Advanced Algorithmic Techniques (COMP523)

Online Algorithms

Recap and plan

- **Last lectures:**
  - Randomised Algorithms
  - Randomised approximation algorithms.
    - Applications: MAX-SAT, MAX-3SAT, MAX-CUT
- **Final two lectures:**
  - Online algorithms.
  - Competitive Analysis.

Motivating Examples

Motivating Examples

- Suppose that you need to take 4 modules in your Masters programme, but you don’t know the difficulty, the content of each module, or the lecturer of the module. You need to make a decision based on limited information (topic, last year’s curriculum, etc).

Motivating Examples

- Suppose that you need to take 4 modules in your Masters programme, but you don’t know the difficulty, the content of each module, or the lecturer of the module. You need to make a decision based on limited information (topic, last year’s curriculum, etc).

- Suppose that you have completed your Masters programme successfully and now you are looking for jobs. You have made several applications and you receive an offer from some company. Should you accept it, or should you wait to see if you might get a better offer from another company?

Motivating Examples

- Suppose that you need to take 4 modules in your Masters programme, but you don't know the difficulty, the content of each module, or the lecturer of the module. You need to make a decision based on limited information (topic, last year's curriculum, etc).

- Suppose that you have completed your Masters programme successfully and now you are looking for jobs. You have made several applications and you receive an offer from some company. Should you accept it, or should you wait to see if you might get a better offer from another company?

- Life is an *online setting*…

Hindsight is 20/20

Hindsight is 20/20

- If you knew what would happen in the future, you could make all the right decisions.

Hindsight is 20/20

- If you knew what would happen in the future, you could make all the right decisions.
  - But you are not *clairvoyant*.

Hindsight is 20/20

- If you knew what would happen in the future, you could make all the right decisions.
  - But you are not *clairvoyant*.
- Let's say that you make a series of local (*myopic*) decisions, based only on information that you have seen so far (and possibly what you *expect* to see in the future).

Hindsight is 20/20

- If you knew what would happen in the future, you could make all the right decisions.
  - But you are not *clairvoyant*.
- Let's say that you make a series of local (*myopic*) decisions, based only on information that you have seen so far (and possibly what you *expect* to see in the future).
  - You can compare the quality of your decisions to that of the clairvoyant.

Hindsight is 20/20

- If you knew what would happen in the future, you could make all the right decisions.
  - But you are not *clairvoyant*.
- Let's say that you make a series of local (*myopic*) decisions, based only on information that you have seen so far (and possibly what you *expect* to see in the future).
  - You can compare the quality of your decisions to that of the clairvoyant.
  - If they are not much worse, then you can convince yourself that you have made good decisions.

Let’s talk about algorithms

Let's talk about algorithms

- Suppose that the *input* of a problem P is given to you in *steps*.

Let's talk about algorithms

- Suppose that the *input* of a problem P is given to you in *steps*.
- You have to make a decision in every step.

Let’s talk about algorithms

- Suppose that the *input* of a problem P is given to you in *steps*.
- You have to make a decision in every step.
- The goal is to *optimise some objective* (e.g., minimise some cost).

Let’s talk about algorithms

- Suppose that the *input* of a problem P is given to you in *steps*.
- You have to make a decision in every step.
- The goal is to *optimise some objective* (e.g., minimise some cost).
- You don’t know the length of the input - the *input supply* might stop at any point.

Let's talk about algorithms

- Suppose that the *input* of a problem P is given to you in *steps*.
- You have to make a decision in every step.
- The goal is to *optimise some objective* (e.g., minimise some cost).
- You don't know the length of the input - the *input supply* might stop at any point.
- You will compare against the *offline optimal algorithm*, which knows the future, and computes the optimal solution on the entire input.

Online algorithms

- Online Algorithm: An algorithm that must make decisions now about events that will happen in the future, without having knowledge of these events.

Recall: Load Balancing

- We have a set of m *identical* machines $M_1, \ldots, M_m$
- We have a set of n jobs, with job j having processing time t_j.
- We want to assign every job to some machine.
- Let $A(i)$ be the set of jobs assigned to machine i.
- The load of machine i is $$T_i = \sum_{j \in A(i)} t_j$$
- The goal is to minimise the makespan, i.e.,

  $T = \max_i T_i$

Online Load Balancing

- We have a set of m *identical* machines $M_1, \ldots, M_m$
- We have a set of n jobs, with job j having processing time t_j.
    - The jobs arrive over time, one in each time step.
- We want to assign every job to some machine.
    - We will assign a job immediately upon arrival to some machine.
- Let $A(i)$ be the set of jobs assigned to machine i.
- The load of machine i is $$T_i = \sum_{j \in A(i)} t_j$$
- The goal is to minimise the makespan, i.e.,

