Online Parallel Paging and Green Paging

Kunal Agrawal
Washington U

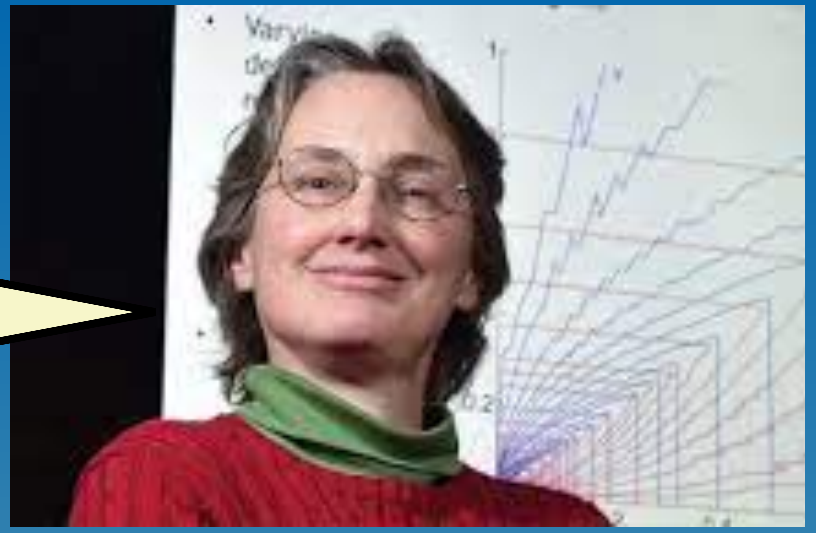
Michael Bender
Stony Brook

Rathish Das
Waterloo→Liverpool

William Kuszmaul
MIT

Enoch Peserico
U Padova

Michele Scquizzato
U Padova



Online Parallel Paging and Green Paging

Kunal Agrawal
Washington U

Michael Bender
Stony Brook

Rathish Das
Waterloo→Liverpool

William Kuszmaul
MIT

Enoch Peserico
U Padova

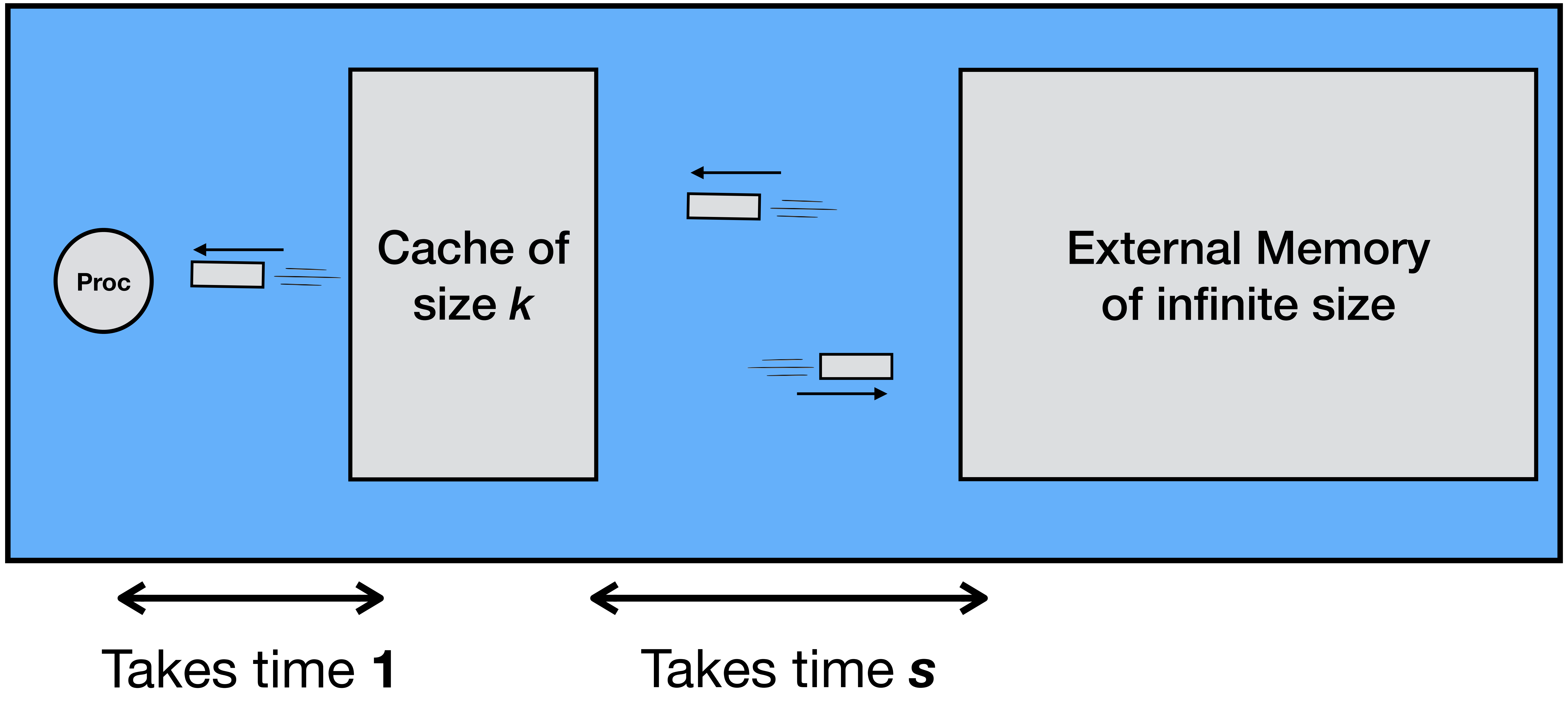
