p cores and k pages.

An input R is formed by p request sequences $R = R_1, \ldots, R_p$ composed of page requests, where each core must serve its corresponding request sequence.

Page requests arrive at discrete timesteps.

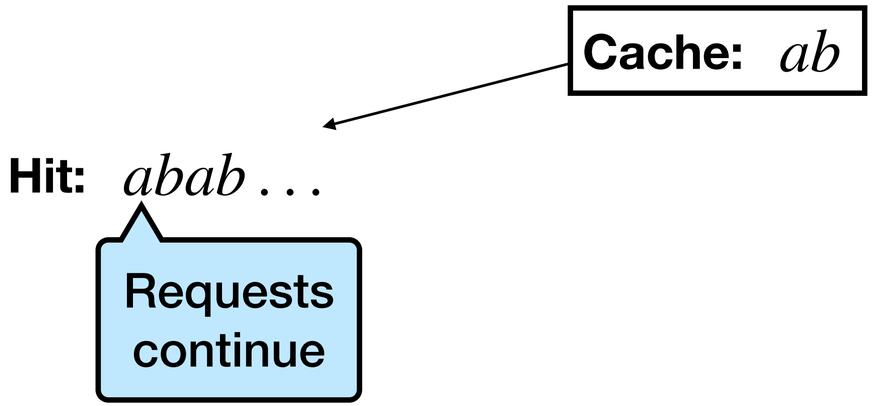
Cache: ab

Free-Interleaving Model [LS12]

Problem setup: given a multicore processor with p cores and k pages.

An input R is formed by p request sequences $R = R_1, \ldots, R_p$ composed of page requests, where each core must serve its corresponding request sequence.

Page requests arrive at discrete timesteps.

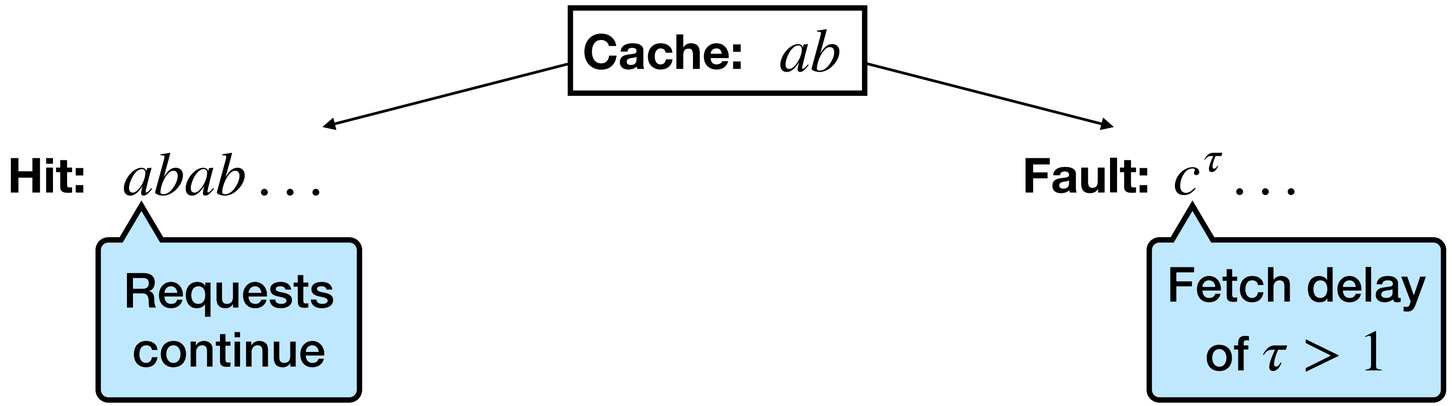


Free-Interleaving Model [LS12]

Problem setup: given a multicore processor with p cores and k pages.

An input R is formed by p request sequences $R = R_1, \ldots, R_p$ composed of page requests, where each core must serve its corresponding request sequence.

Page requests arrive at discrete timesteps.



Locality of Reference

Main theorem: For any lazy caching algorithm A,

$LRU \leq_c A$ **on inputs with locality of reference.**

Intuitively: an input has locality of reference if the number of distinct pages requested is bounded [AFG02].

Locality of Reference

Main theorem: For any lazy caching algorithm A,

$LRU \leq_c A$ on inputs with locality of reference.

Intuitively: an input has locality of reference if the number of distinct pages requested is bounded [AFG02].

$abcde\ldots$

No locality

😡

$ababa\ldots$

Some locality

💚

Advantage of LRU with Locality

Goal: For any lazy algorithm *A*, define a bijection π over the space of inputs with locality I satisfying $LRU(R) \leq A(\pi(R))$ for all $R \in I$.