    $T = \max_i T_i$

Example

jobs M_1 M_2 M_3

Example

2

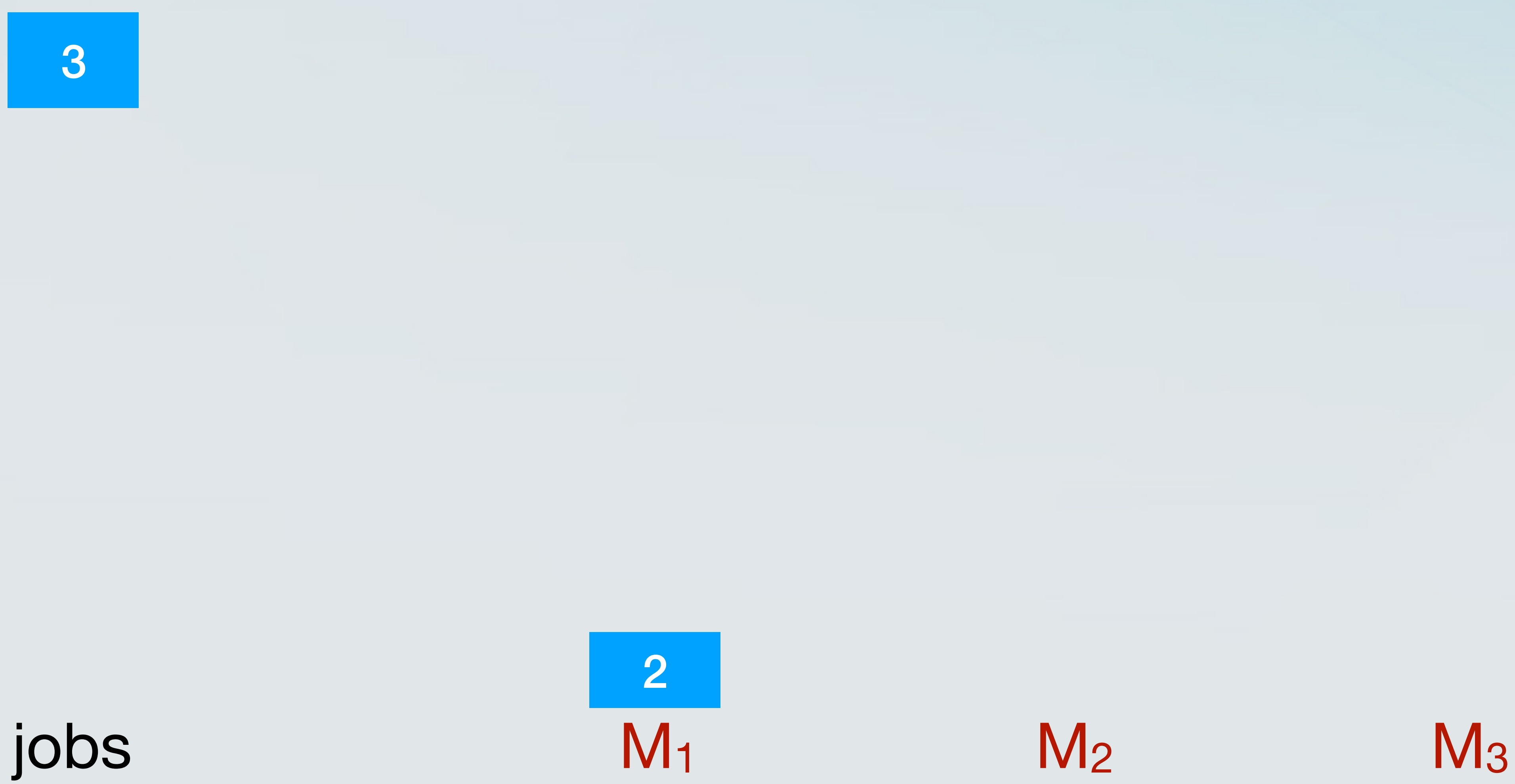
jobs M_1 M_2 M_3

Example

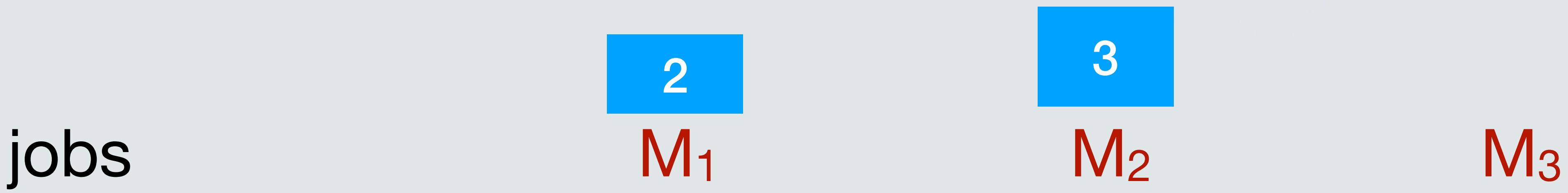
2

jobs M_1 M_2 M_3

Example

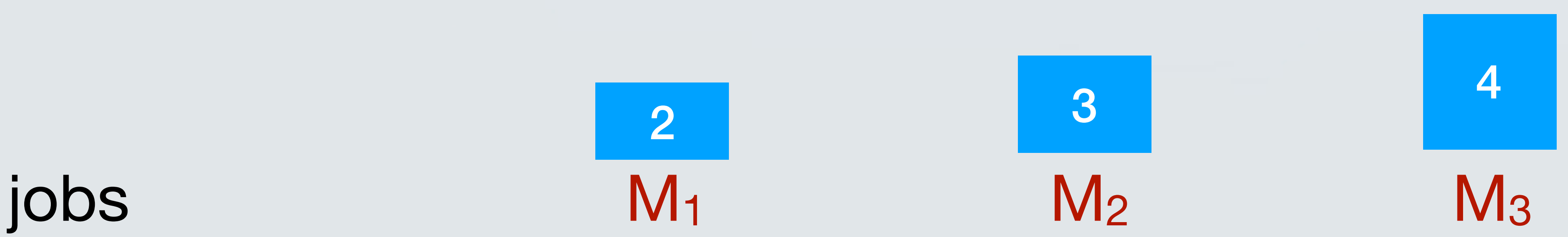


Example

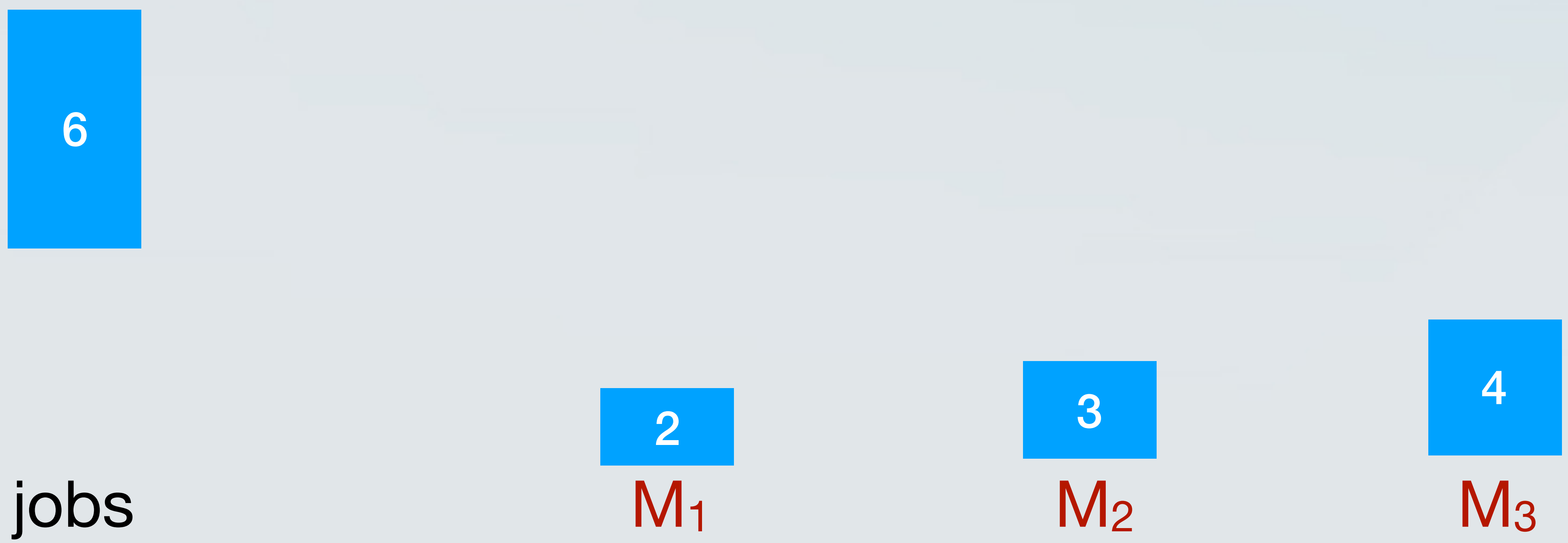


Example

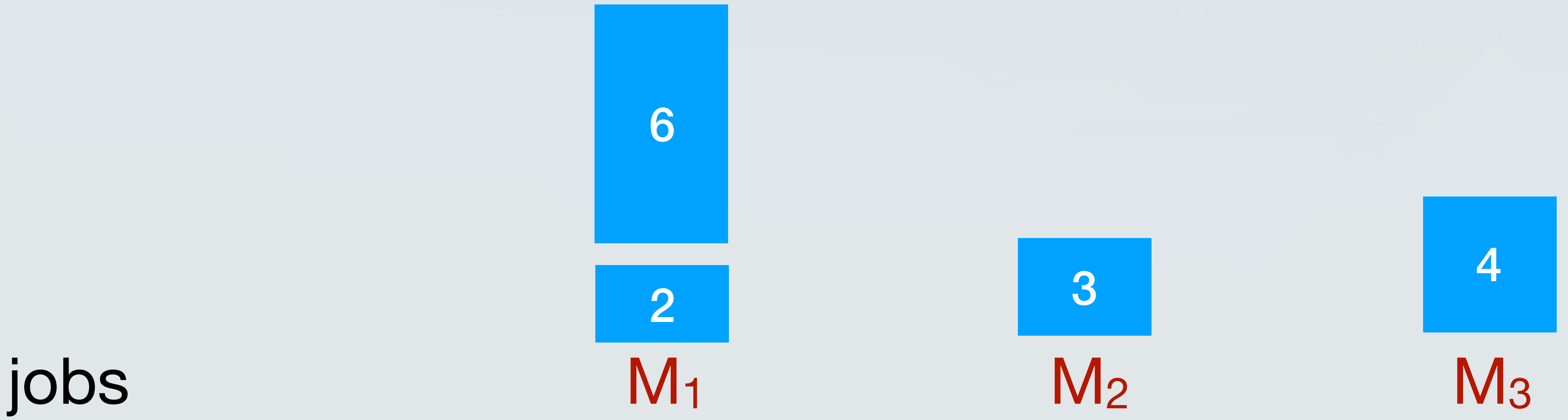
Example



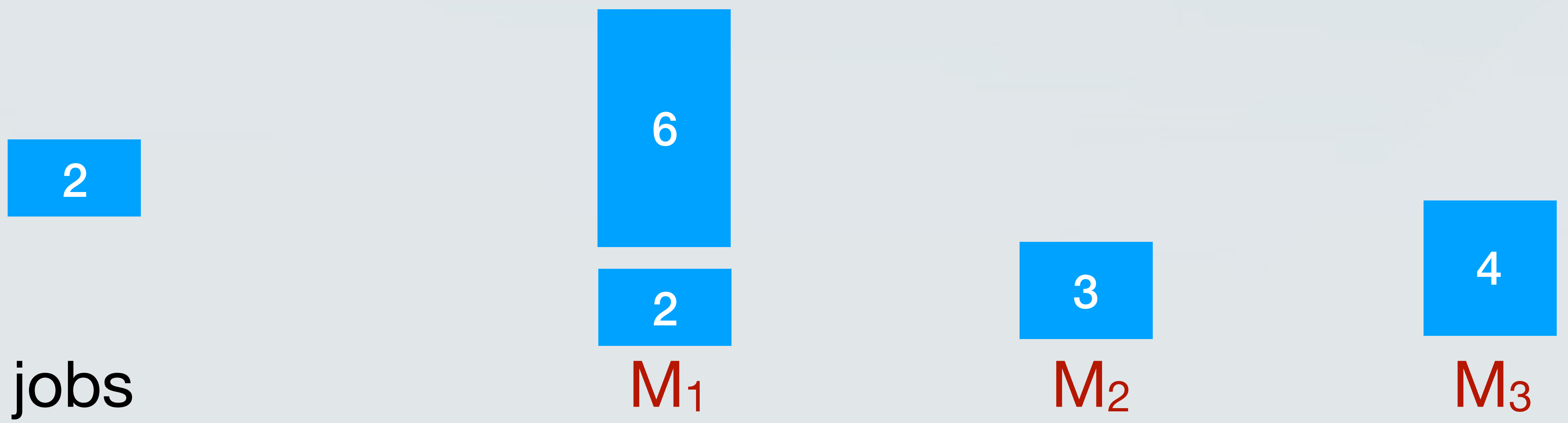
Example



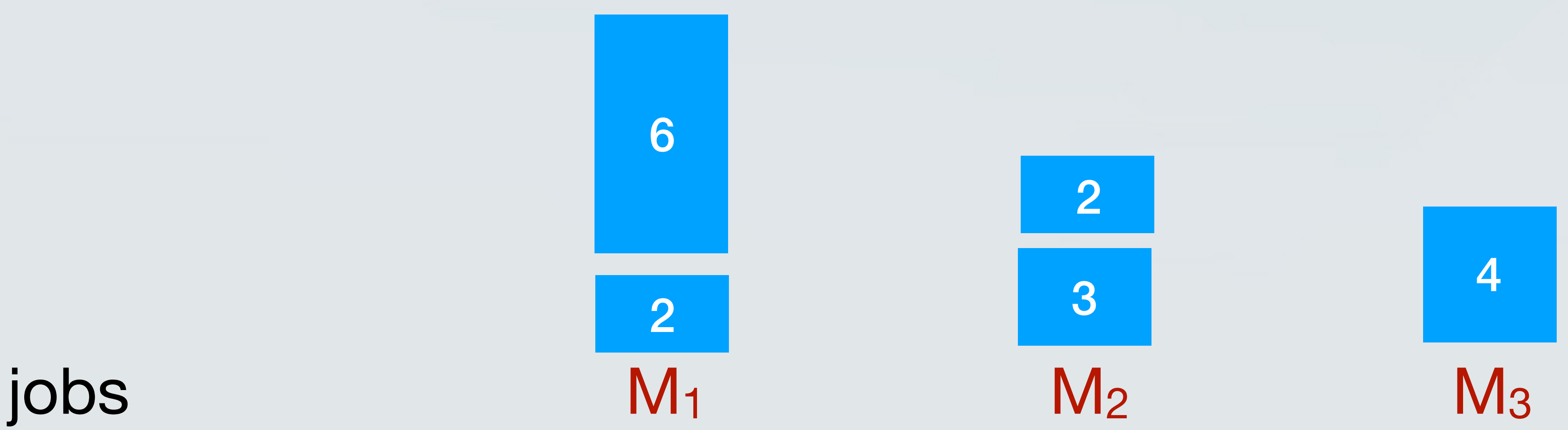
Example



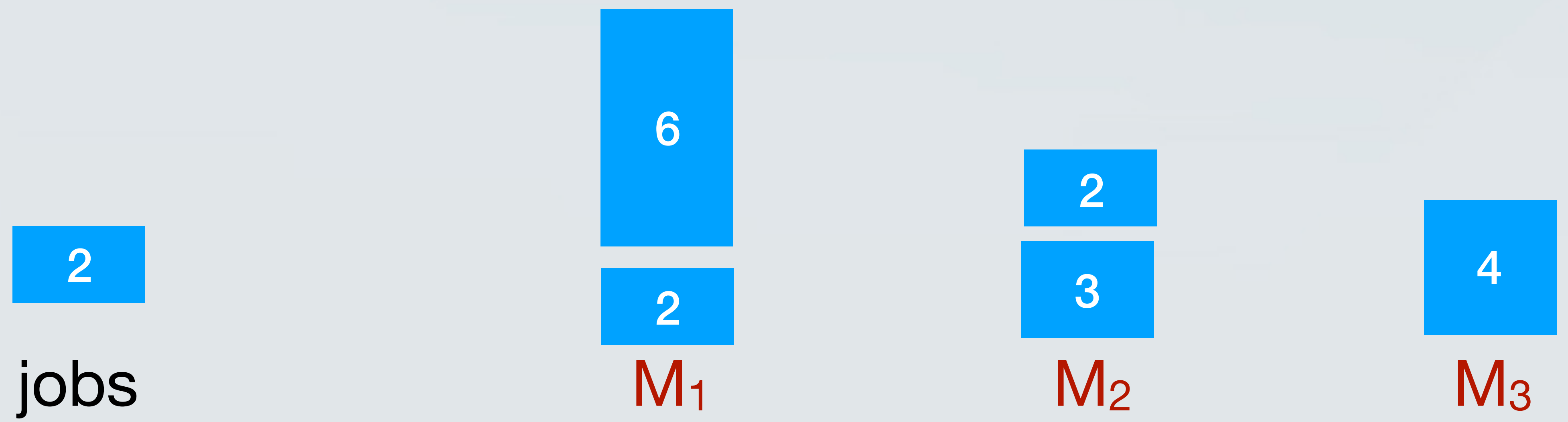
Example



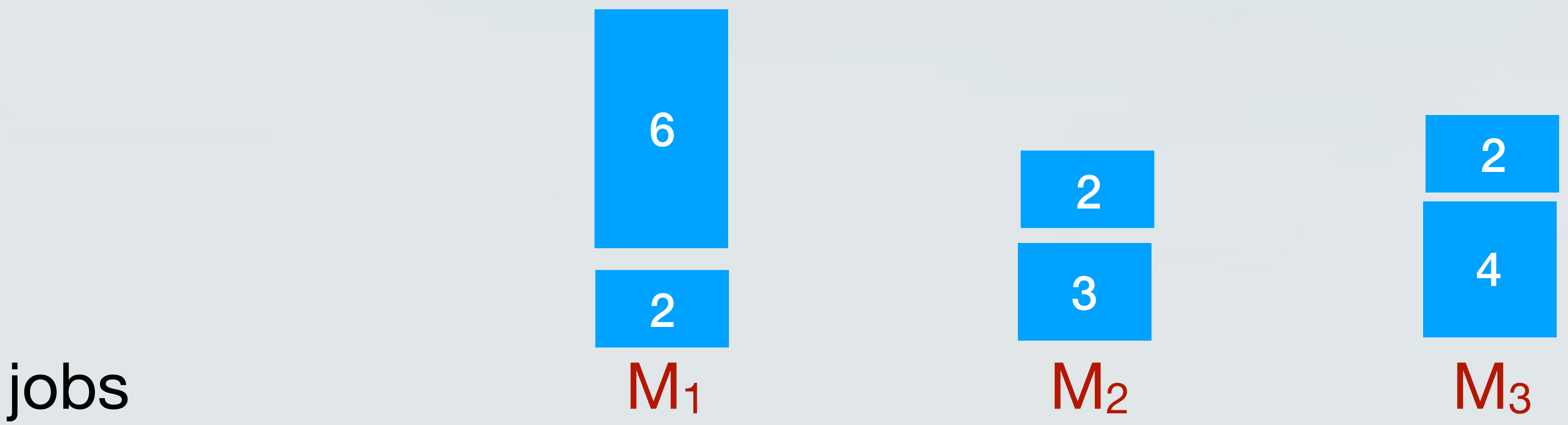
Example



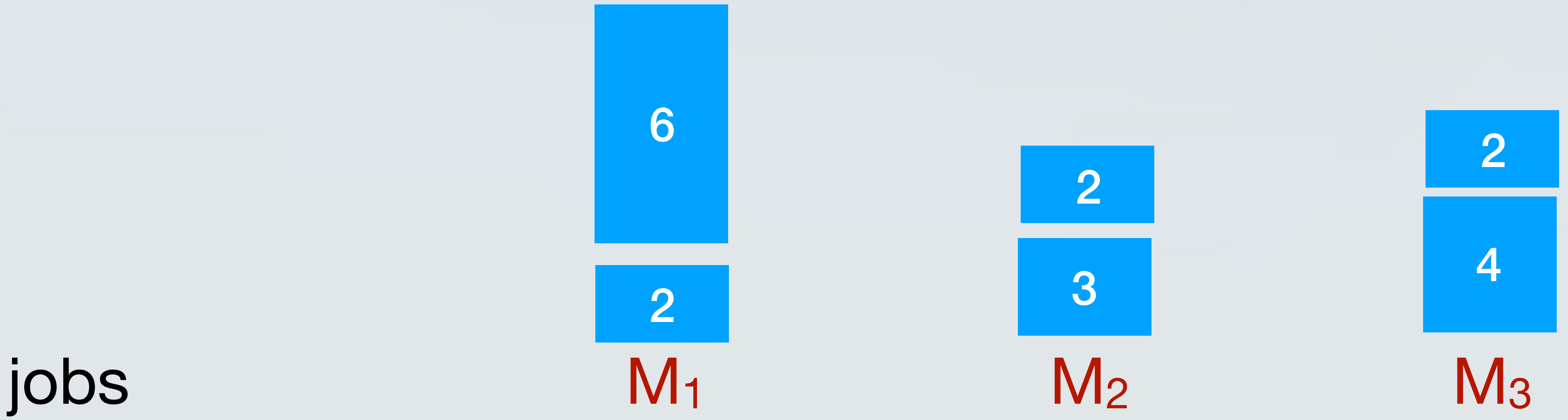
Example



Example



Example



makespan = 8

Online algorithms

- Let’s design an online algorithm for Load Balancing.
- Ideas?

Approximation Ratio

- Consider a minimisation problem P and an objective obj.
  - Here: Load Balancing on identical machines and makespan.
  - Consider an approximation algorithm A.
  - Consider an input x to the problem P.
  - Let obj(A(x)) be the value of the objective from the solution of A on x.
  - Let opt(x) be the minimum possible value of the objective on x.

Approximation ratio

- The approximation ratio of A is defined as

$$\max_x \text{obj}(A(x)) / \text{opt}(x)$$

  - i.e., the worst case ratio of the objective achieved by the algorithm over the optimal value of the objective, over all possible inputs to the problem.

Competitive Ratio

- The competitive ratio of algorithm A is defined as

$$\max_x \; obj(A(x)) \,/\, opt(x)$$

  - i.e., the worst case ratio of the objective achieved by the online algorithm over the optimal value of the objective, over all possible inputs to the problem.

Competitive Ratio vs Approximation Ratio

- Very similar notions.
- Difference:
  - Approximation ratio: The constraint of our algorithm is that it must run in polynomial time. If we didn't have a time constraint, we would obtain the optimal.
  - Competitive Ratio: The constraint of our algorithm is that it does not know the future part of the input. If we had access to the future part of the input, we would obtain the optimal.

Greedy algorithm for load balancing

- Pick any job.
- Assign it to the machine with the smallest load so far.
- Remove it from the pile of jobs.

Algorithm **Greedy-Balance**

Start with no jobs assigned
Set $T_i = 0$ and $A(i) = \varnothing$ for all machines M_i
For $j = 1, \ldots, n$
    Let M_i be the machine that achieves the minimum $\min_k T_k$
    Assign job j to machine M_i
    Set $A(i) = A(i) \cup \{ j \}$
    Set $T_i = T_i + t_j$
EndFor

Greedy algorithm for online load balancing

- Pick the job that arrives in the current time step.
- Assign it to the machine with the smallest load so far.
- Remove it from the pile of jobs.

Algorithm **Greedy-Balance**

Start with no jobs assigned
Set $T_i = 0$ and $A(i) = \varnothing$ for all machines M_i
For $j = 1, \ldots, n$
    Let M_i be the machine that achieves the minimum $\min_k T_k$
    Assign job j to machine M_i
    Set $A(i) = A(i) \cup \{ j \}$
    Set $T_i = T_i + t_j$
EndFor

Competitive ratio of Greedy

- What is the competitive ratio of the Greedy algorithm?

Competitive ratio of Greedy

- What is the competitive ratio of the Greedy algorithm?
    - We have already done the analysis for the approximation ratio!

Competitive ratio of Greedy

- What is the competitive ratio of the Greedy algorithm?
  - We have already done the analysis for the approximation ratio!
    - 2 (using a “generous” analysis)

Competitive ratio of Greedy

- What is the competitive ratio of the Greedy algorithm?
    - We have already done the analysis for the approximation ratio!
        - 2 (using a “generous” analysis)
        - 2 - 1/m (using tighter analysis).

The limits of online algorithms

- Lower bounds: We can show lower bounds on the competitive ratio of *any* online algorithm, using elementary arguments.

- This comes *in contrast to* approximation algorithms, where inapproximability results typically required advanced techniques.

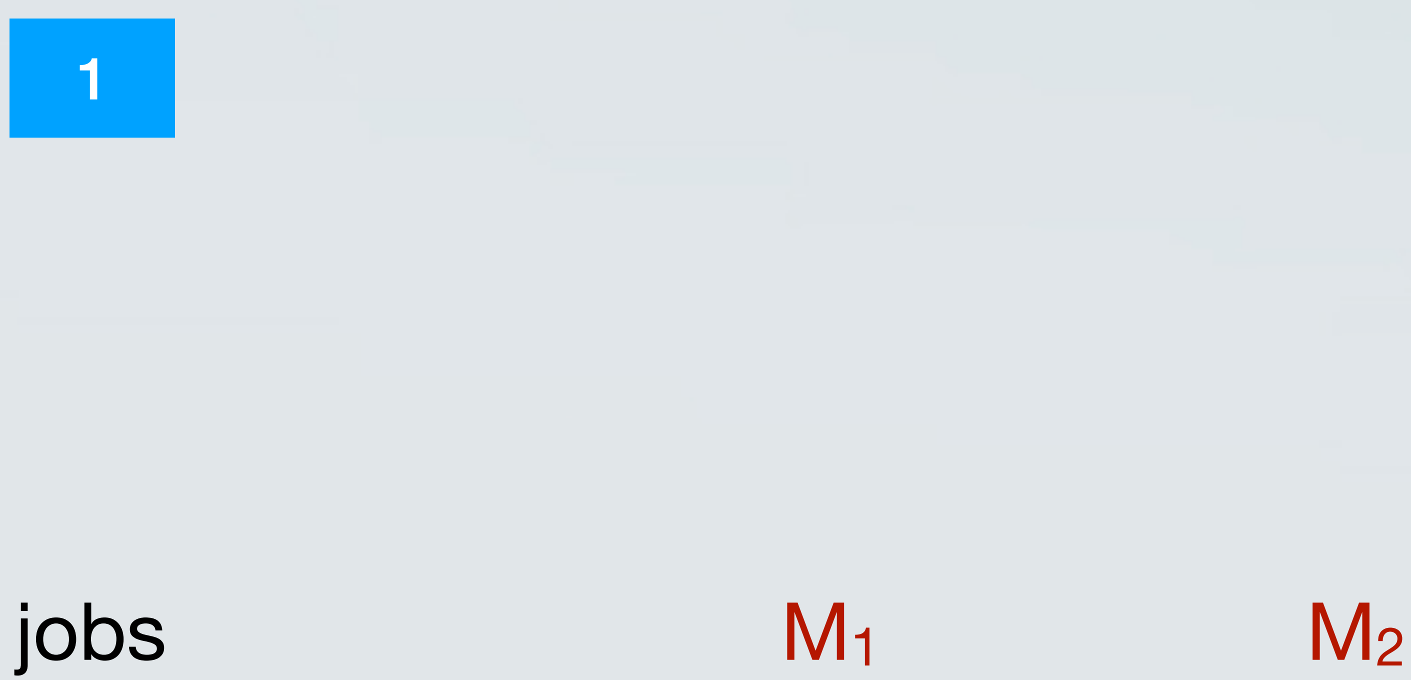
Terminology

- We will say that the input is given by an *adversary*, who wishes to minimise the competitive ratio of the algorithm.
- This is equivalent to considering the *worst possible case* for the input sequence.

Example: Load Balancing with $m=2$

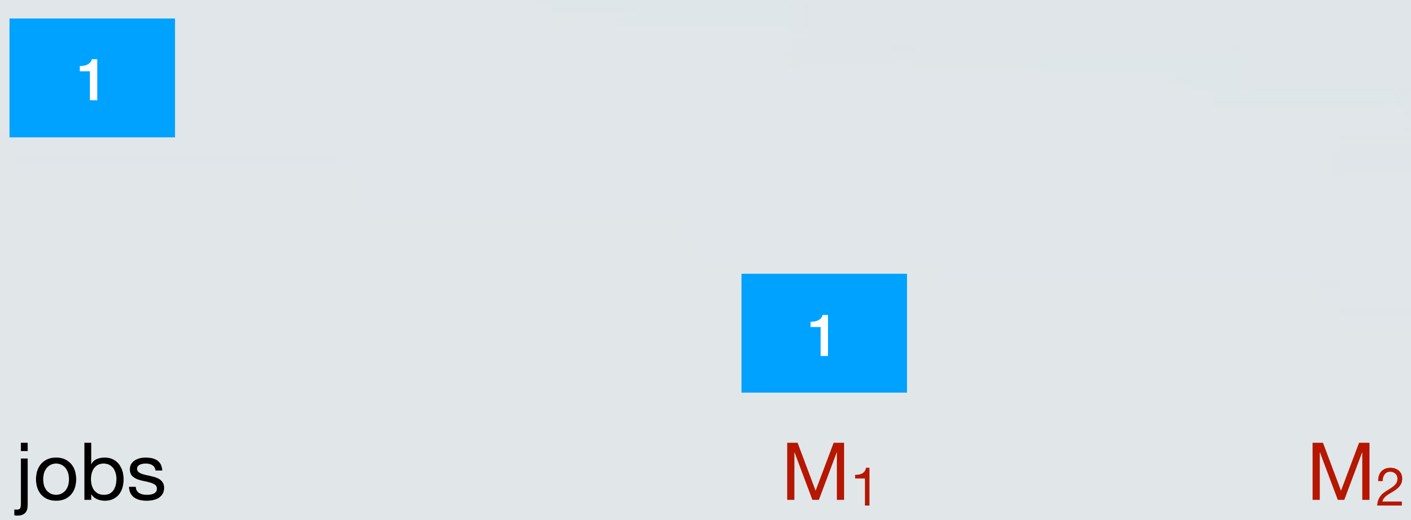
jobs M_1 M_2

Example: Load Balancing with m=2



Example: Load Balancing with m=2

Example: Load Balancing with m=2



Example: Load Balancing with $m=2$

Example: Load Balancing with m=2

Case 1: Both jobs go to M_1

Example: Load Balancing with $m=2$

Case 1: Both jobs go to M_1

Competitive ratio is 2.

Example: Load Balancing with m=2

jobs M_1 M_2

Example: Load Balancing with $m=2$

1

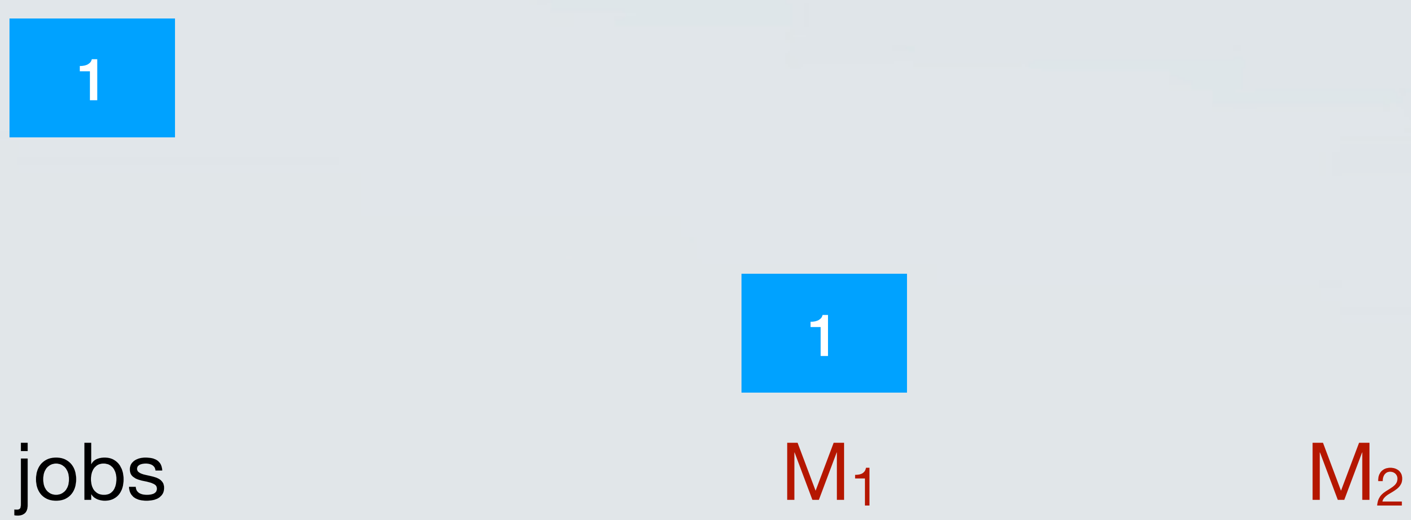
jobs

M_1

M_2

Example: Load Balancing with m=2

Example: Load Balancing with m=2



Example: Load Balancing with $m=2$

	1	1
jobs	M_1	M_2

Example: Load Balancing with m=2

Case 2: Each job goes to a different machine.

Example: Load Balancing with m=2

Case 2: Each job goes to a different machine.

The adversary introduces a new job with size 2.

Example: Load Balancing with $m=2$

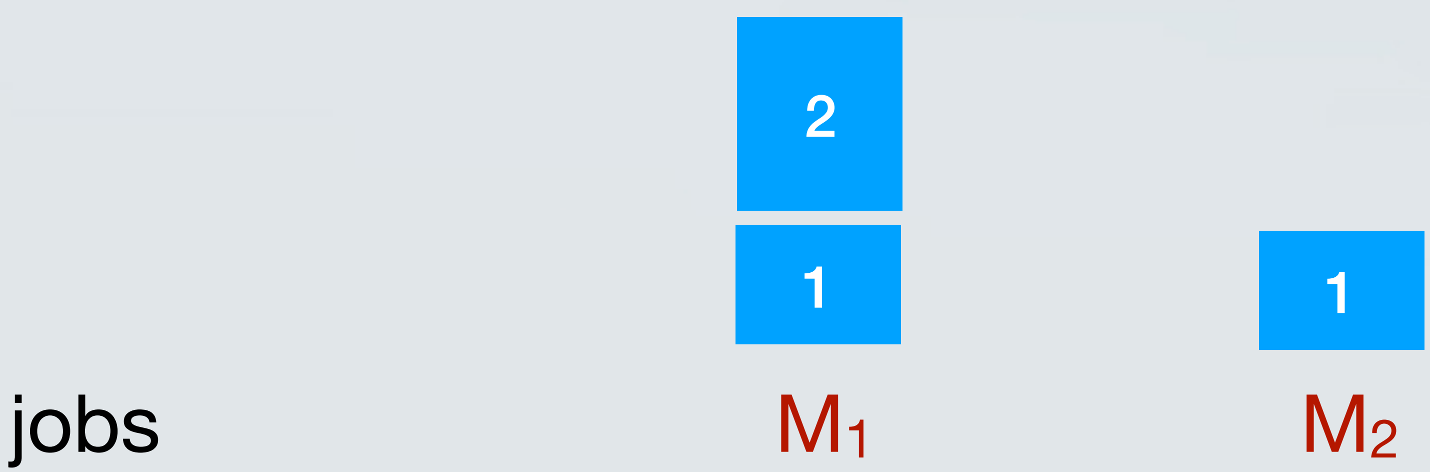
Case 2: Each job goes to a different machine.

The adversary introduces a new job with size 2.

Example: Load Balancing with m=2

Case 2: Each job goes to a different machine.

The adversary introduces a new job with size 2.

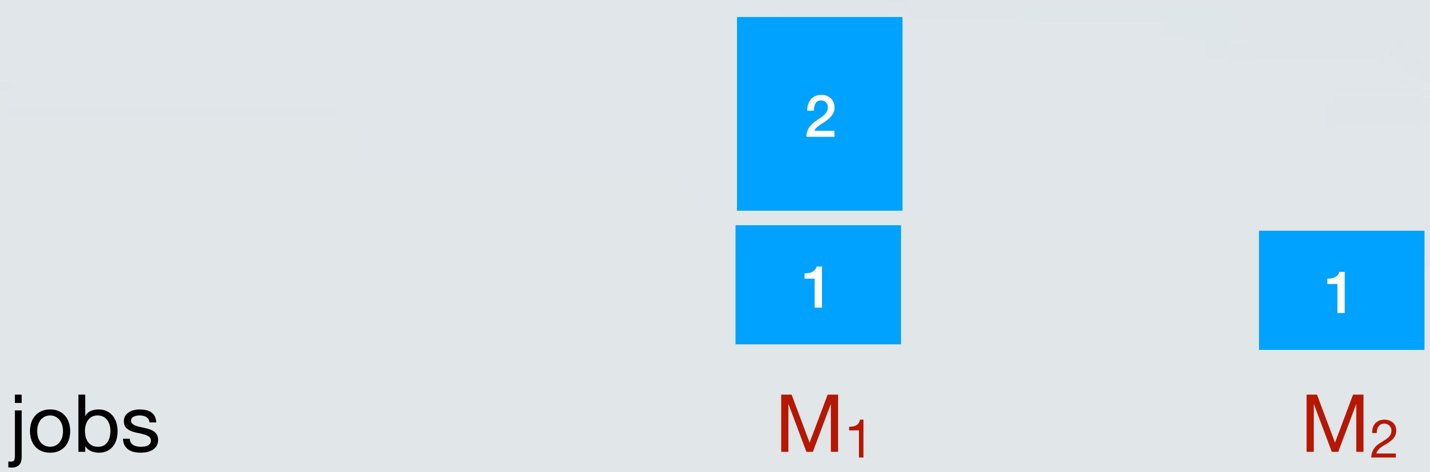


Example: Load Balancing with m=2

Case 2: Each job goes to a different machine.

The adversary introduces a new job with size 2.

Competitive ratio is 3/2.