Michele Scquizzato
U Padova

Classical Sequential Paging

[Beladi 66], [Mattson, Gecsei, Slutz, Traiger 70], [Sleator, Tarjan 85], [Borodin, El-Yaniv 98]

Processor makes page/block requests:

$r_1,\ r_2,\ r_3,\ r_4,\ r_5,\ r_6,\ r_7,\ r_8,\ \ldots$



- Up to *k* pages can be kept in cache at a time.
- **Algorithmic decision:** control which pages are moved in/out of cache.

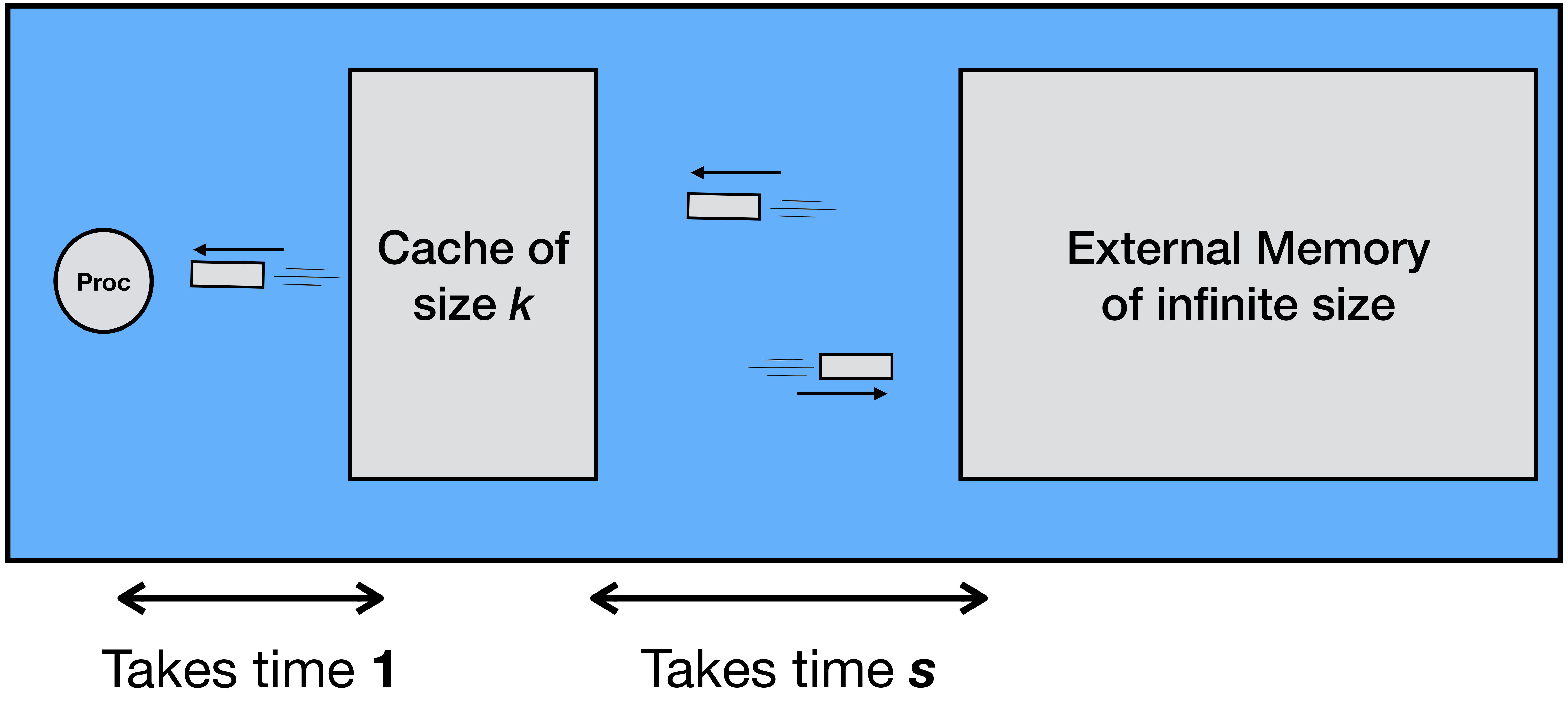
Goal: Complete the request sequence as fast as possible.