Intuition: Analyze timesteps where A and LRU differ. Suppose at time t, LRU evicts page σ_{LRU}, while A evicts σ_{NLRU}.

Advantage of LRU with Locality

Goal: For any lazy algorithm A, define a bijection π over the space of inputs with locality I satisfying $LRU(R) \leq A(\pi(R))$ for all $R \in I$.

Intuition: Analyze timesteps where A and LRU differ. Suppose at time t, LRU evicts page σ_{LRU}, while A evicts σ_{NLRU}.

Case 1: Swapping $\sigma_{LRU}, \sigma_{NLRU}$ in the continuation maintains locality.
$\pi(R)$ swaps them in the continuation.
$LRU(R) = A(\pi(R))$. 😐

Advantage of LRU with Locality

Goal: For any lazy algorithm A, define a bijection π over the space of inputs with locality I satisfying $LRU(R) \leq A(\pi(R))$ for all $R \in I$.

Intuition: Analyze timesteps where A and LRU differ. Suppose at time t, LRU evicts page σ_{LRU}, while A evicts σ_{NLRU}.

Case 1: Swapping $\sigma_{LRU}, \sigma_{NLRU}$ in the continuation maintains locality.
$\pi(R)$ swaps them in the continuation.
$LRU(R) = A(\pi(R))$. 😐

Case 2: Swapping $\sigma_{LRU}, \sigma_{NLRU}$ in the continuation does not maintain locality.
To show: $LRU(R) \leq A(\pi(R))$

Constructing a Mapping

Case 2: Swapping $\sigma_{LRU}, \sigma_{NLRU}$ in the continuation does not maintain locality.

To show: $LRU(R) \leq A(\pi(R))$

If there is a miss before a request to σ_{NLRU} after time t, set $\pi(R) = R$.

Therefore, $LRU(R) = A(\pi(R))$.

😐

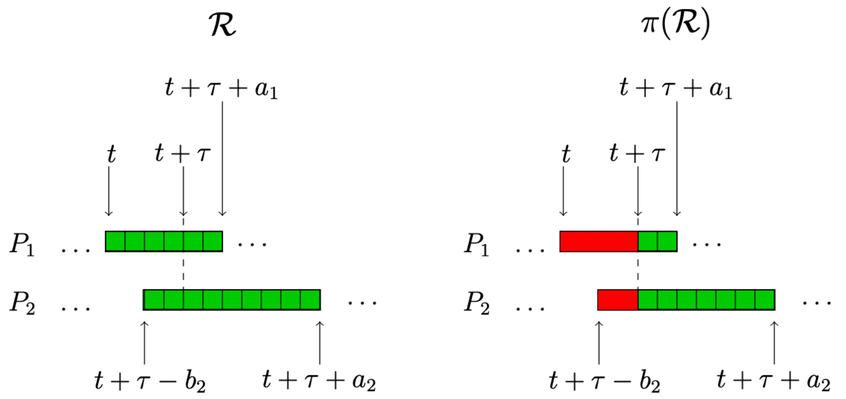
Constructing a Mapping

Case 2: Swapping $\sigma_{LRU}, \sigma_{NLRU}$ in the continuation does not maintain locality.
To show: $LRU(R) \leq A(\pi(R))$

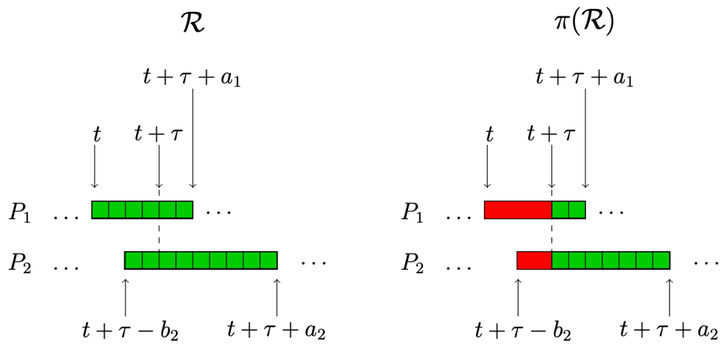
If there is a miss before a request to σ_{NLRU} after time *t*, set $\pi(R) = R$.
Therefore, $LRU(R) = A(\pi(R))$.
😐