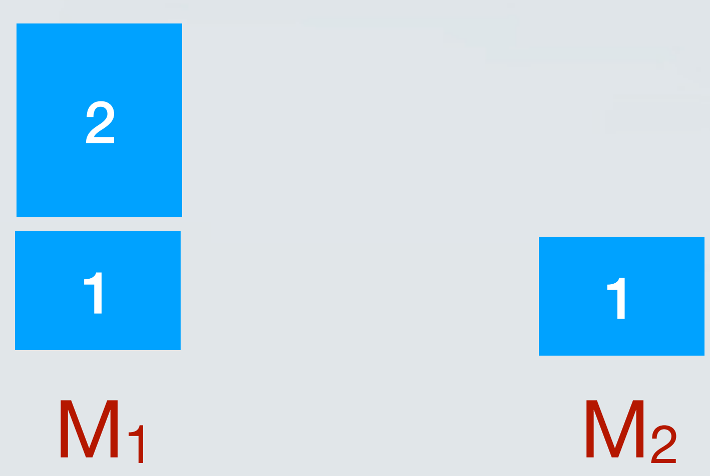
Example: Load Balancing with m=2

Case 2: Each job goes to a different machine.

The adversary introduces a new job with size 2.

Competitive ratio is 3/2.

jobs



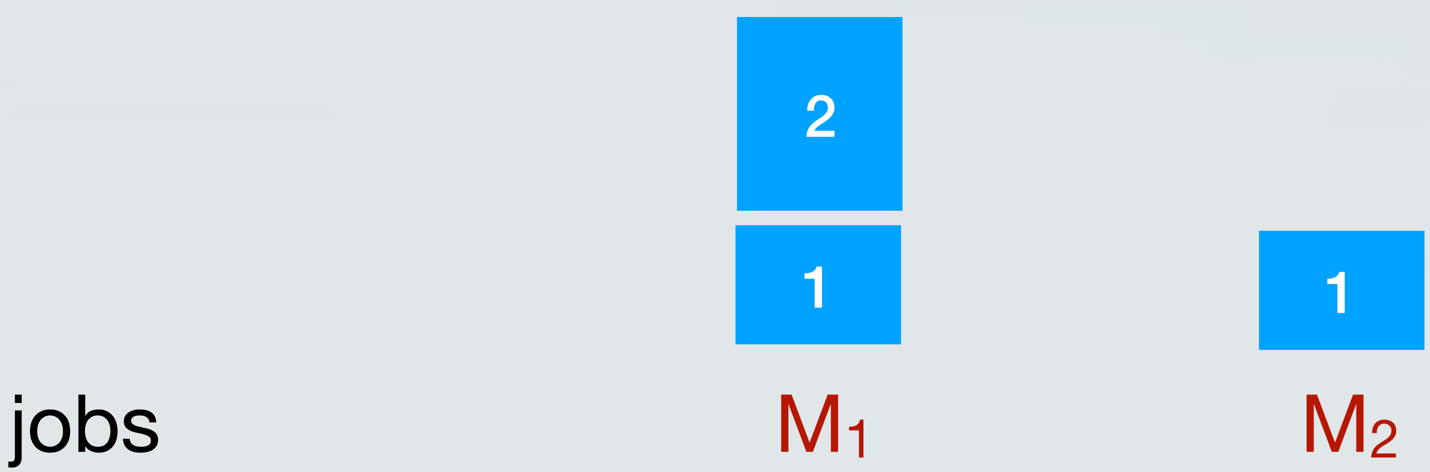
The greedy algorithm achieves 3/2.

Example: Load Balancing with m=2

Case 2: Each job goes to a different machine.

The adversary introduces a new job with size 2.

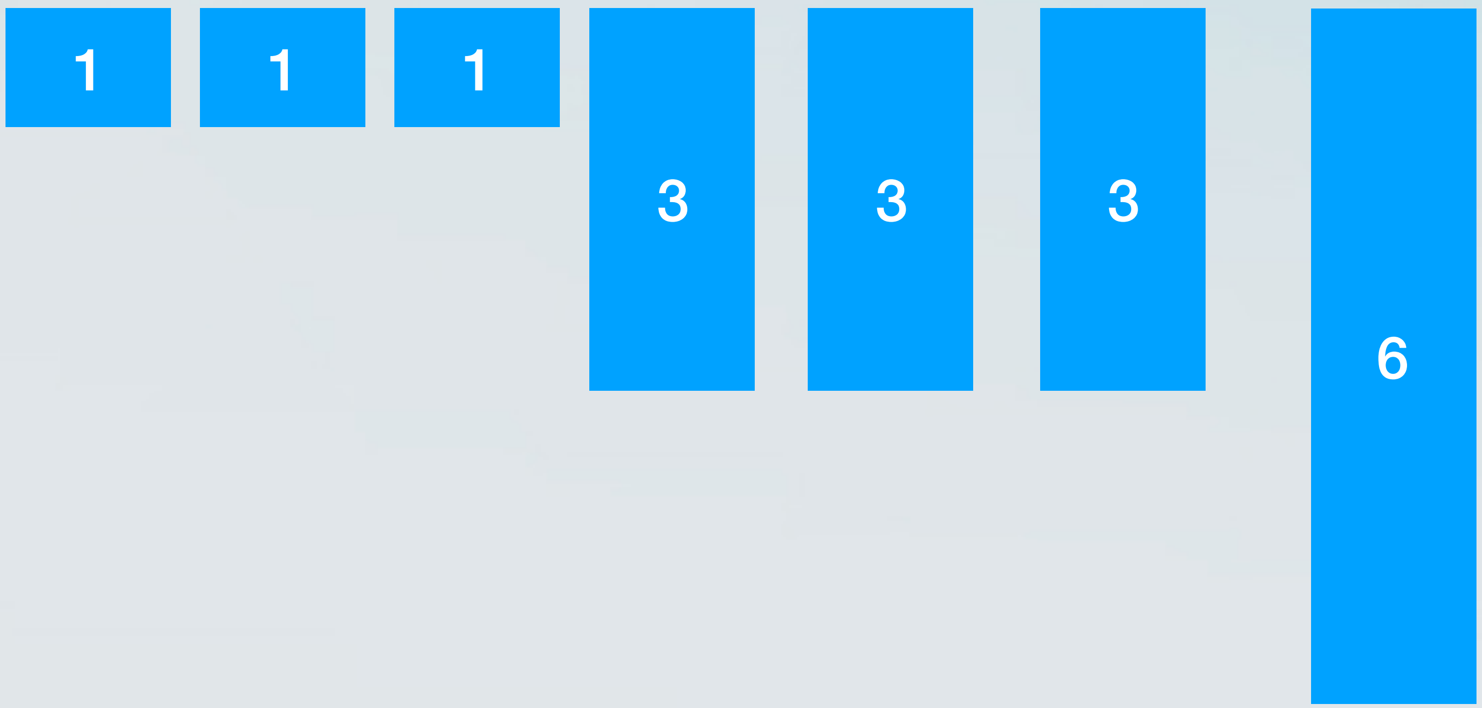
Competitive ratio is 3/2.



The greedy algorithm achieves 3/2.

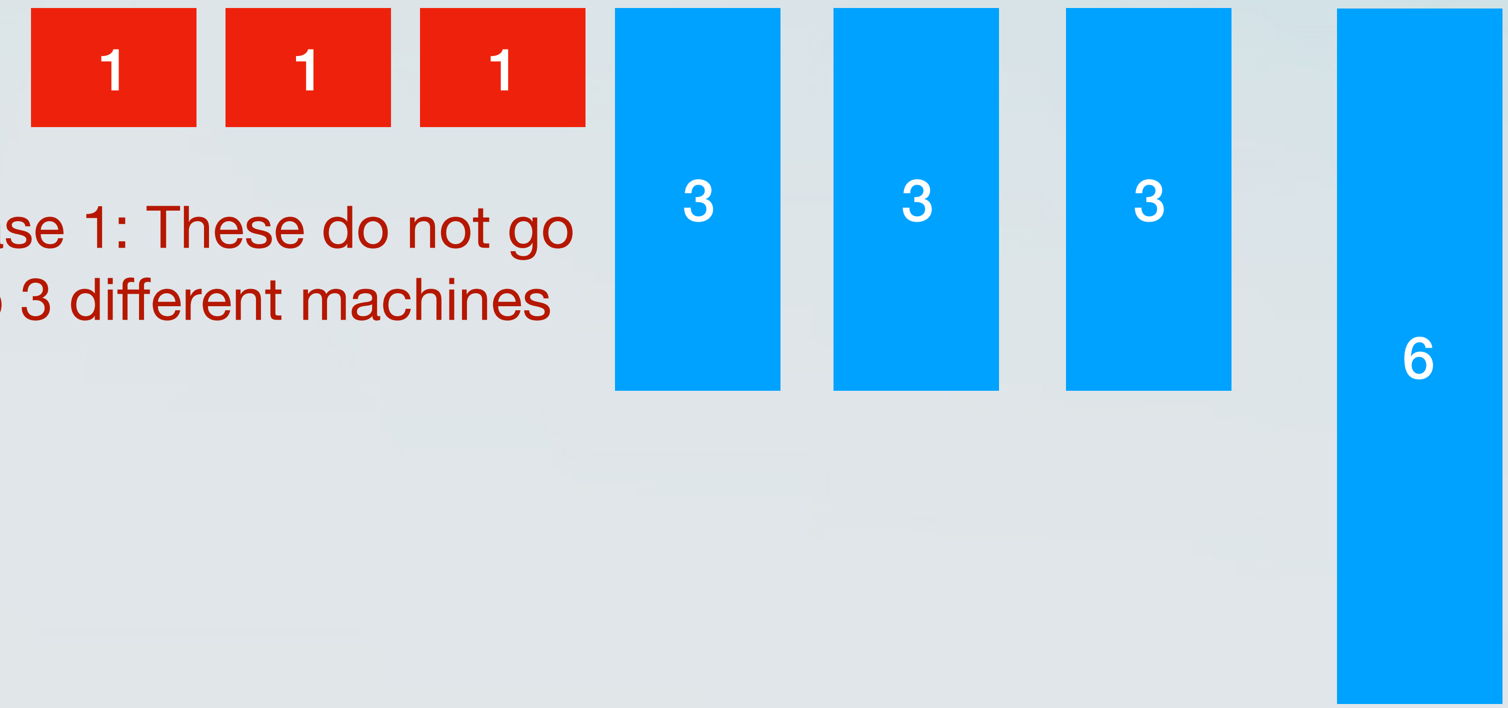
The greedy algorithm is the best possible for two machines.

Example: Load Balancing with m=3



jobs

Example: Load Balancing with m=3



Case 1: These do not go to 3 different machines

jobs

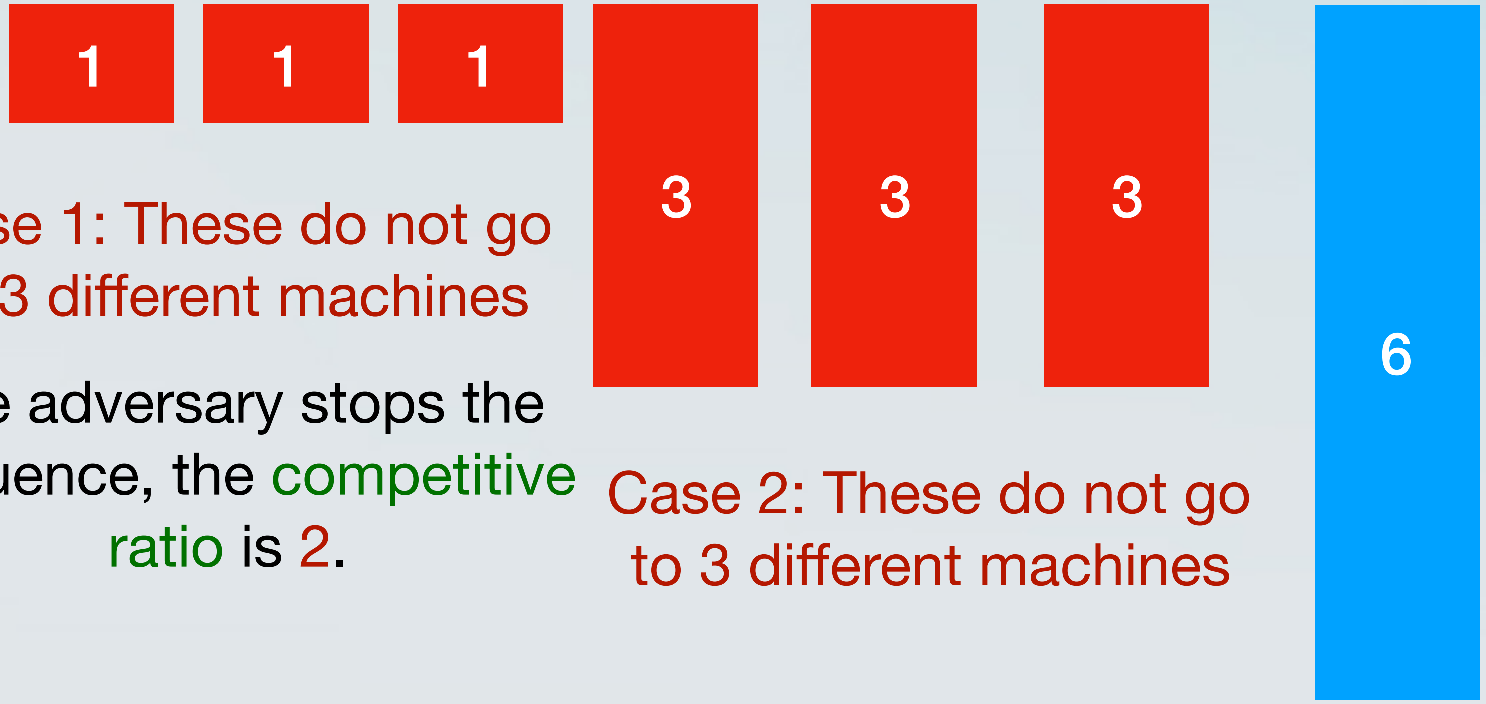
Example: Load Balancing with m=3

Case 1: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 2.

jobs

Example: Load Balancing with m=3



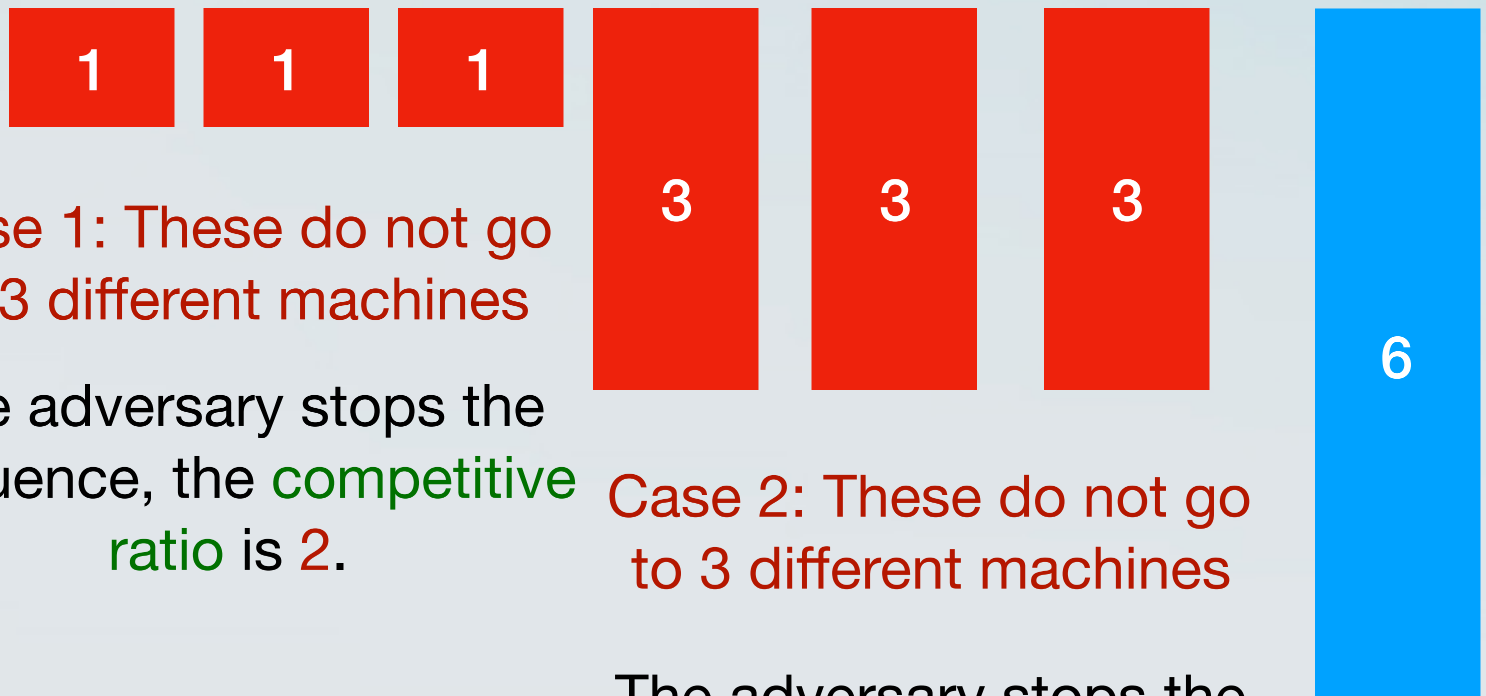
Case 1: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 2.

Case 2: These do not go to 3 different machines

jobs

Example: Load Balancing with m=3



jobs

Case 1: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 2.

Case 2: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 7/4.

Example: Load Balancing with m=3

1 1 1 3 3 3 6

jobs

Case 1: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 2.

Case 2: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 7/4.

Case 3: Every machine has load 4 before adding this

Example: Load Balancing with m=3

1 1 1

Case 1: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 2.

3 3 3

Case 2: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 7/4.

6

jobs

Case 3: Every machine has load 4 before adding this

After adding this, the maximum load is 10, but the optimal is 6. The competitive ratio is 5/3.

Example: Load Balancing with m=3

1 1 1 3 3 3 6 jobs

Case 1: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 2.

Case 2: These do not go to 3 different machines

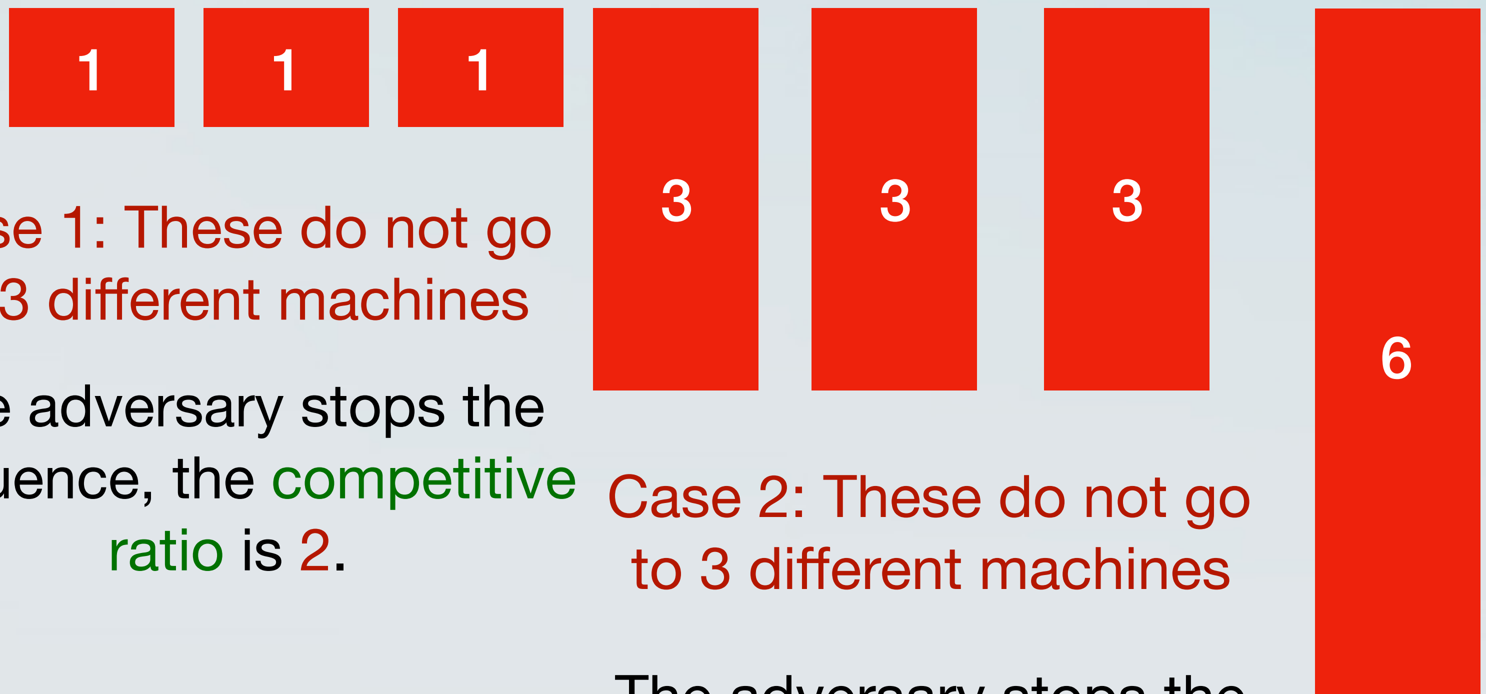
The adversary stops the sequence, the competitive ratio is 7/4.

Case 3: Every machine has load 4 before adding this

After adding this, the maximum load is 10, but the optimal is 6. The competitive ratio is 5/3.

The greedy algorithm achieves 5/3.