Classical (Offline) Sequential Paging

[Beladi 66], [Mattson, Gecsei, Slutz, Traiger 70], [Sleator, Tarjan 85], [Borodin, El-Yaniv 98]

Processor makes page/block requests:

$r_1, \; r_2, \; r_3, \; r_4, \; r_5, \; r_6, \; r_7, \; r_8, \; \ldots$



Offline Opt: Evict the page that will be used *farthest in future*.

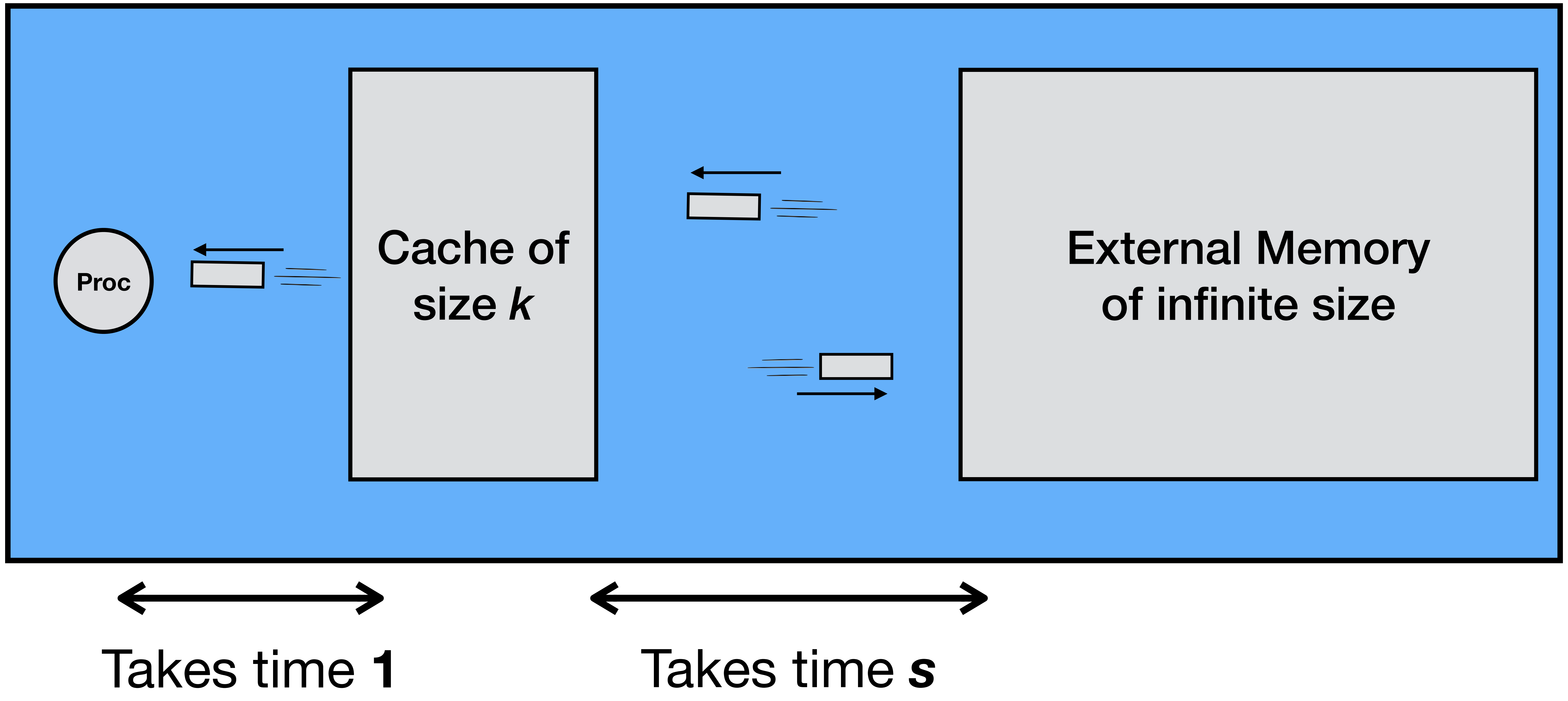
[Beladi 66], [Mattson, Gecsei, Slutz, Traiger 70]