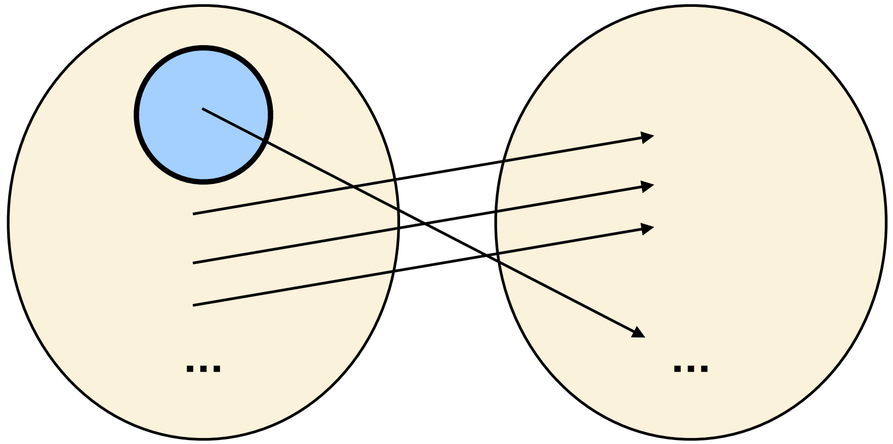
If there was not a miss before a request to σ_{NLRU} after time *t*, LRU hits on the first request to σ_{NLRU}.

💚 $LRU(R) \leq A(\pi(R))$



Mapping to Inputs of Different Length

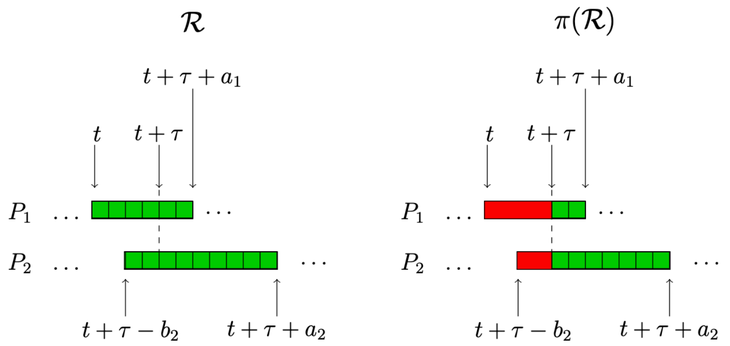


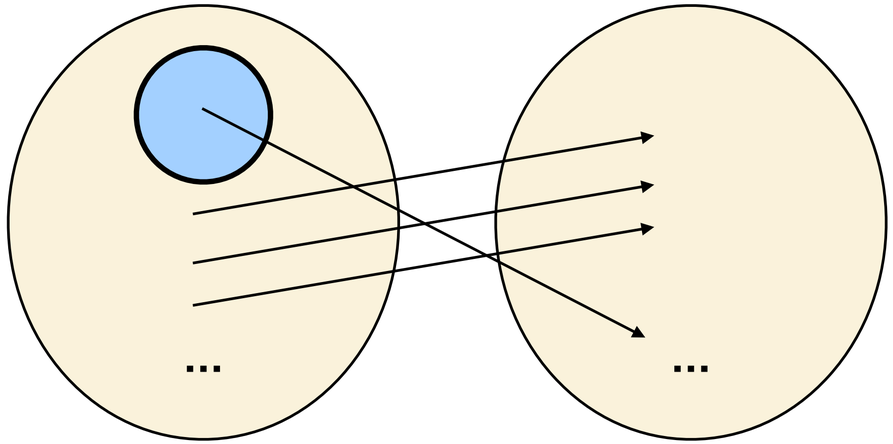


$$LRU(R) = A(\pi(R))$$

The previous two cases cover entire codomain, but **not the domain**.