Example: Load Balancing with m=3



jobs

Case 1: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 2.

Case 2: These do not go to 3 different machines

The adversary stops the sequence, the competitive ratio is 7/4.

Case 3: Every machine has load 4 before adding this

After adding this, the maximum load is 10, but the optimal is 6. The competitive ratio is 5/3.

The greedy algorithm achieves 5/3.

The greedy algorithm is the best possible for three machines.

Example: Load Balancing with $m \geq 4$

- It can be proven using similar arguments that for $m \geq 4$ machines, the competitive ratio of *any* online algorithm is at least 1.70.
- The Greedy Algorithm achieves 1.75 for $m = 4$, so it is *not* the best possible for this case.

Better Algorithms

Better Algorithms

- We saw several better algorithms for Load Balancing.

Better Algorithms

- We saw several better algorithms for Load Balancing.
  - The problem even has an FPTAS.

Better Algorithms

- We saw several better algorithms for Load Balancing.
  - The problem even has an FPTAS.
  - Could we use those instead of Greedy?

Better Algorithms

- We saw several better algorithms for Load Balancing.
  - The problem even has an FPTAS.
  - Could we use those instead of Greedy?
  - You might be tempted to think so, but not really!

Better Algorithms

- We saw several better algorithms for Load Balancing.
  - The problem even has an FPTAS.
  - Could we use those instead of Greedy?
  - You might be tempted to think so, but not really!
- Greedy approximation algorithms can *sometimes* be used as online algorithms, but in general

  approximation algorithms ≠ online algorithms

Better Algorithms

- It is possible to design better online algorithms for the scheduling problem.

Better Algorithms

- It is possible to design better online algorithms for the scheduling problem.
- For example, for $m = 4$, there is an algorithm with competitive ratio at 1.733.

Better Algorithms

- It is possible to design better online algorithms for the scheduling problem.
- For example, for $m = 4$, there is an algorithm with competitive ratio at 1.733.
- Lower bound: For $m = 4$, no online algorithm has competitive ratio better than 1.732.

Better Algorithms

- It is possible to design better online algorithms for the scheduling problem.
- For example, for $m = 4$, there is an algorithm with competitive ratio at 1.733.
- Lower bound: For $m = 4$, no online algorithm has competitive ratio better than 1.732.
- For *general* m, the best possible competitive ratio is between 1.88 and 1.92.

Better Algorithms

- It is possible to design better online algorithms for the scheduling problem.
- For example, for $m = 4$, there is an algorithm with competitive ratio at 1.733.
- Lower bound: For $m = 4$, no online algorithm has competitive ratio better than 1.732.
- For *general m*, the best possible competitive ratio is between 1.88 and 1.92.
- Idea: The Tetris principle - *maintain imbalance*.

Paging

Paging

- We have two types of memory, a *fast memory* (*cache*) and a *slow memory*.

Paging

- We have two types of memory, a *fast memory* (*cache*) and a *slow memory*.
- The cache has capacity k pages, the slow memory has capacity n pages.

Paging

- We have two types of memory, a *fast memory* (*cache*) and a *slow memory*.
- The cache has capacity k pages, the slow memory has capacity n pages.
- We have a sequence of *page requests*.

Paging

- We have two types of memory, a *fast memory* (*cache*) and a *slow memory*.
- The cache has capacity k pages, the slow memory has capacity n pages.
- We have a sequence of *page requests*.
- If the page is in the cache, the algorithm returns it at no cost.

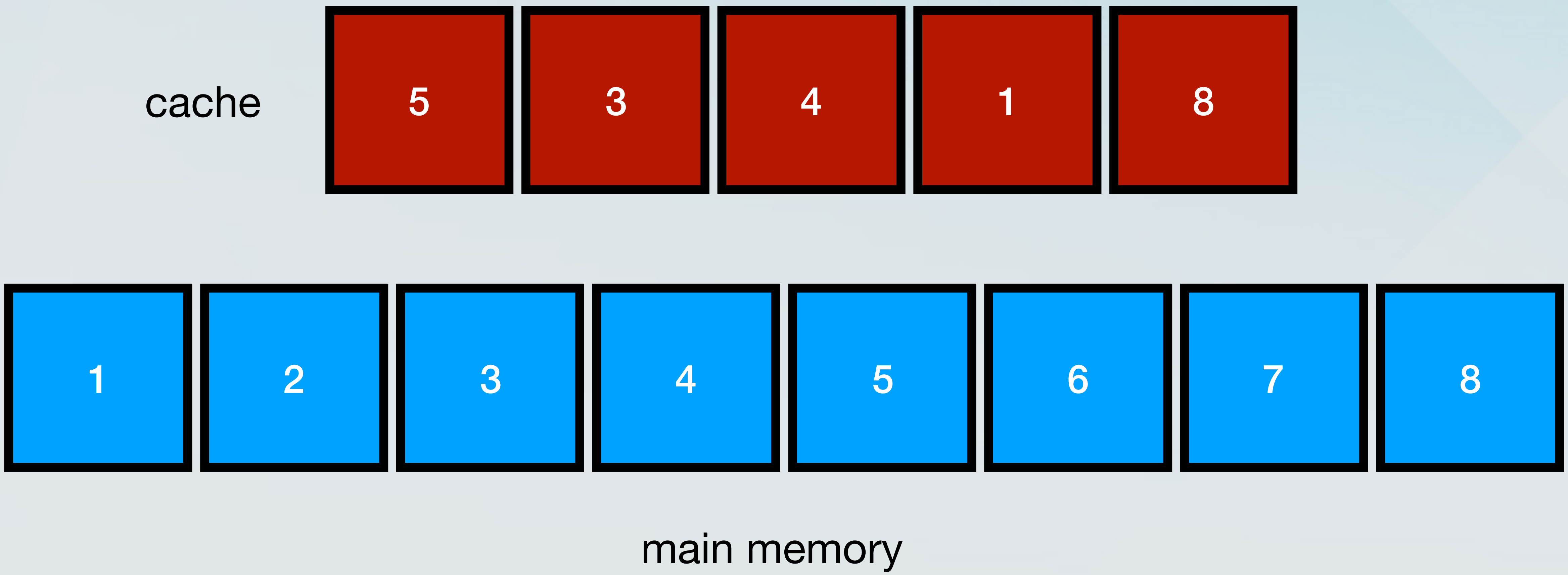
Paging

- We have two types of memory, a *fast memory* (*cache*) and a *slow memory*.
- The cache has capacity k pages, the slow memory has capacity n pages.
- We have a sequence of *page requests*.
- If the page is in the cache, the algorithm returns it at no cost.
- If the page is not in the cache, the algorithm "*faults*" and has to bring it from the cache, paying a *cost* of 1.

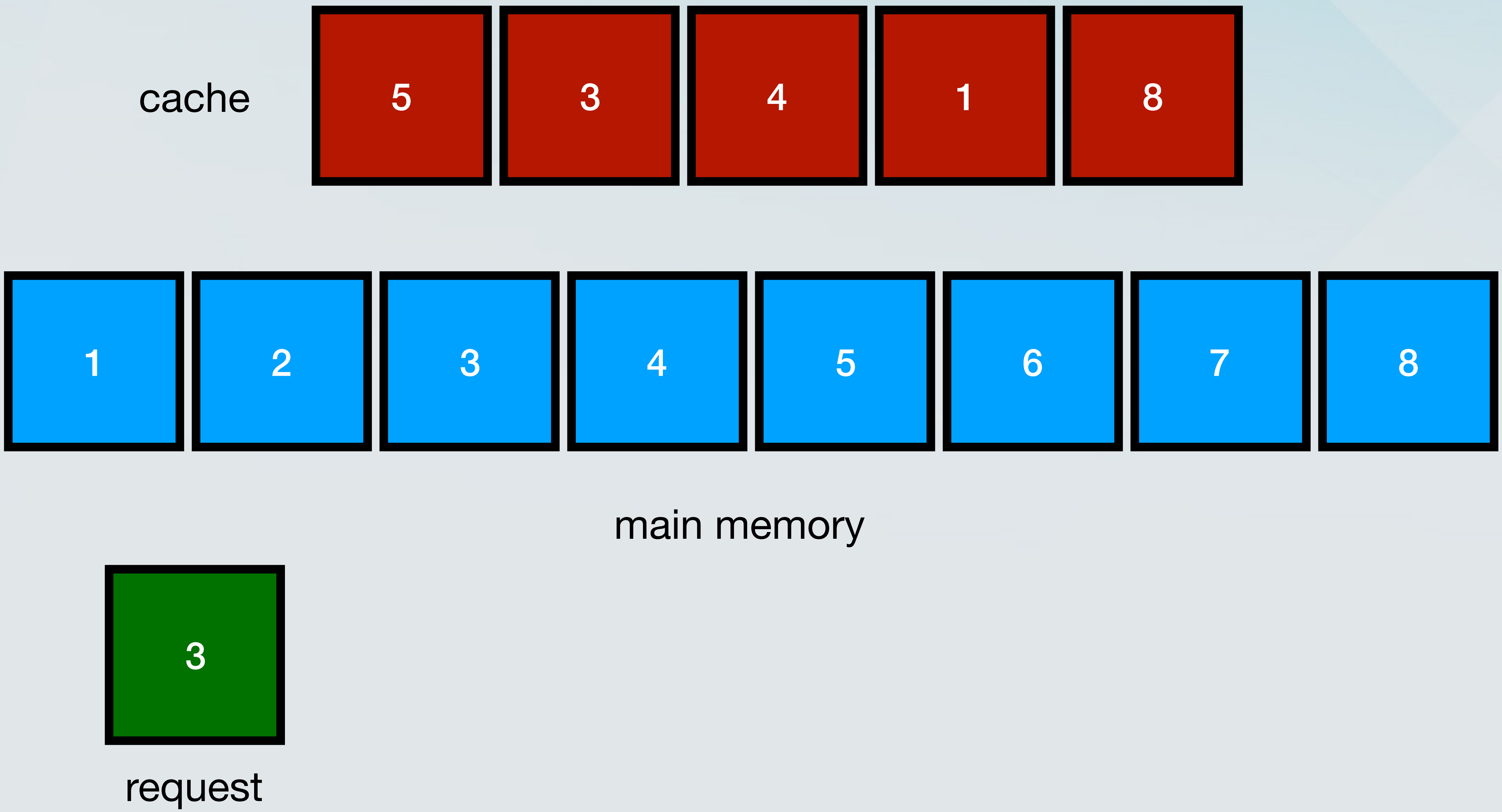
Paging

- We have two types of memory, a *fast memory* (*cache*) and a *slow memory*.
- The cache has capacity k pages, the slow memory has capacity n pages.
- We have a sequence of *page requests*.
- If the page is in the cache, the algorithm returns it at no cost.
- If the page is not in the cache, the algorithm "*faults*" and has to bring it from the cache, paying a *cost* of 1.
- The algorithm must also choose a page in the cache to *replace* with the page brought from the slow memory.