Classical Online Sequential Paging

[Beladi 66], [Mattson, Gecsei, Slutz, Traiger 70], [Sleator, Tarjan 85], [Borodin, El-Yaniv 98]

Processor makes page/block requests:

$r_1,\ r_2,\ r_3,\ r_4,\ r_5,\ r_6,\ r_7,\ r_8,\ \ldots$



Natural online alg: Always evict the page that was *least recently used (LRU).*

Classical Theorem [Sleator, Tarjan 85]**:**

With O(1) resource augmentation, LRU is O(1)-competitive,

i.e., $LRU_{k} \leq 2\ OPT_{k/2}$.

The Parallel Paging Problem

[Fiat and Karlin, STOC 1995] [Hassidim ICS 2010]
[López-Ortiz & Salinger ITCS 2012, WAOA 2012]

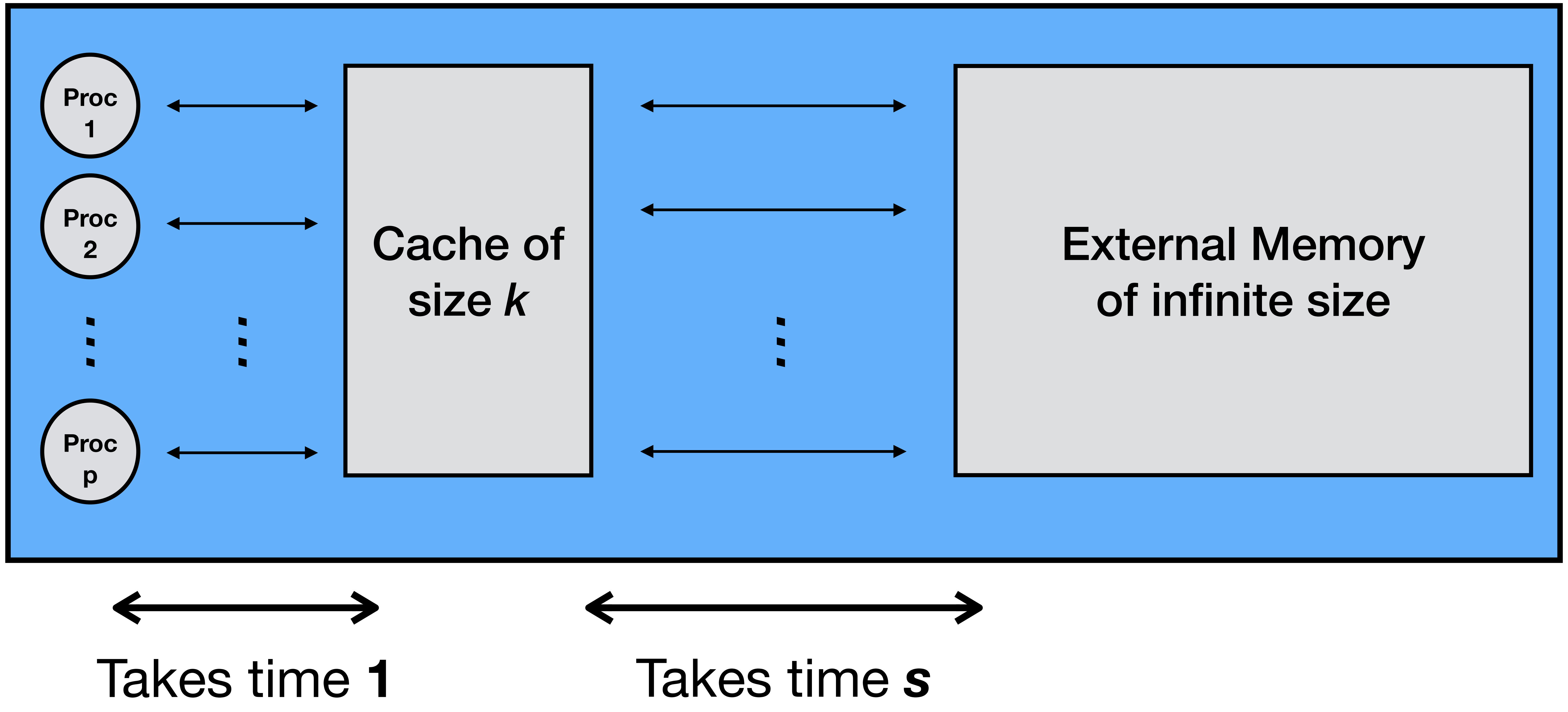
p request sequences occur in parallel:

$r_{11},\ r_{12},\ r_{13},\ r_{14},\ r_{15},\ r_{16},\ \ldots.$

$r_{21},\ r_{22},\ r_{23},\ r_{24},\ r_{25},\ r_{26},\ \ldots.$

$\vdots$

$r_{p1},\ r_{p2},\ r_{p3},\ r_{p4},\ r_{p5},\ r_{p6},\ \ldots.$



- Different processors access disjoint sets of pages.
- Processors can access cache in parallel.
- Processors move pages between cache and external memory in parallel.

But processors must share cache of size k.

The Parallel Paging Problem

[Fiat and Karlin, STOC 1995] [Hassidim ICS 2010]
[López-Ortiz & Salinger ITCS 2012, WAOA 2012]

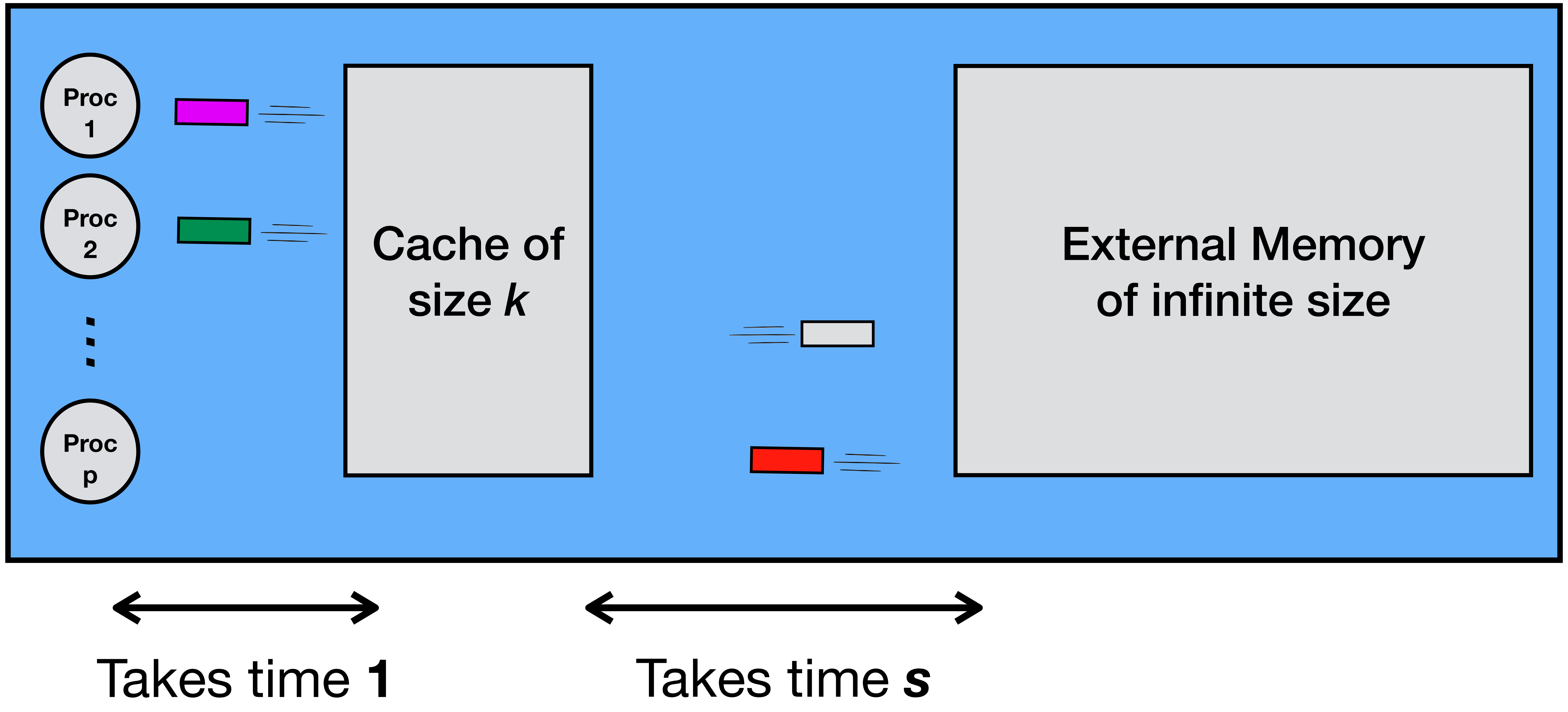
p request sequences occur in parallel:

$r_{11},\ r_{12},\ r_{13},\ r_{14},\ r_{15},\ r_{16},\ ....$

$r_{21},\ r_{22},\ r_{23},\ r_{24},\ r_{25},\ r_{26},\ ....$

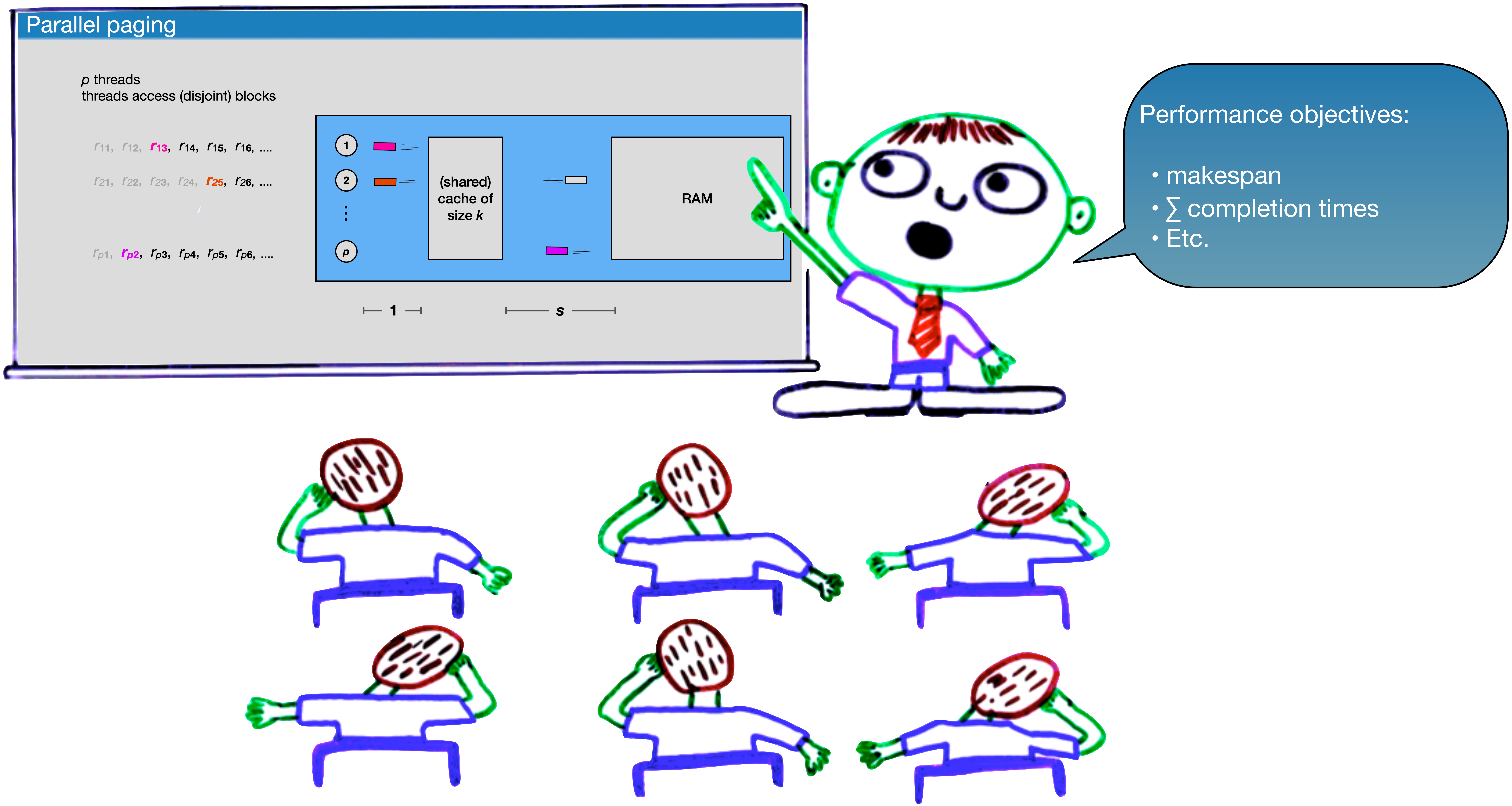
⋮

$r_{p1},\ r_{p2},\ r_{p3},\ r_{p4},\ r_{p5},\ r_{p6},\ ....$

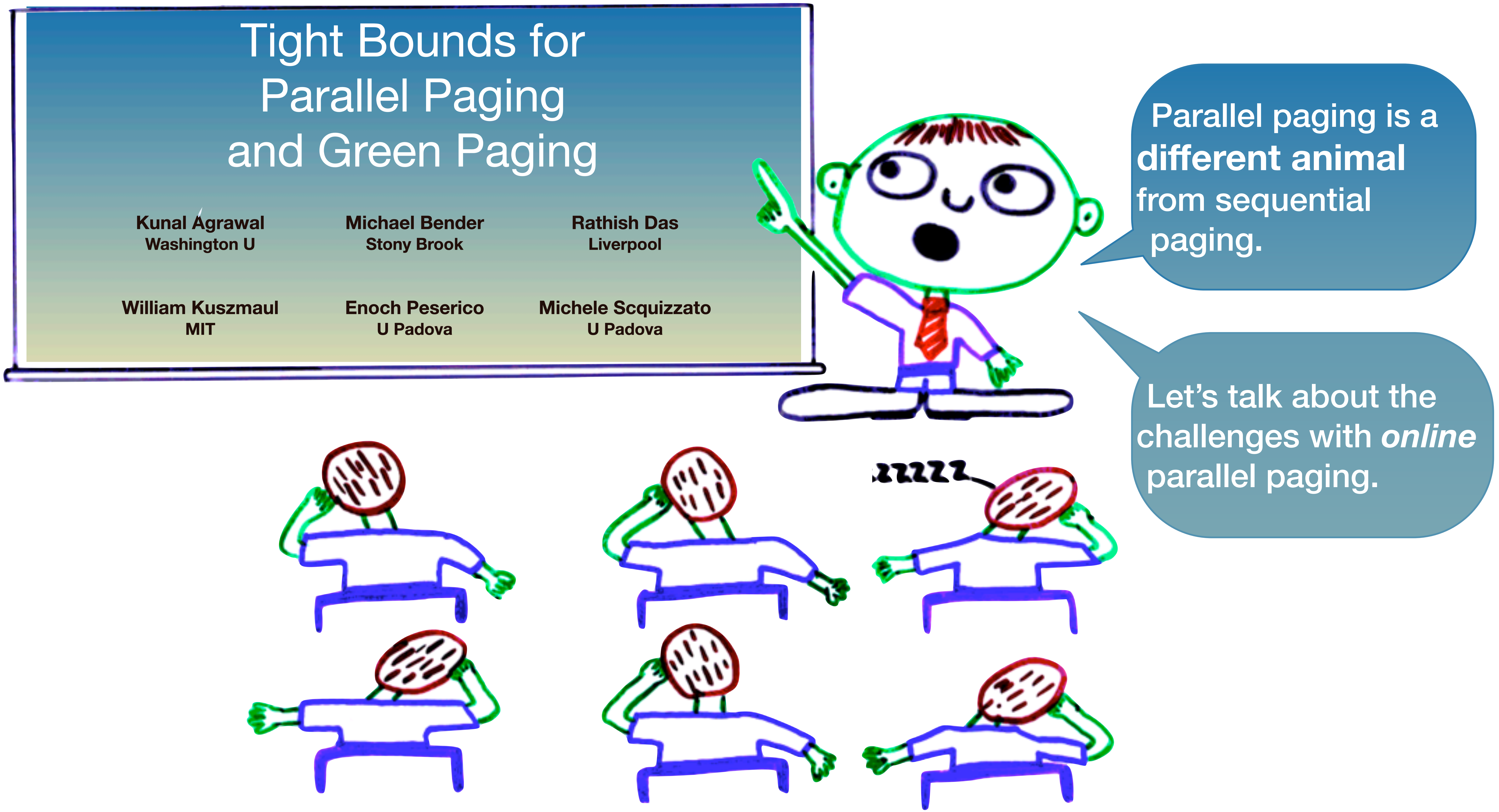


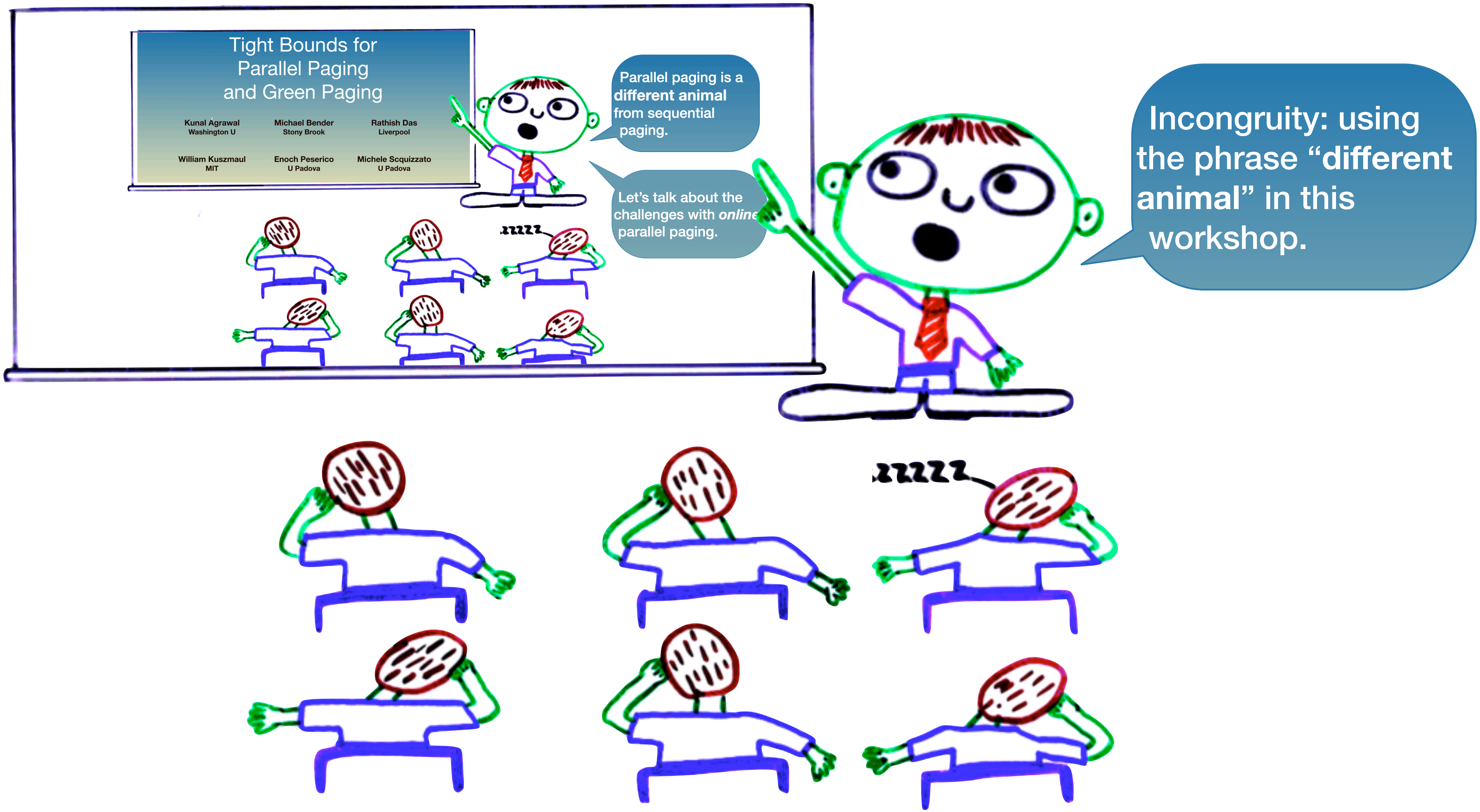
Algorithmic decision:

when a new block is brought into cache, which block should be evicted?



Parallel paging
p threads
threads access (disjoint) blocks
r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, ...
r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, ...
r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, ...
1
2
p