Mapping to Inputs of Different Length





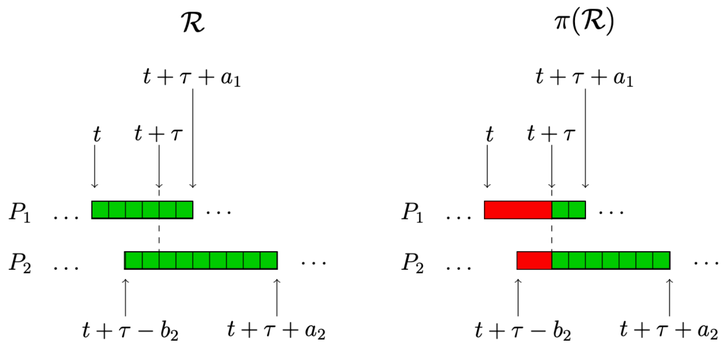
$$LRU(R) = A(\pi(R))$$

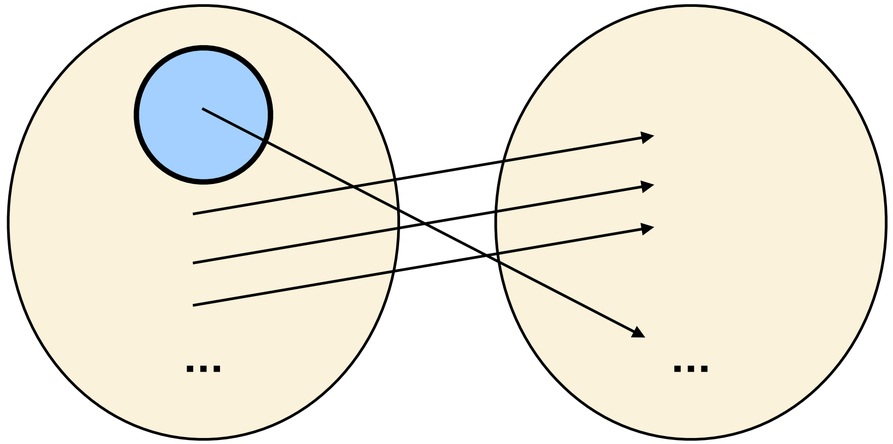
The previous two cases cover entire codomain, but **not the domain**.

For remaining inputs, map them arbitrarily to **inputs of higher cost**.

$$LRU(R) < A(\pi(R))$$

Mapping to Inputs of Different Length





$$LRU(R) = A(\pi(R))$$

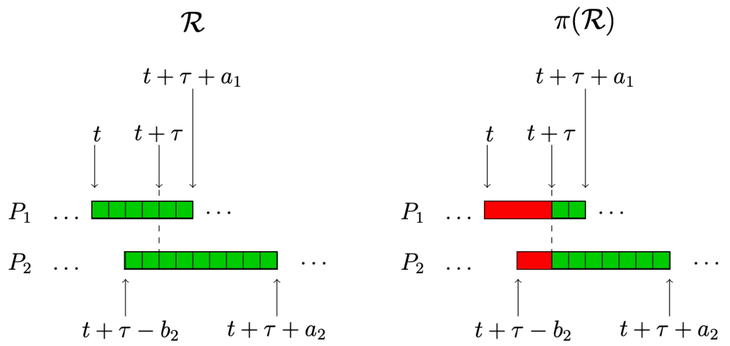
The previous two cases cover entire codomain, but **not the domain**.

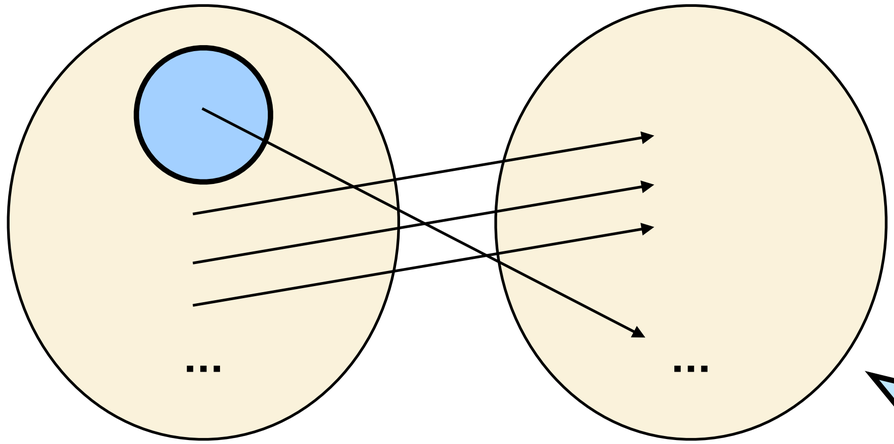
For remaining inputs, map them arbitrarily to **inputs of higher cost**.

Equivalence with bijective mapping

$$LRU(R) < A(\pi(R))$$

Mapping to Inputs of Different Length





$LRU(R) = A(\pi(R))$

$LRU(R) \leq A(\pi(R))$

The previous two cases cover entire codomain, but **not the domain**.

For remaining inputs, map them arbitrarily to **inputs of higher cost**.

Equivalence with bijective mapping

$LRU(R) < A(\pi(R))$

Conclusions

Cyclic analysis compares online algorithms directly on a set of inputs and establishes the **advantage of LRU** on inputs with locality.

Conclusions

Cyclic analysis compares online algorithms directly on a set of inputs and establishes the **advantage of LRU** on inputs with locality.

Since it is more general and flexible than bijective analysis, we expect it to be useful in the **study of other online problems**.

Conclusions

Cyclic analysis compares online algorithms directly on a set of inputs and establishes the **advantage of LRU** on inputs with locality.

Since it is more general and flexible than bijective analysis, we expect it to be useful in the **study of other online problems**.

Multicore caching is **well-studied in practice** and motivated by hierarchical memory.

Conclusions

Cyclic analysis compares online algorithms directly on a set of inputs and establishes the **advantage of LRU** on inputs with locality.

Since it is more general and flexible than bijective analysis, we expect it to be useful in the **study of other online problems**.

Multicore caching is **well-studied in practice** and motivated by hierarchical memory.

The need for **theoretical understanding** will only grow as multicore architectures become more prevalent.