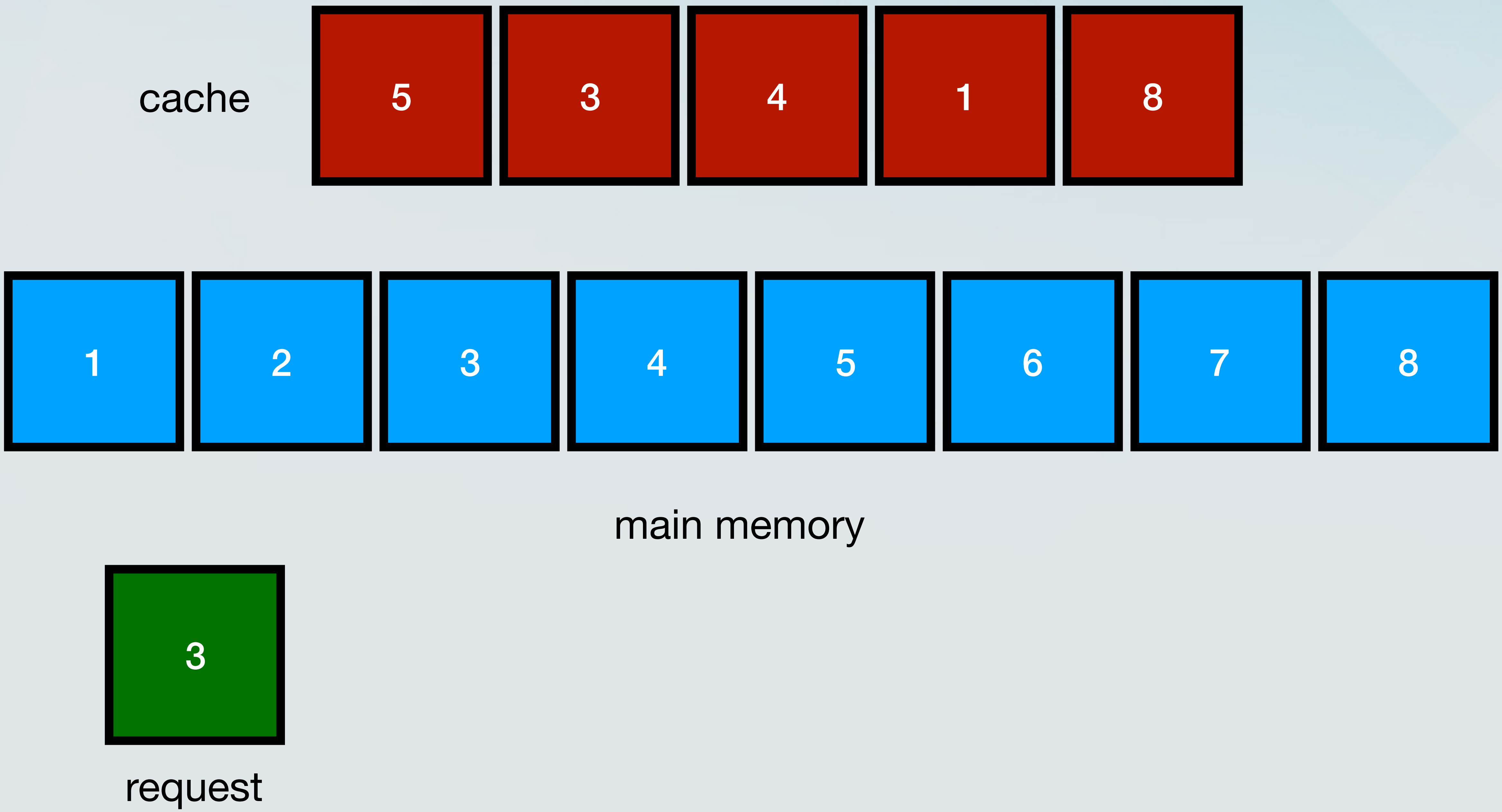
Example



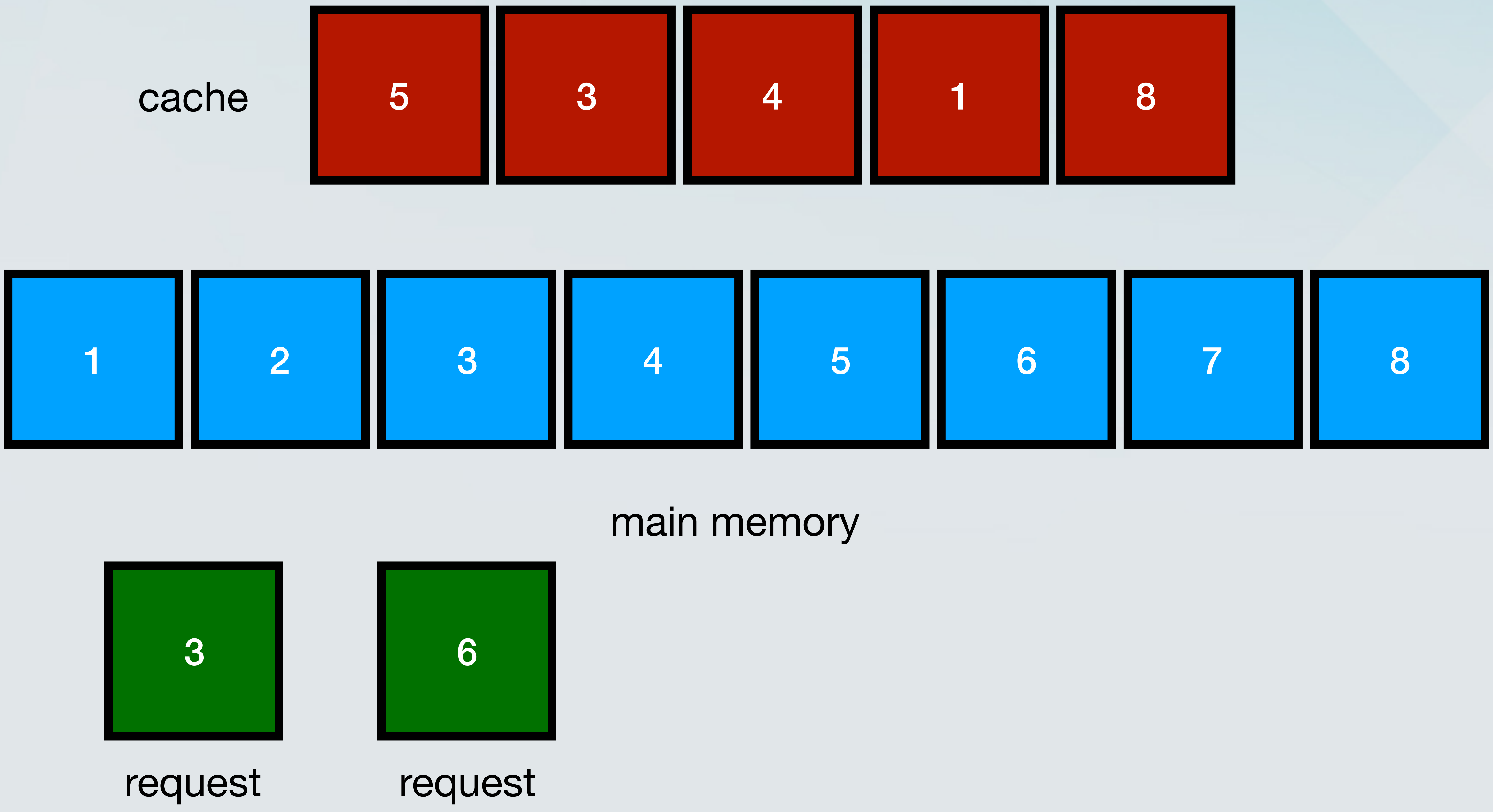
Example



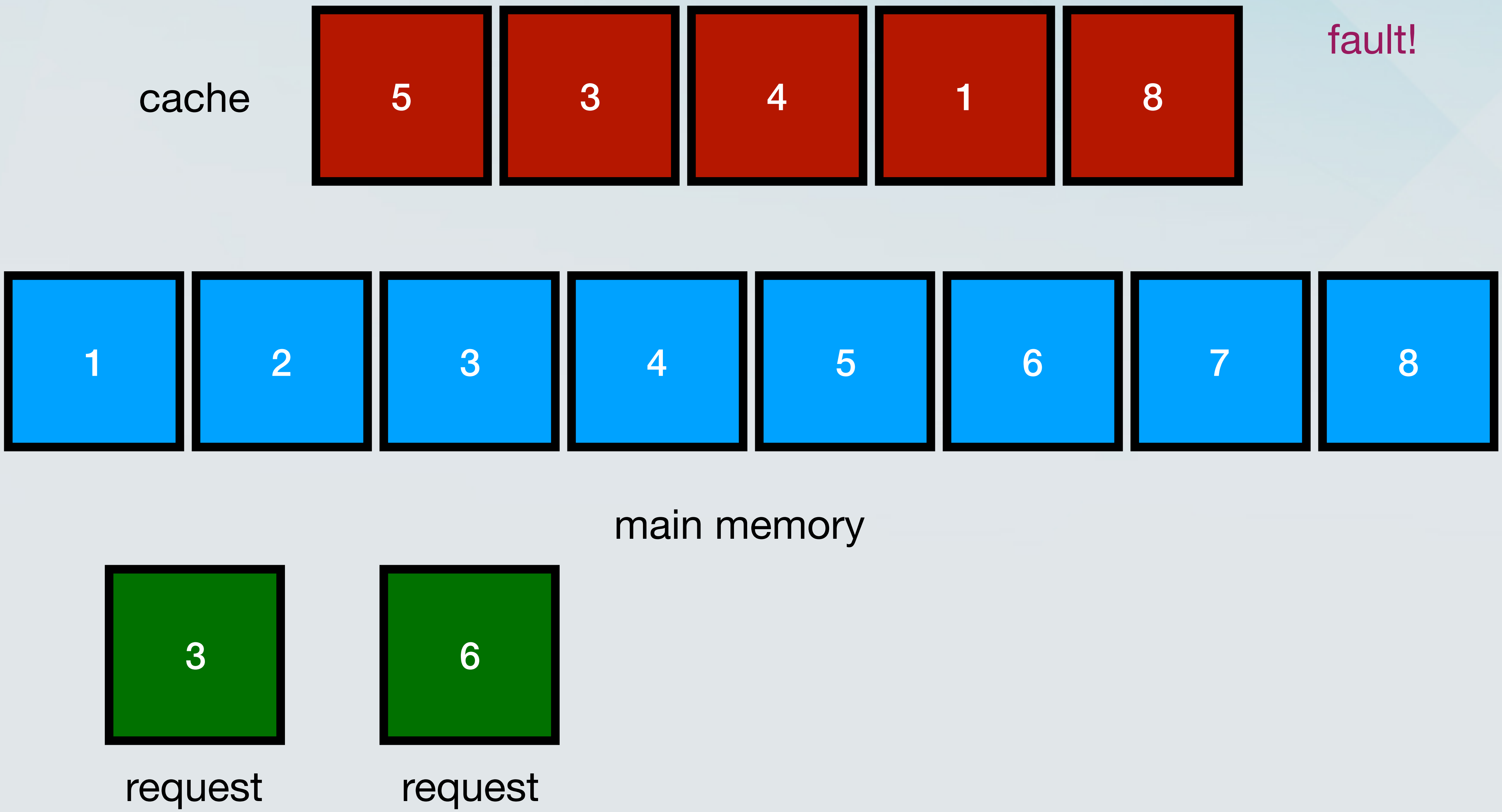
Example



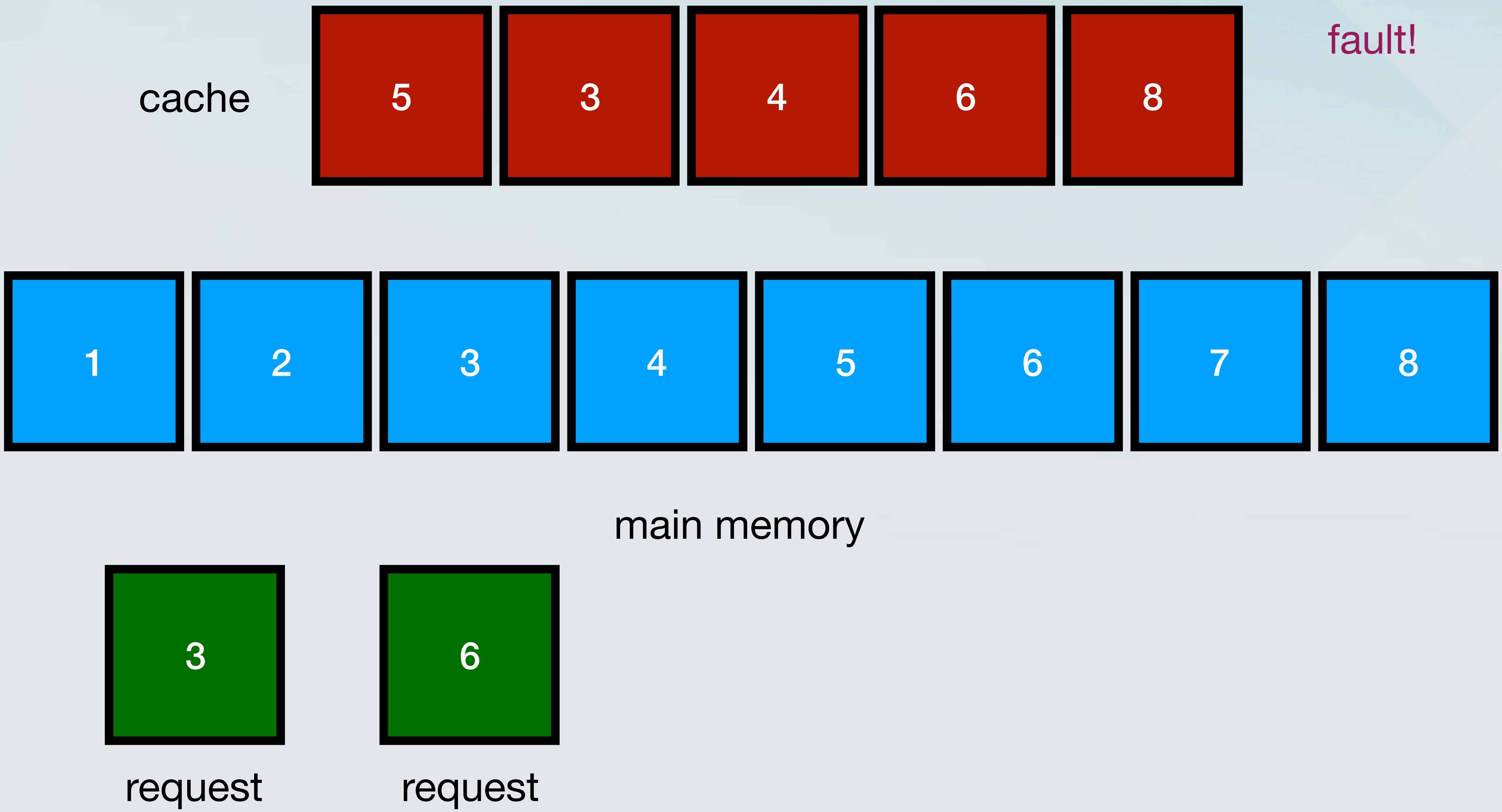
Example



Example



Example



Costs

Costs

- The cost of an algorithm is the number of “*faults*” that it makes.

Costs

- The cost of an algorithm is the number of “*faults*” that it makes.
- How does the cost of an online algorithm compare to the cost of the optimal offline algorithm?

Costs

- The cost of an algorithm is the number of "*faults*" that it makes.
- How does the cost of an online algorithm compare to the cost of the optimal offline algorithm?
- The online algorithm makes x "*faults*".

Costs

- The cost of an algorithm is the number of “*faults*” that it makes.

- How does the cost of an online algorithm compare to the cost of the optimal offline algorithm?

- The online algorithm makes x “*faults*”.

- The offline optimal makes $y \leq x$ “*faults*”.

Costs

- The cost of an algorithm is the number of "*faults*" that it makes.

- How does the cost of an online algorithm compare to the cost of the optimal offline algorithm?

- The online algorithm makes x "*faults*".

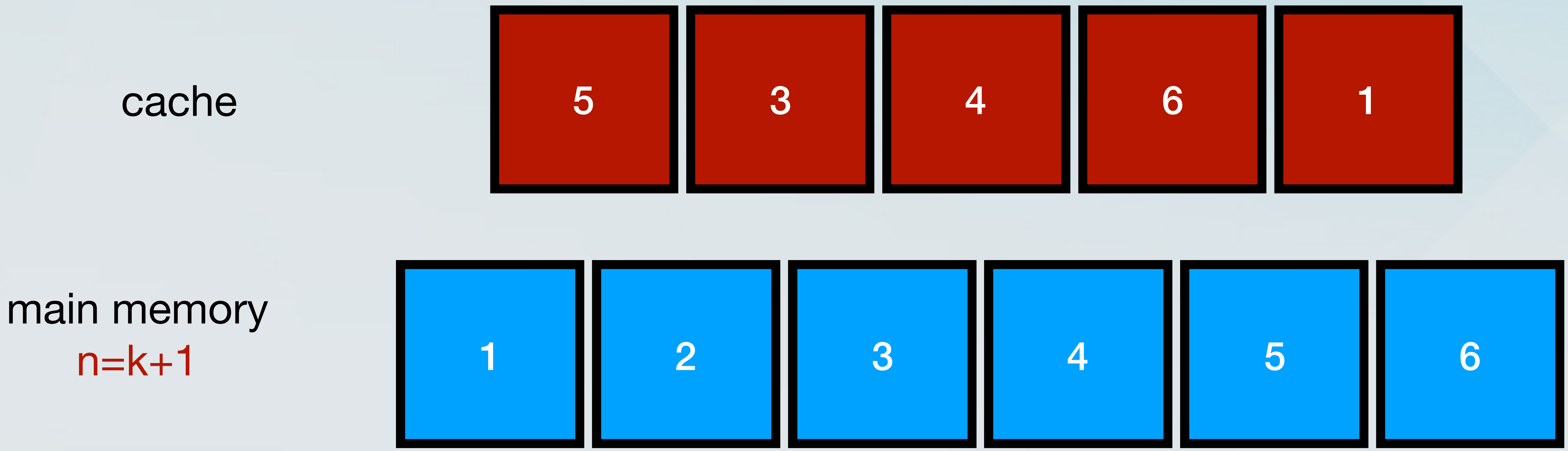
- The offline optimal makes $y \leq x$ "*faults*".

- We are interested in x/y.

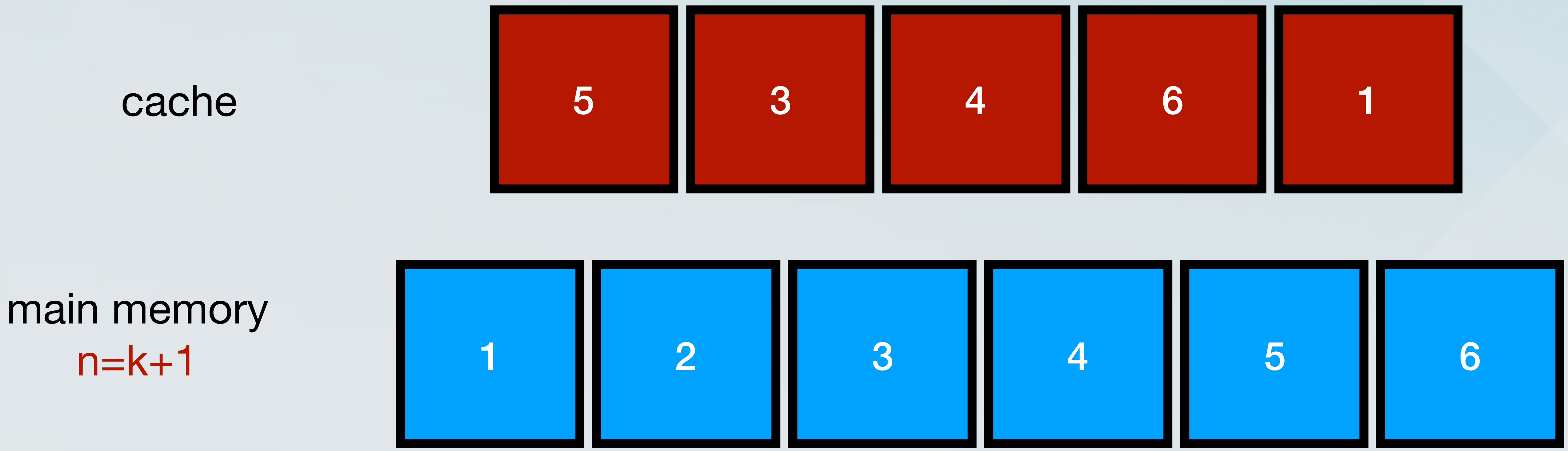
Lower bound on Paging algorithms

- Theorem: The competitive ratio of *any* online algorithm for paging is at least k.

Lower Bound

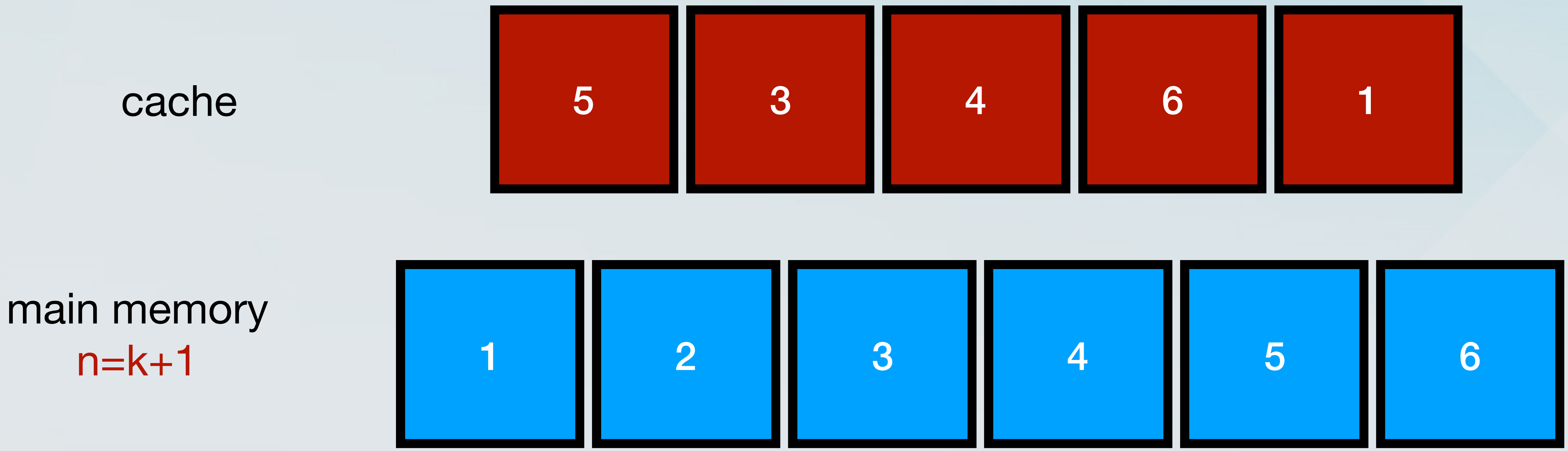


Lower Bound

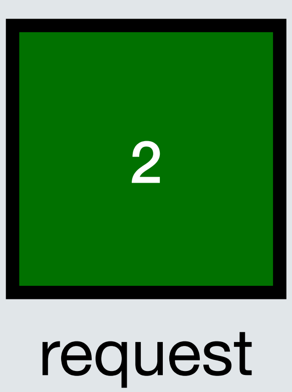


Adversary: Always ask for the page missing from the cache for the algorithm.

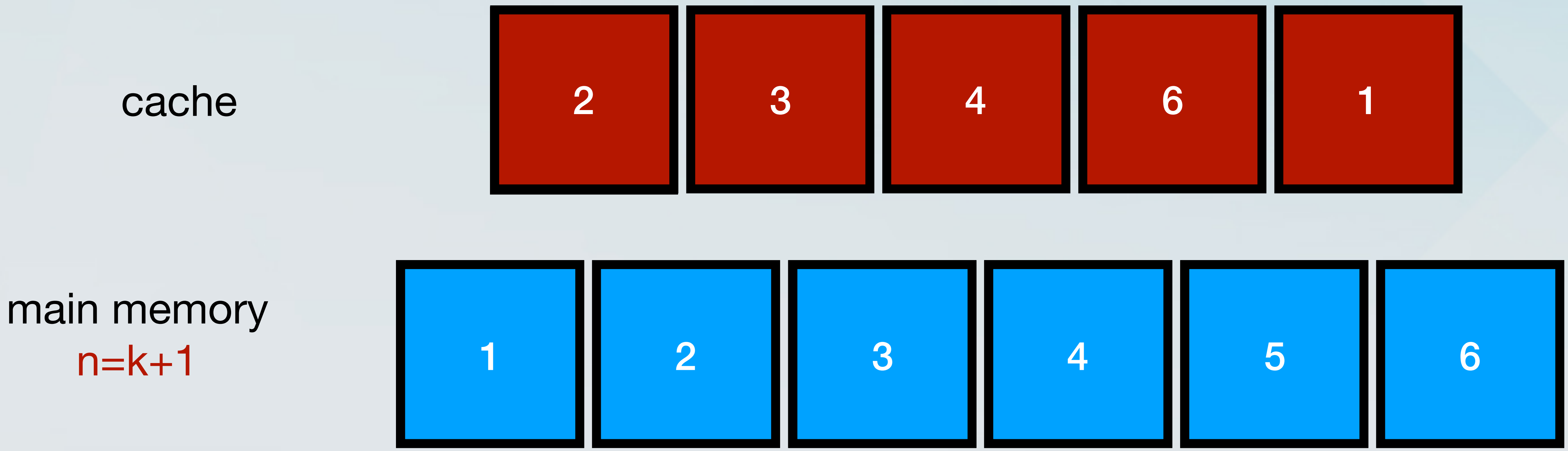
Lower Bound



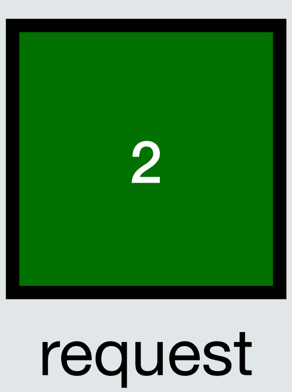
Adversary: Always ask for the page missing from the cache for the algorithm.



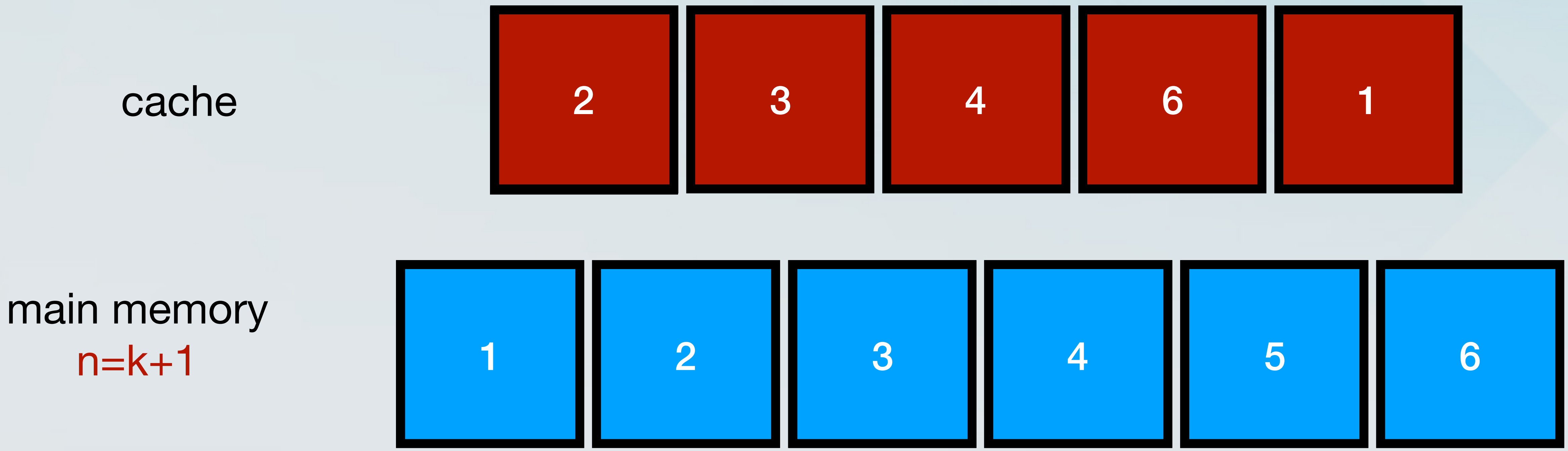
Lower Bound



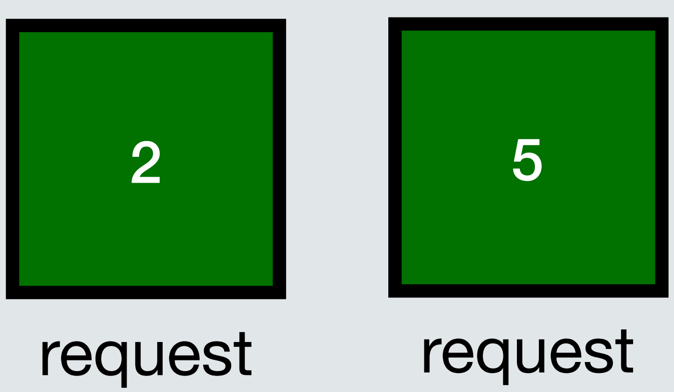
Adversary: Always ask for the page missing from the cache for the algorithm.



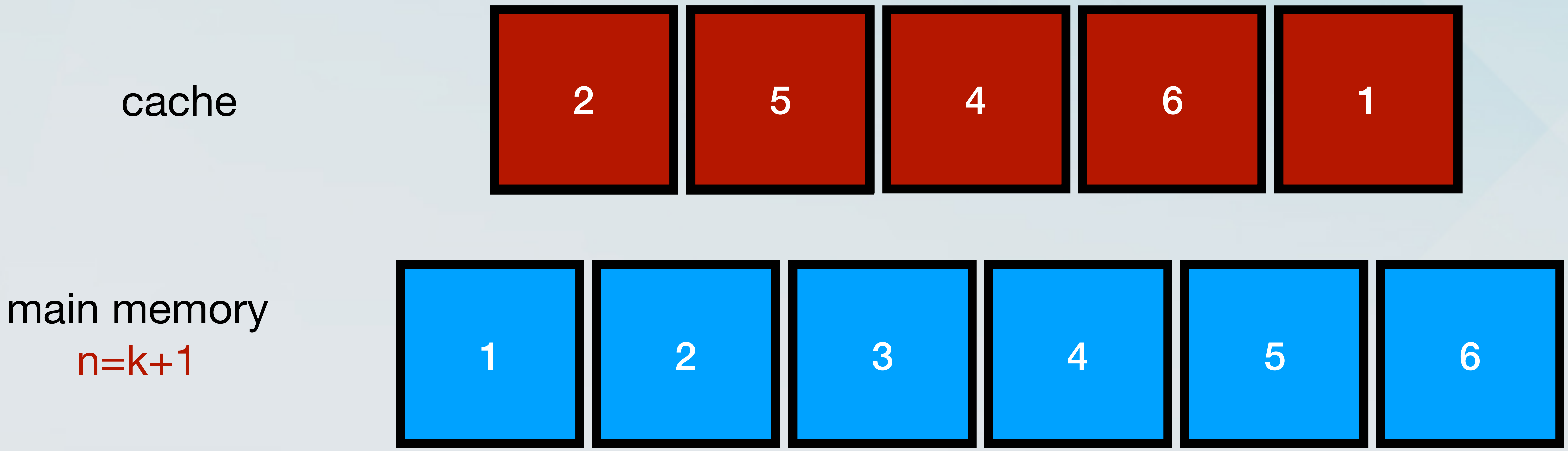
Lower Bound



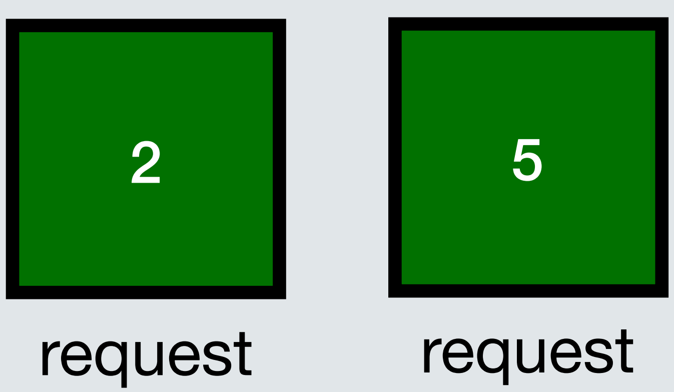
Adversary: Always ask for the page missing from the cache for the algorithm.



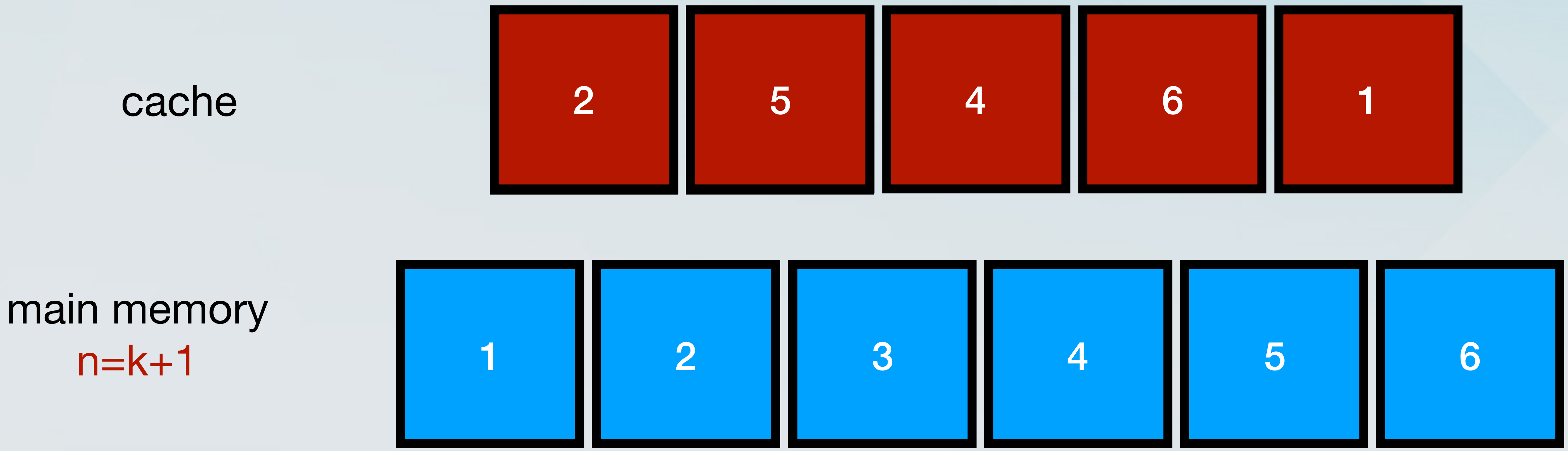
Lower Bound



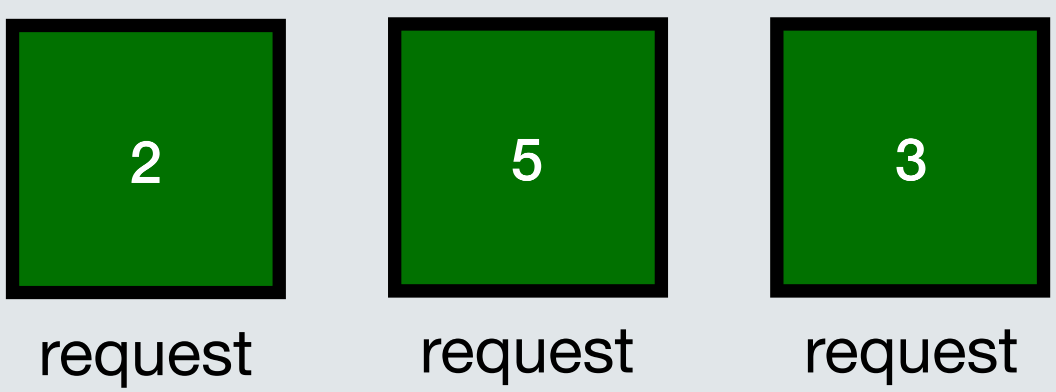
Adversary: Always ask for the page missing from the cache for the algorithm.



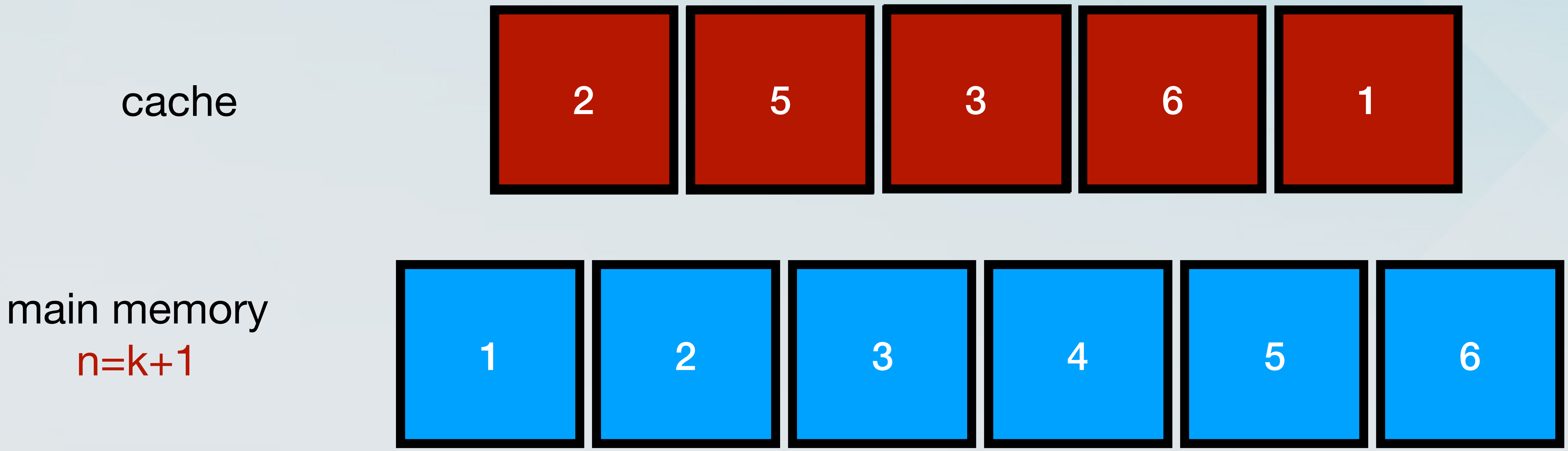
Lower Bound



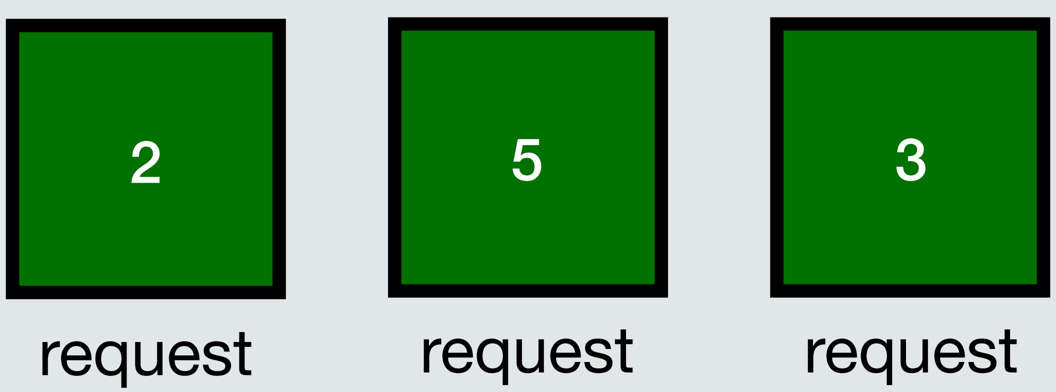
Adversary: Always ask for the page missing from the cache for the algorithm.



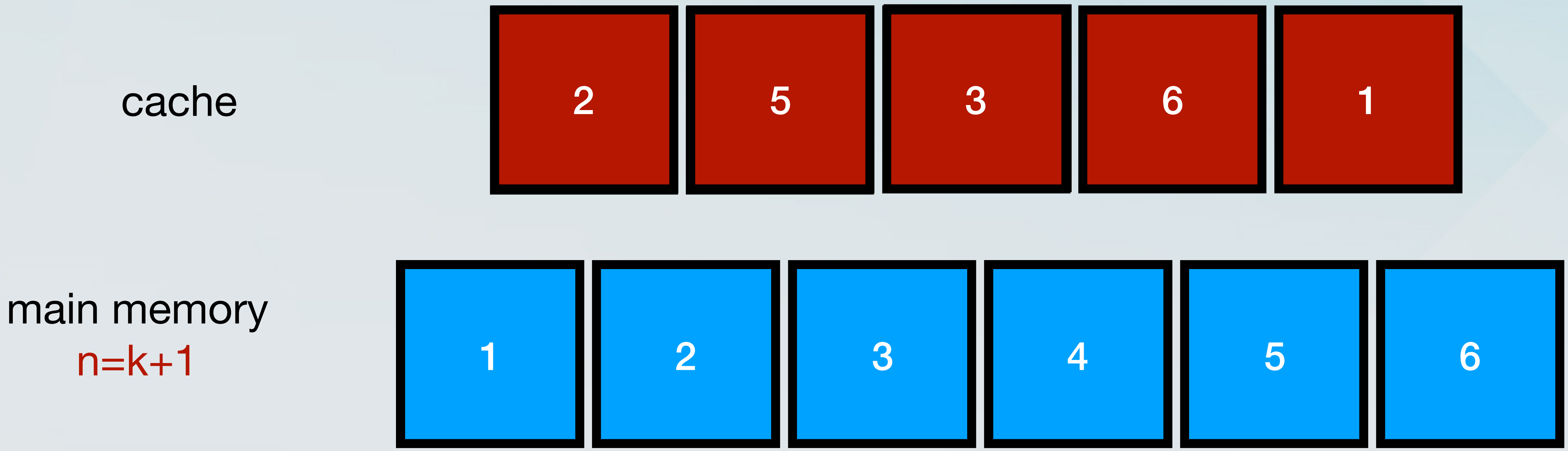
Lower Bound



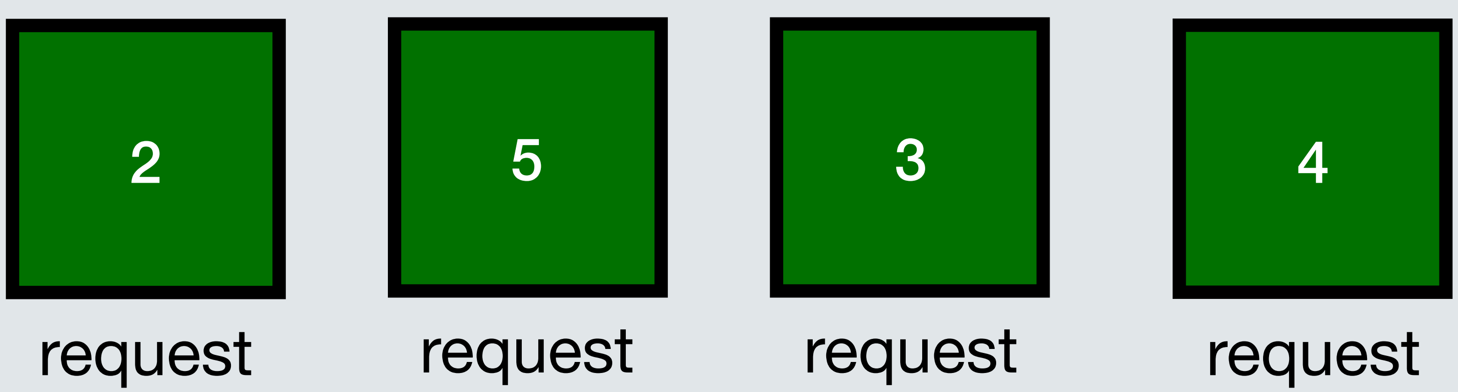
Adversary: Always ask for the page missing from the cache for the algorithm.



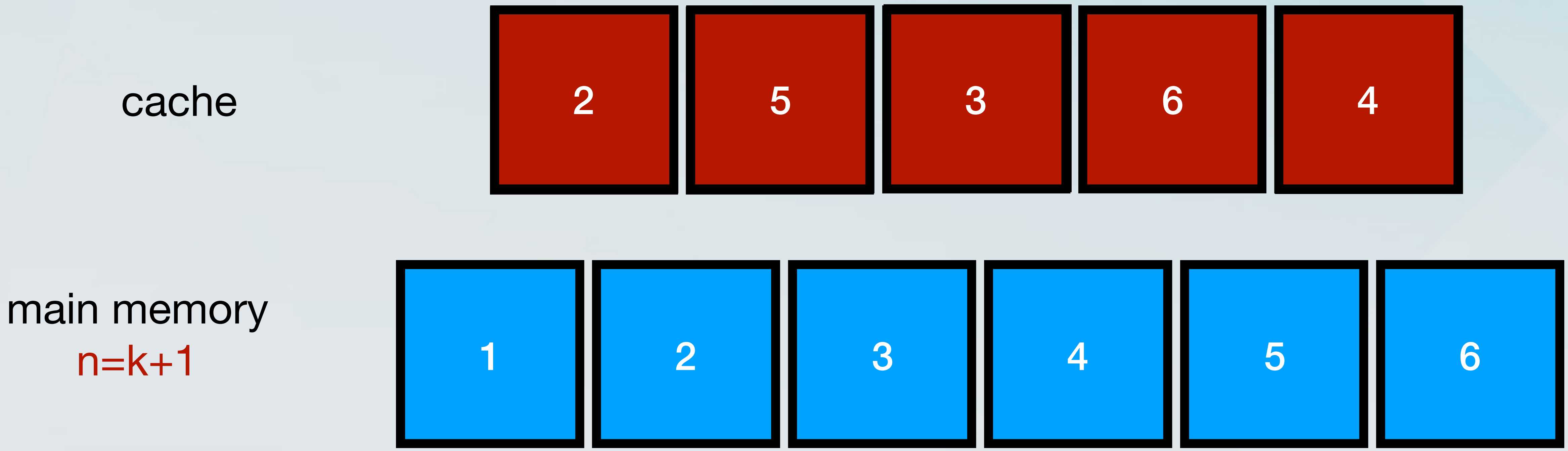
Lower Bound



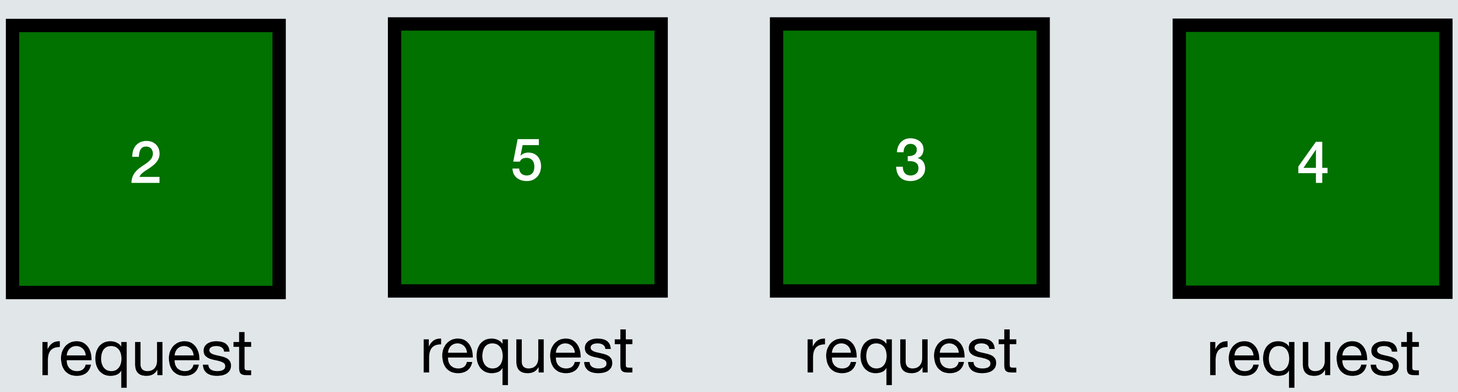
Adversary: Always ask for the page missing from the cache for the algorithm.



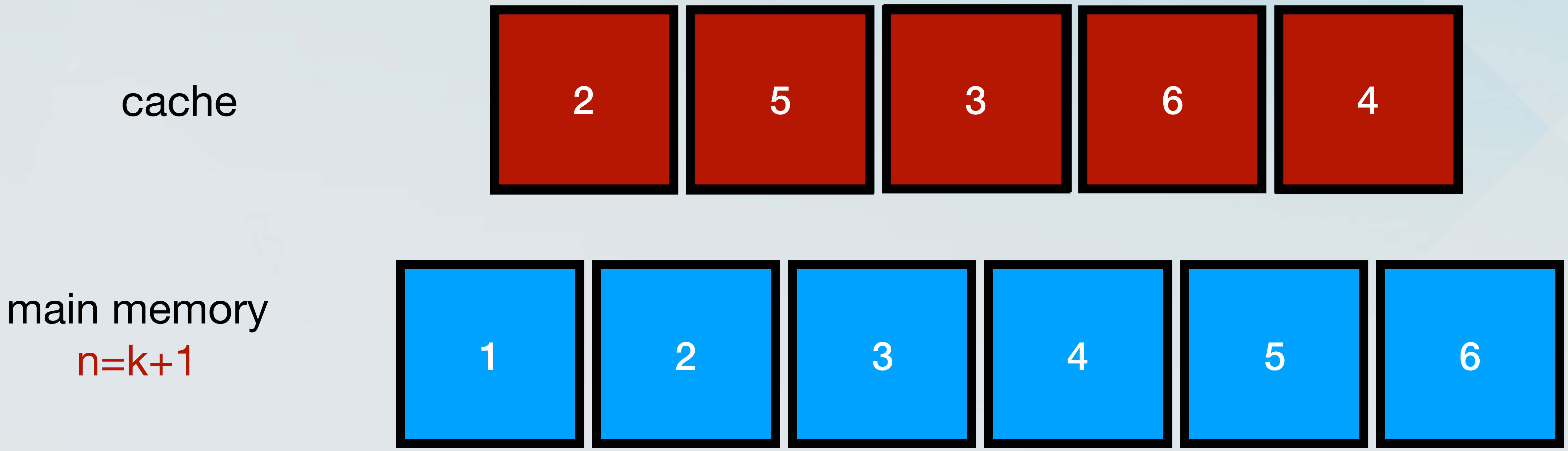
Lower Bound



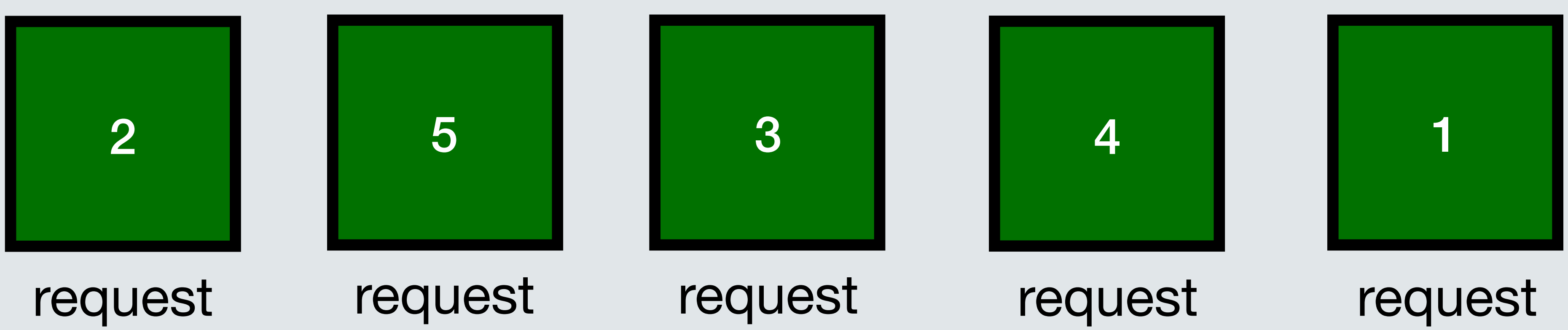
Adversary: Always ask for the page missing from the cache for the algorithm.



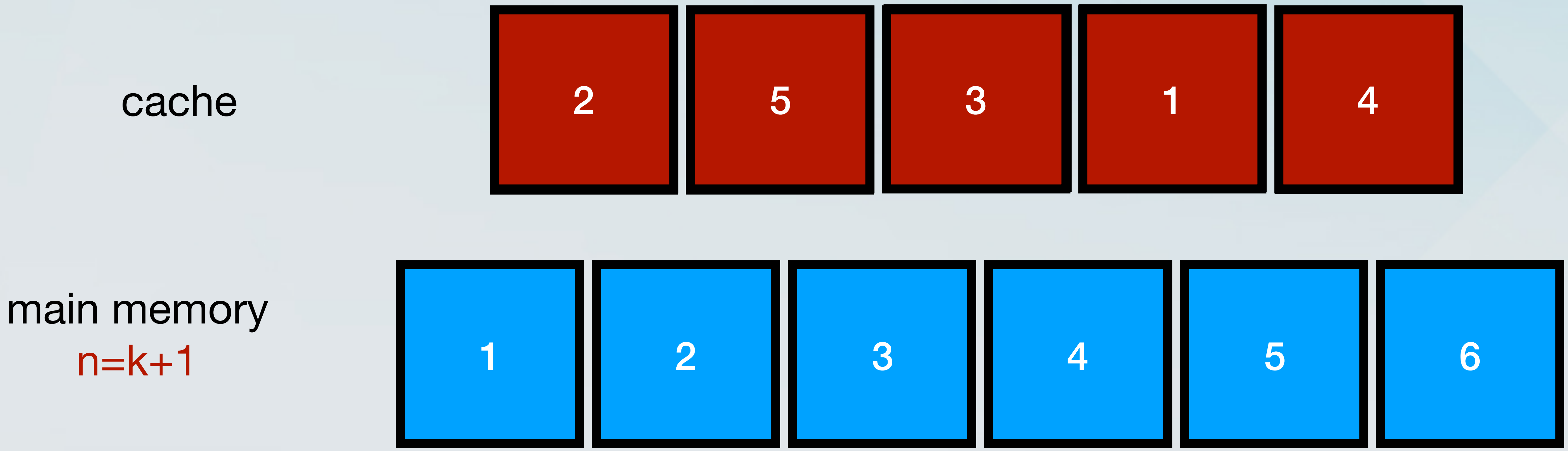
Lower Bound



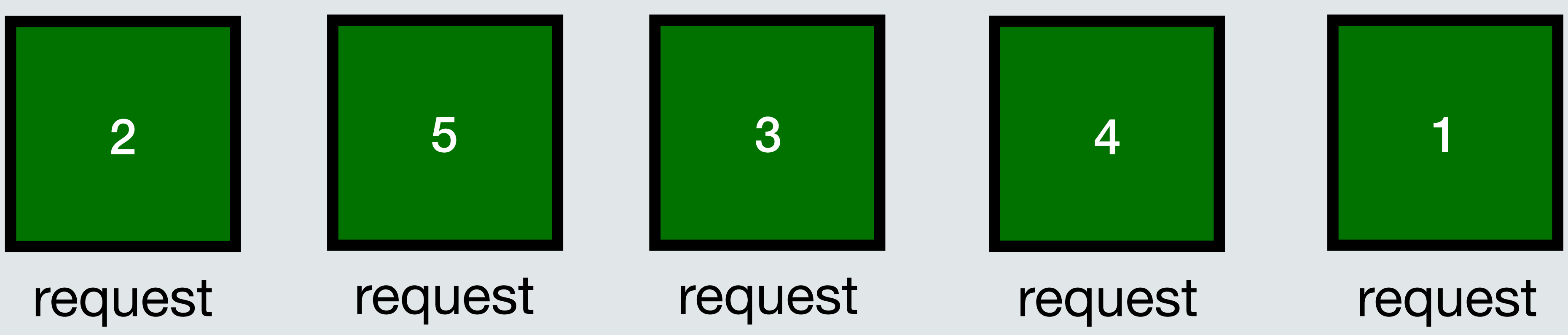
Adversary: Always ask for the page missing from the cache for the algorithm.



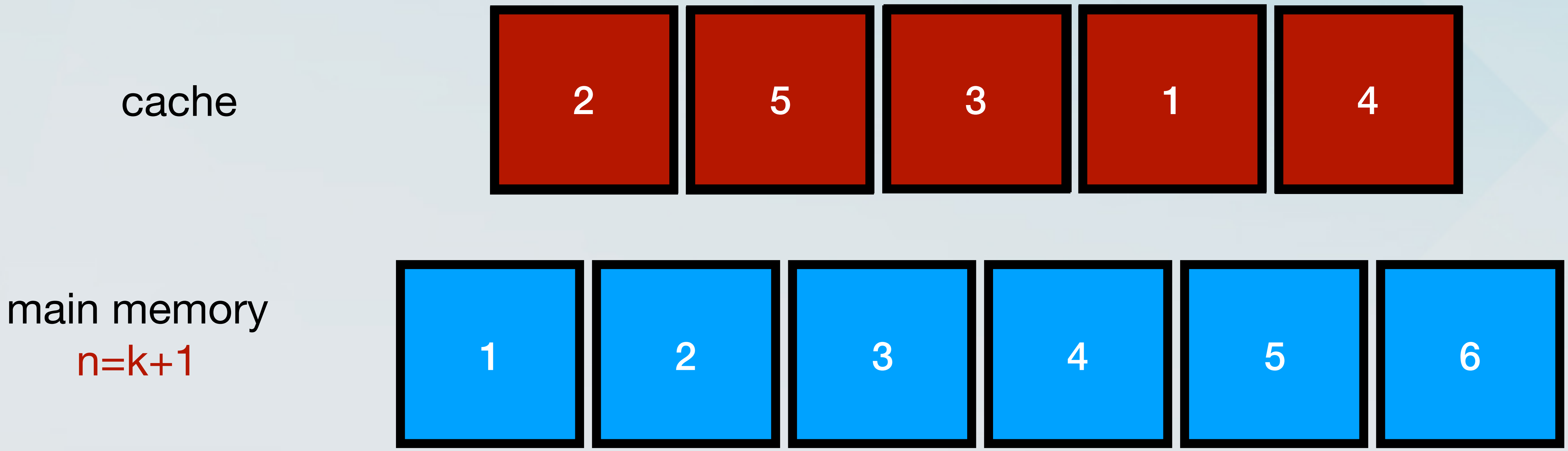
Lower Bound



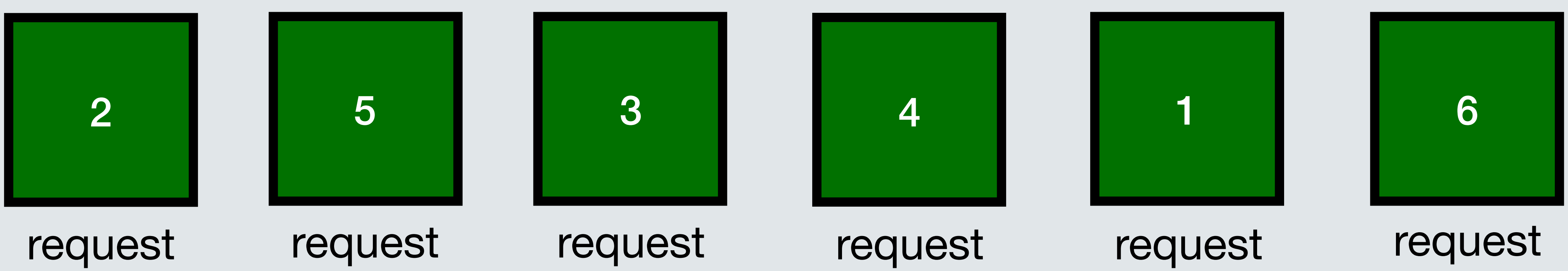
Adversary: Always ask for the page missing from the cache for the algorithm.



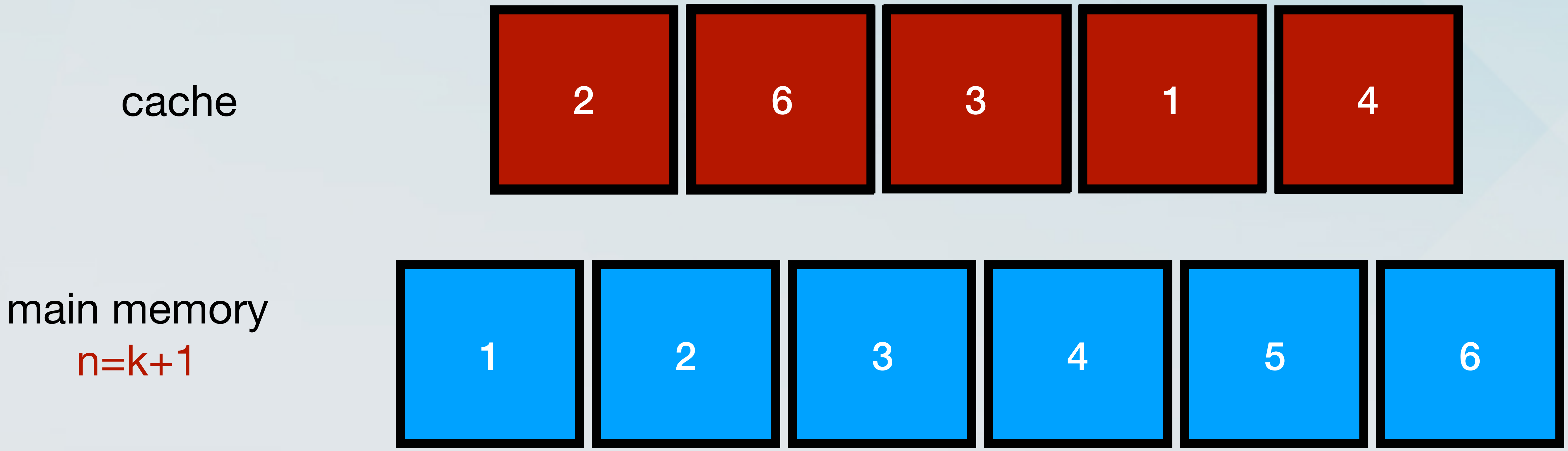
Lower Bound



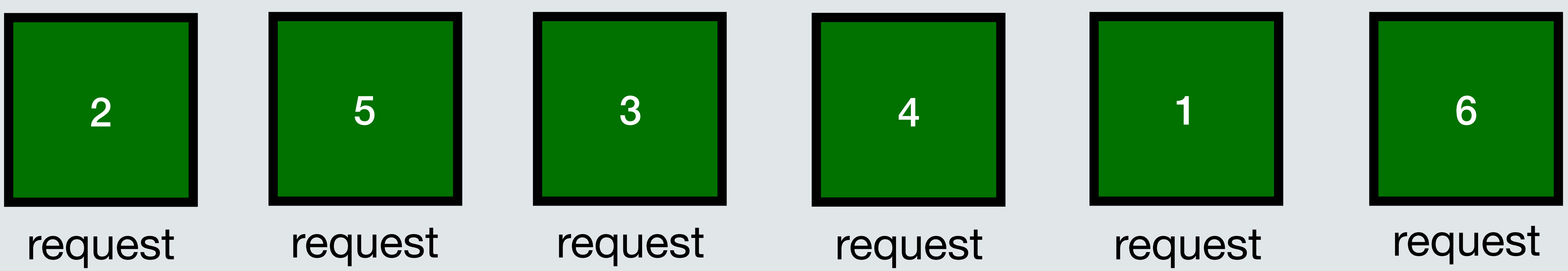
Adversary: Always ask for the page missing from the cache for the algorithm.



Lower Bound



Adversary: Always ask for the page missing from the cache for the algorithm.



Lower Bound

Lower Bound

- The algorithm "*faults*" once at every step.

Lower Bound

- The algorithm “*faults*” once at every step.
- What about the offline optimal?

Lower Bound

- The algorithm “*faults*” once at every step.
- What about the offline optimal?
    - Consider the strategy: “*When replacing a page, replace the one that will be requested the furthest in the future*”.

Lower Bound

- The algorithm "*faults*" once at every step.
- What about the offline optimal?
    - Consider the strategy: "*When replacing a page, replace the one that will be requested the furthest in the future*".
- Suppose that OPT "*faults*" on some page p. OPT replaces a page (to bring in p) that will not be requested in the next k-1 steps.

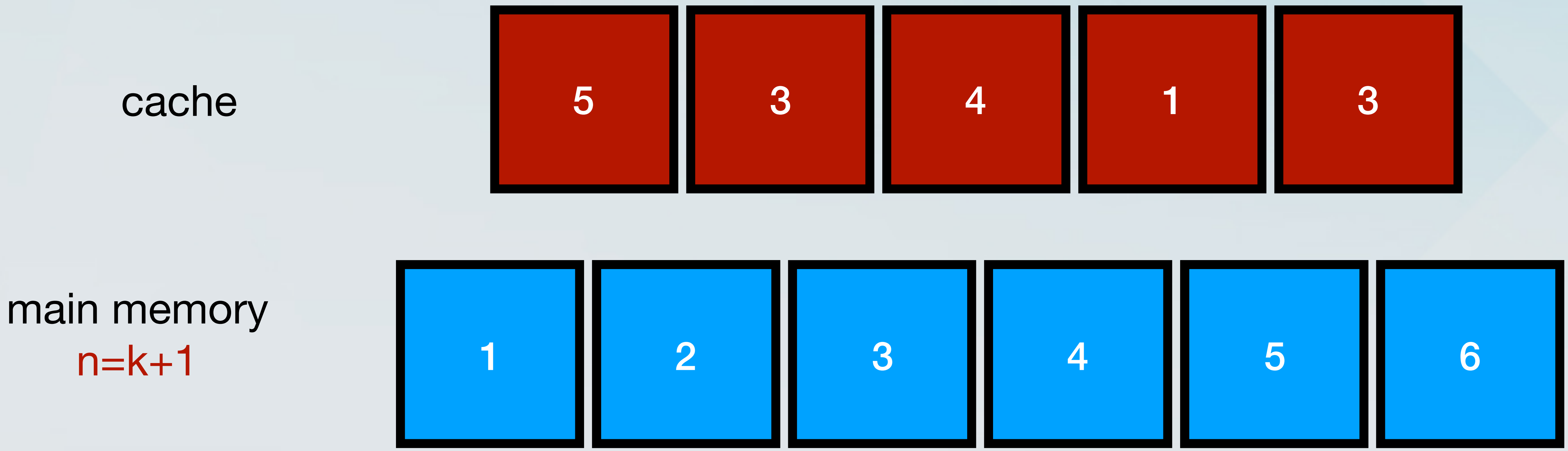
Lower Bound

- The algorithm “*faults*” once at every step.
- What about the offline optimal?
    - Consider the strategy: “*When replacing a page, replace the one that will be requested the furthest in the future*”.
- Suppose that OPT “*faults*” on some page p. OPT replaces a page (to bring in p) that will not be requested in the next k-1 steps.
    - OPT “faults” once every k steps.

Lower Bound

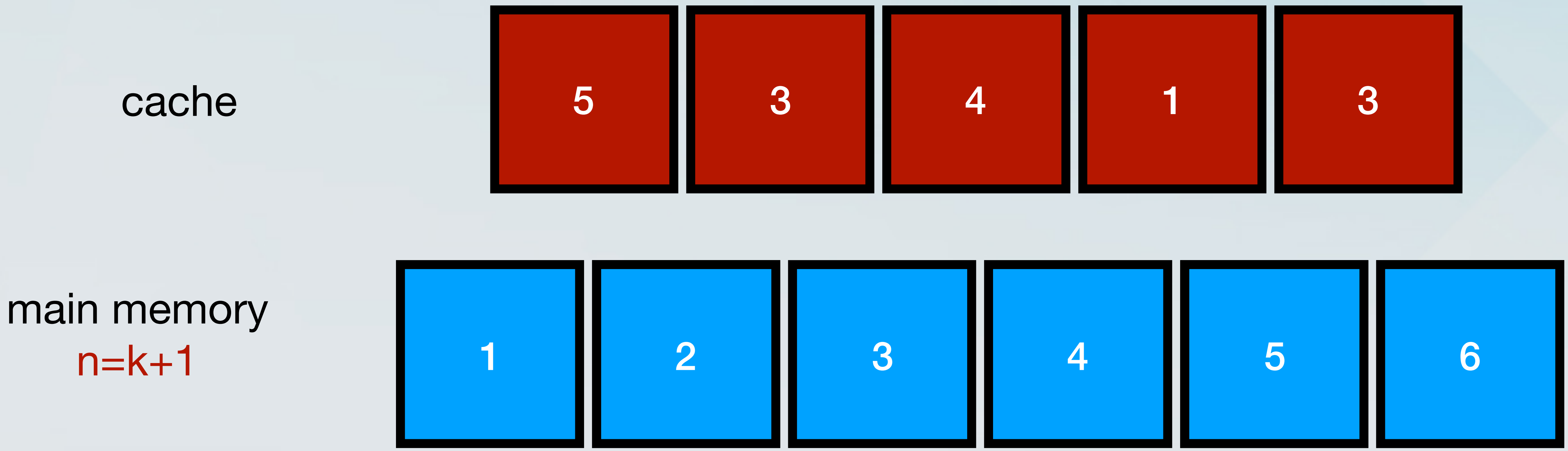
- The algorithm “*faults*” once at every step.
- What about the offline optimal?
    - Consider the strategy: “*When replacing a page, replace the one that will be requested the furthest in the future*”.
- Suppose that OPT “*faults*” on some page p. OPT replaces a page (to bring in p) that will not be requested in the next k-1 steps.
    - OPT “faults” once every k steps.
    - The competitive ratio is at least k.

Lower Bound

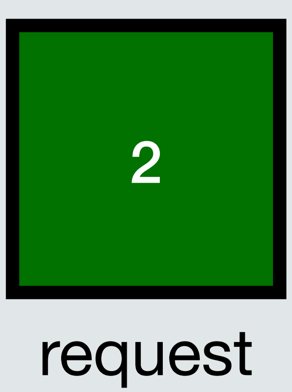


Adversary: Always ask for the page missing from the cache for the algorithm.

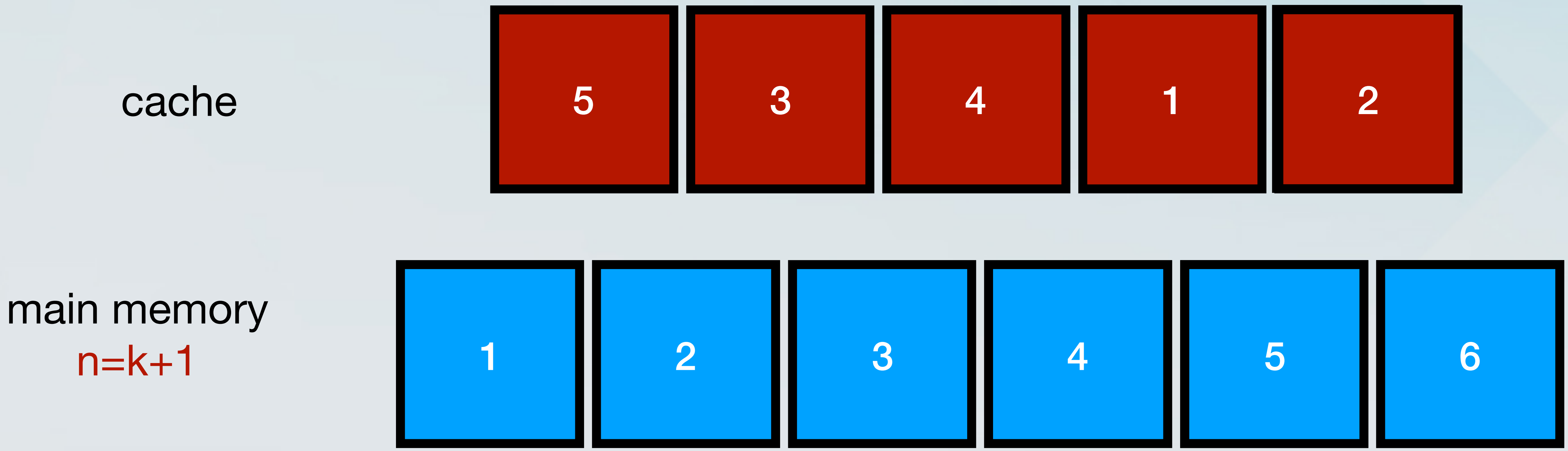
Lower Bound



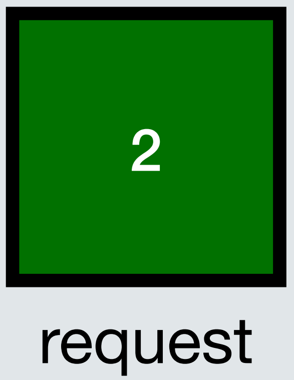
Adversary: Always ask for the page missing from the cache for the algorithm.



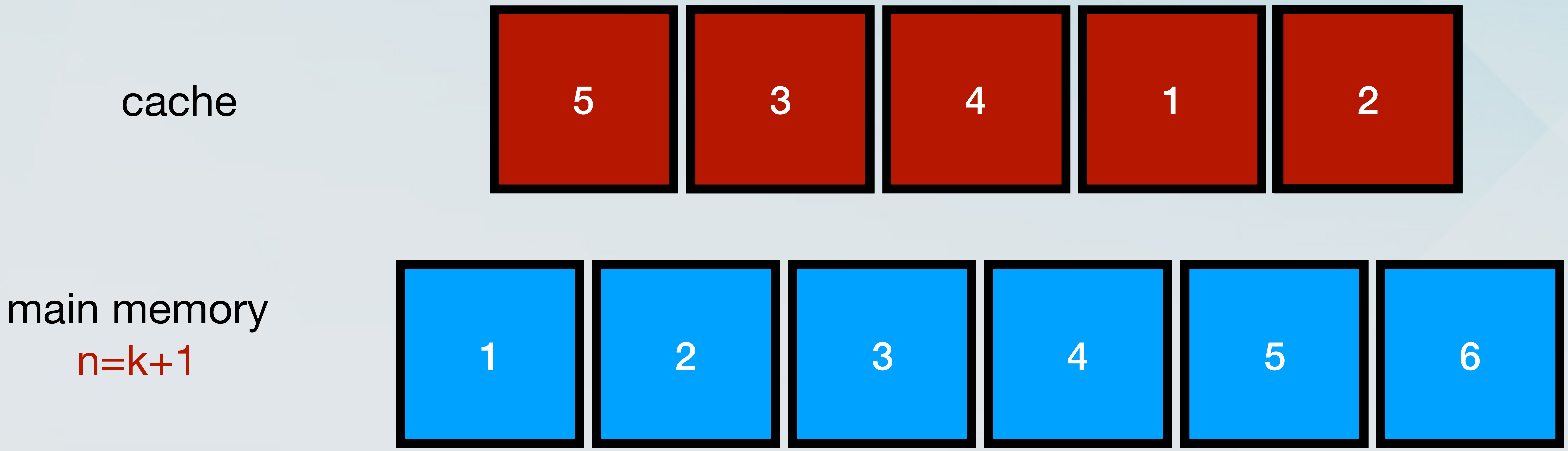
Lower Bound



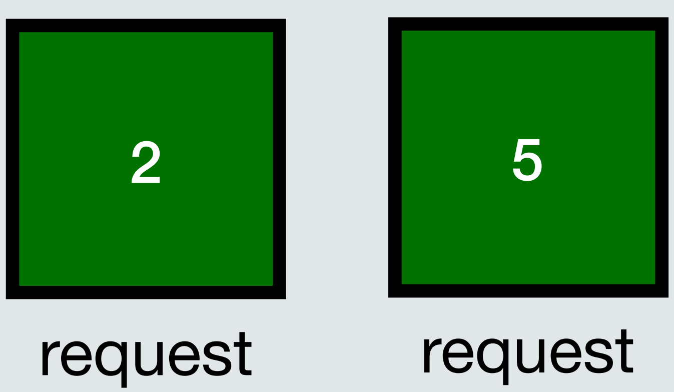
Adversary: Always ask for the page missing from the cache for the algorithm.



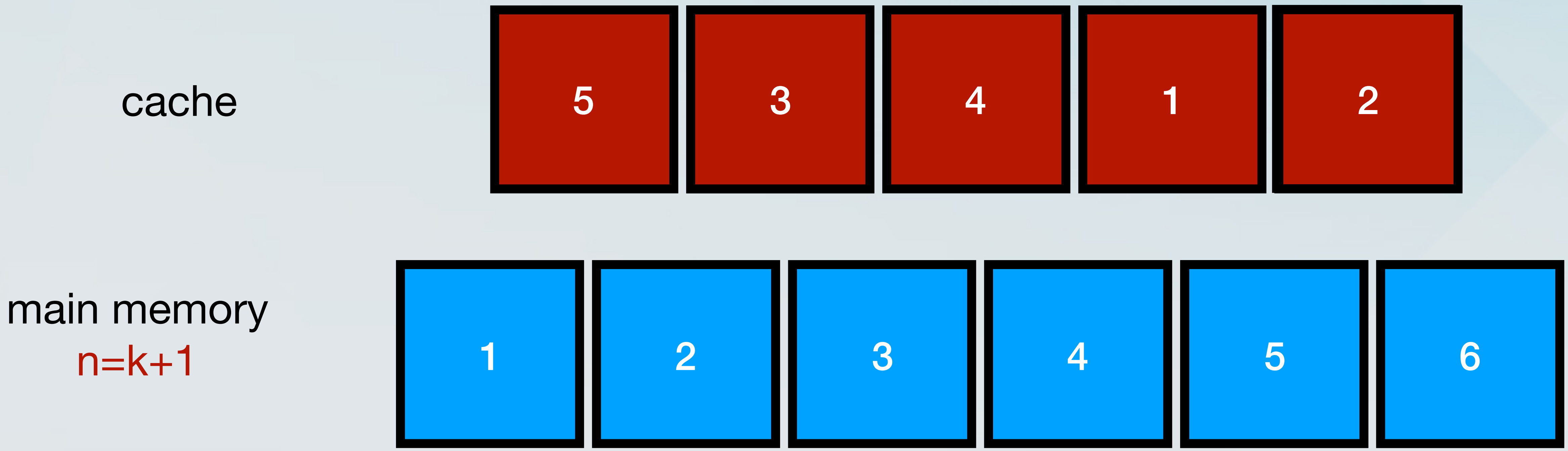
Lower Bound



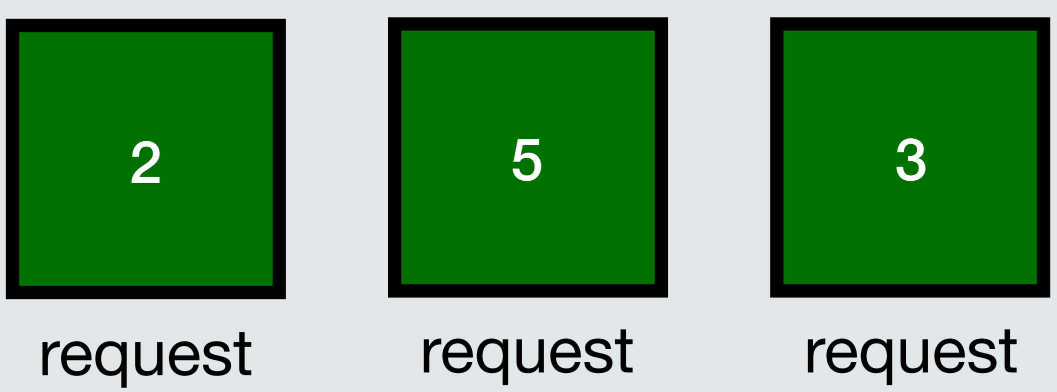
Adversary: Always ask for the page missing from the cache for the algorithm.



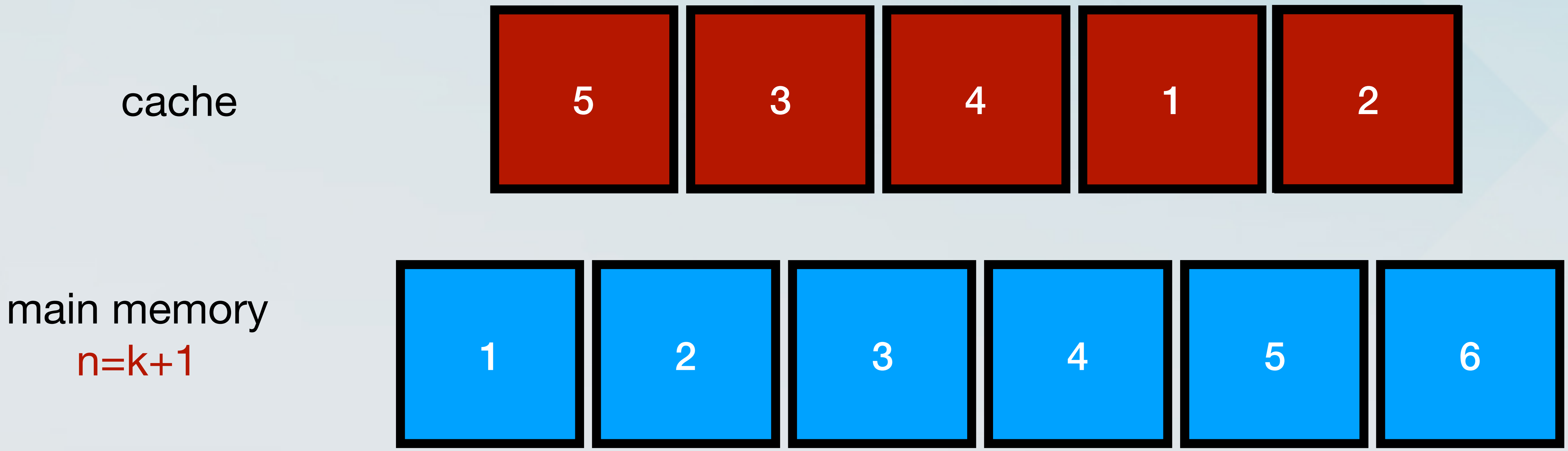
Lower Bound



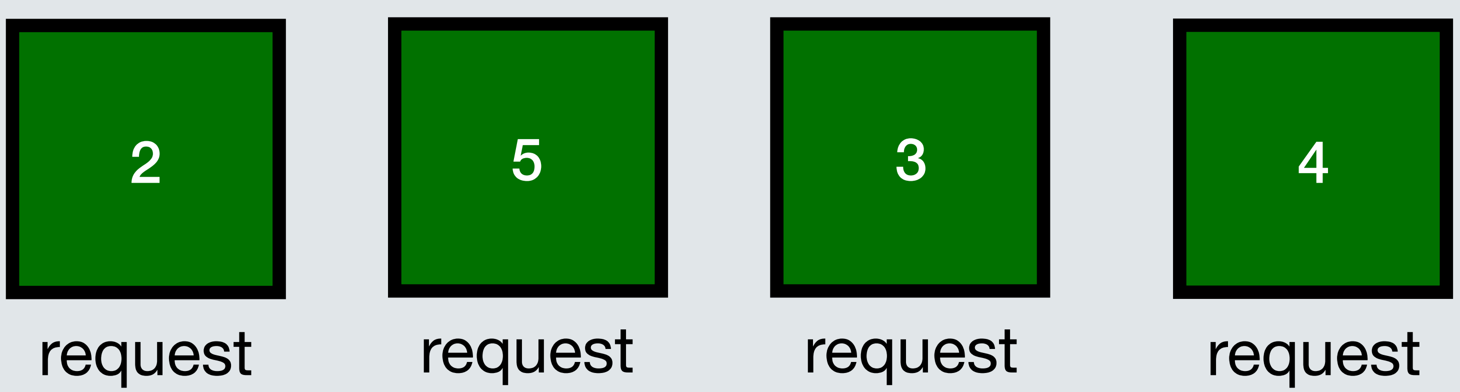
Adversary: Always ask for the page missing from the cache for the algorithm.



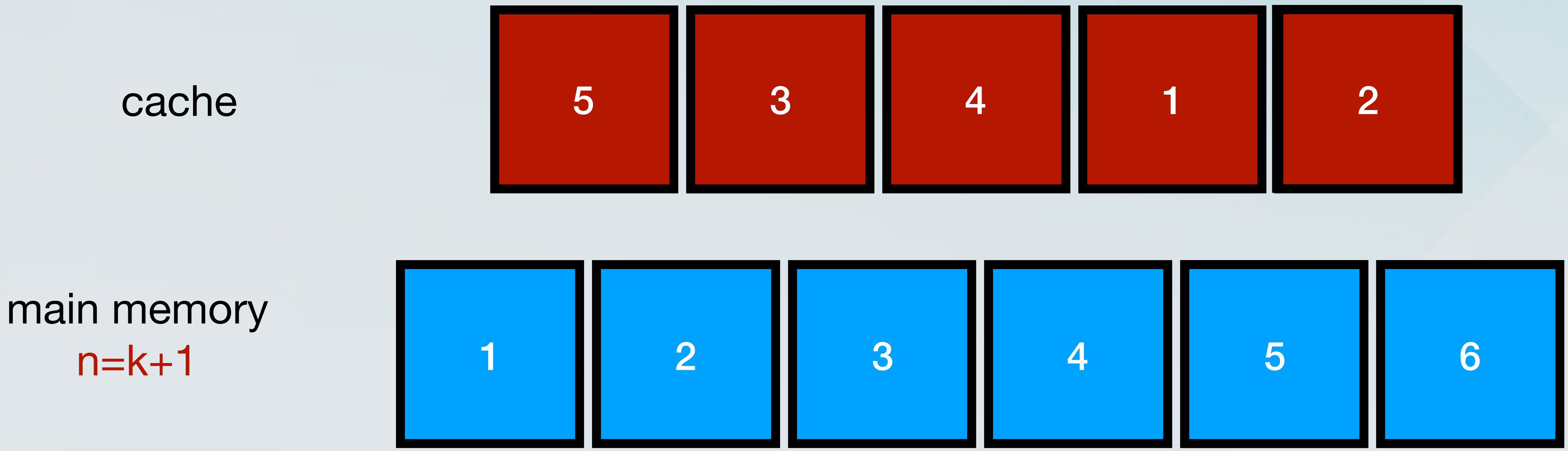
Lower Bound



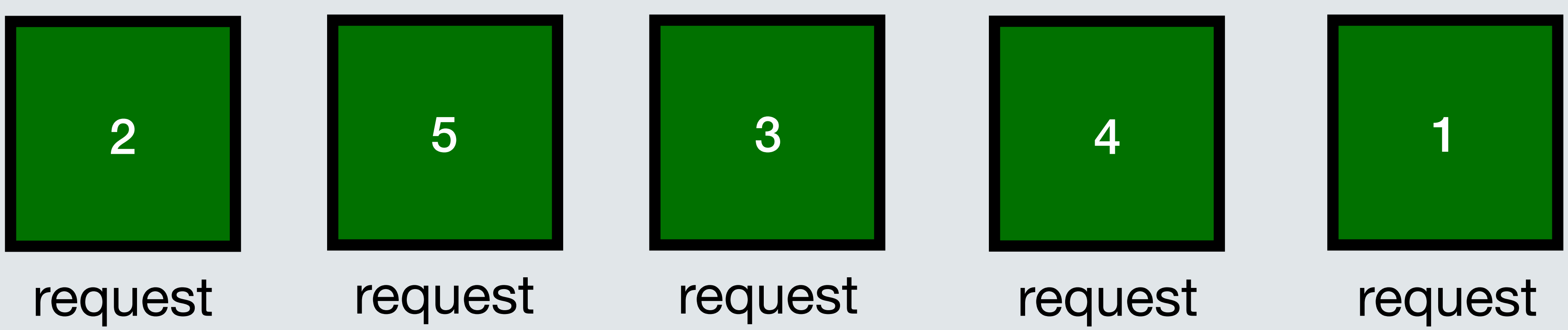
Adversary: Always ask for the page missing from the cache for the algorithm.



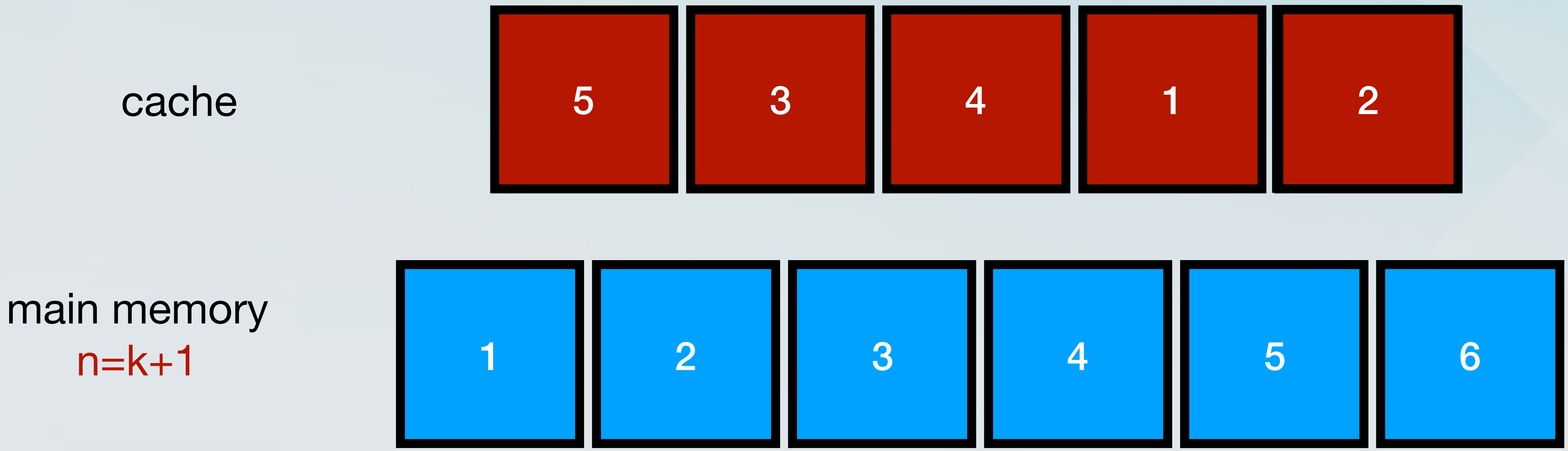
Lower Bound



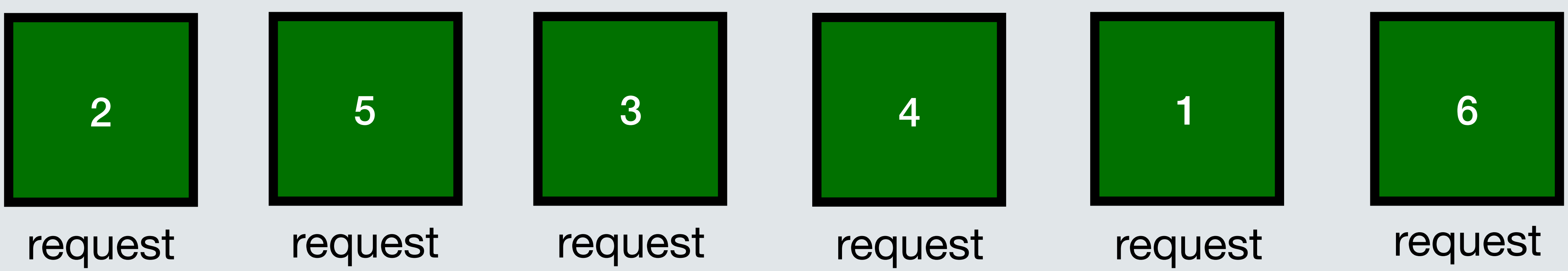
Adversary: Always ask for the page missing from the cache for the algorithm.



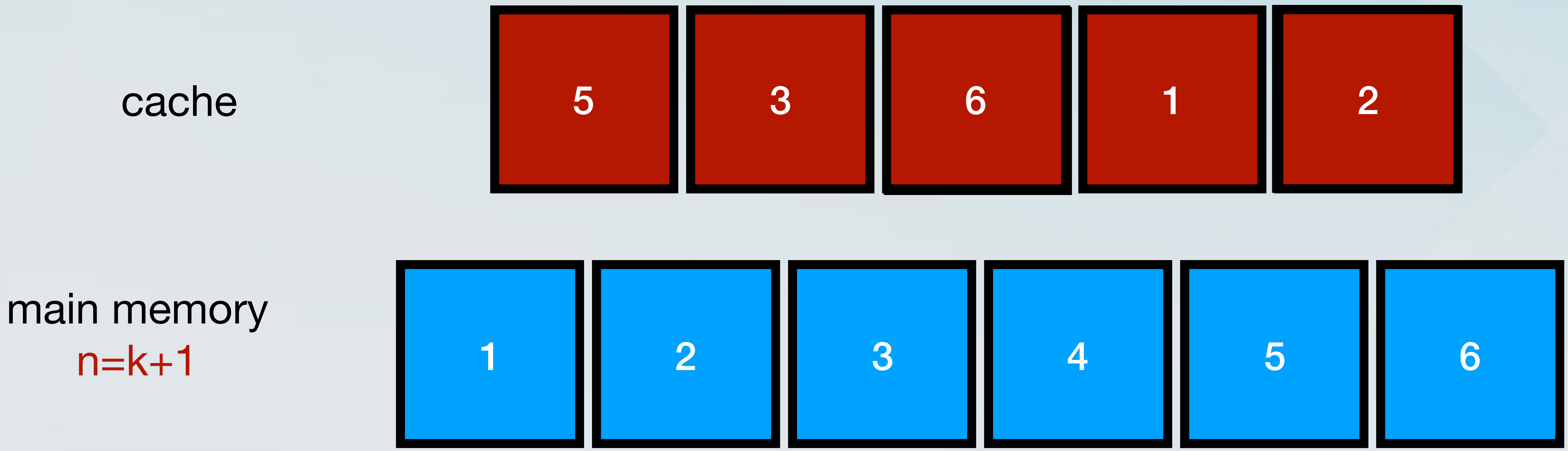
Lower Bound



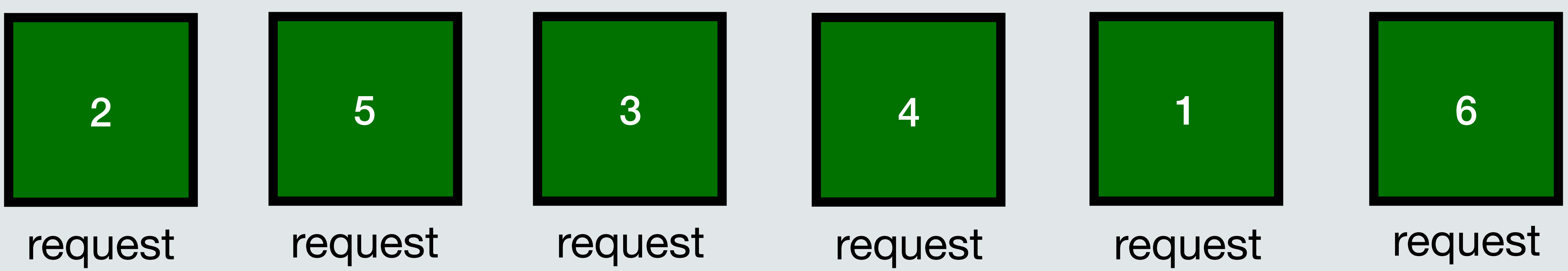
Adversary: Always ask for the page missing from the cache for the algorithm.



Lower Bound



Adversary: Always ask for the page missing from the cache for the algorithm.



Paging Algorithms

- LRU (*Least Recently Used*): Replace the page that was requested the least recently.
- FIFO (*First-In First-Out*): Replace the page that has been in the cache the longest.
- LIFO (*Last-In First-Out*): Replace the page that has been in the cache the shortest.
- LFU (*Least Frequently Used*): Replace the page that was requested the least frequently so far.
- MIN (*Offline Optimal*): Replace the page whose next request happens the furthest in the future.

Paging Algorithms

- Theorem: LRU and FIFO have competitive ratio k.