(shared) cache of size k
RAM
1
s
Performance objectives:
• makespan
• ∑ completion times
• Etc.


Tight Bounds for
Parallel Paging
and Green Paging
Kunal Agrawal
Washington U
Michael Bender
Stony Brook
Rathish Das
Liverpool
William Kuszmaul
MIT
Enoch Peserico
U Padova
Michele Scquizzato
U Padova
Parallel paging is a **different animal** from sequential paging.
Let's talk about the challenges with *online* parallel paging.
zzzzz



Tight Bounds for Parallel Paging and Green Paging
Kunal Agrawal Washington U
Michael Bender Stony Brook
Rathish Das Liverpool
William Kuszmaul MIT
Enoch Peserico U Padova
Michele Scquizzato U Padova
Parallel paging is a **different animal** from sequential paging.
Let's talk about the challenges with *online* parallel paging.
zzzzz
Incongruity: using the phrase "**different animal**" in this workshop.
zzzzz

Tight Bounds for Parallel Paging and Green Paging
Kunal Agrawal Washington U
Michael Bender Stony Brook
Rathish Das Liverpool
William Kuszmaul MIT
Enoch Peserico U Padova
Michele Scquizzato U Padova
Parallel paging is a **different animal** from sequential paging.
Let's talk about the challenges with ***online*** parallel paging.
zzzzz
Incongruity: using the phrase "**different animal**" in this workshop.
zzzzz

Challenge 1: how to partition the cache among the threads?

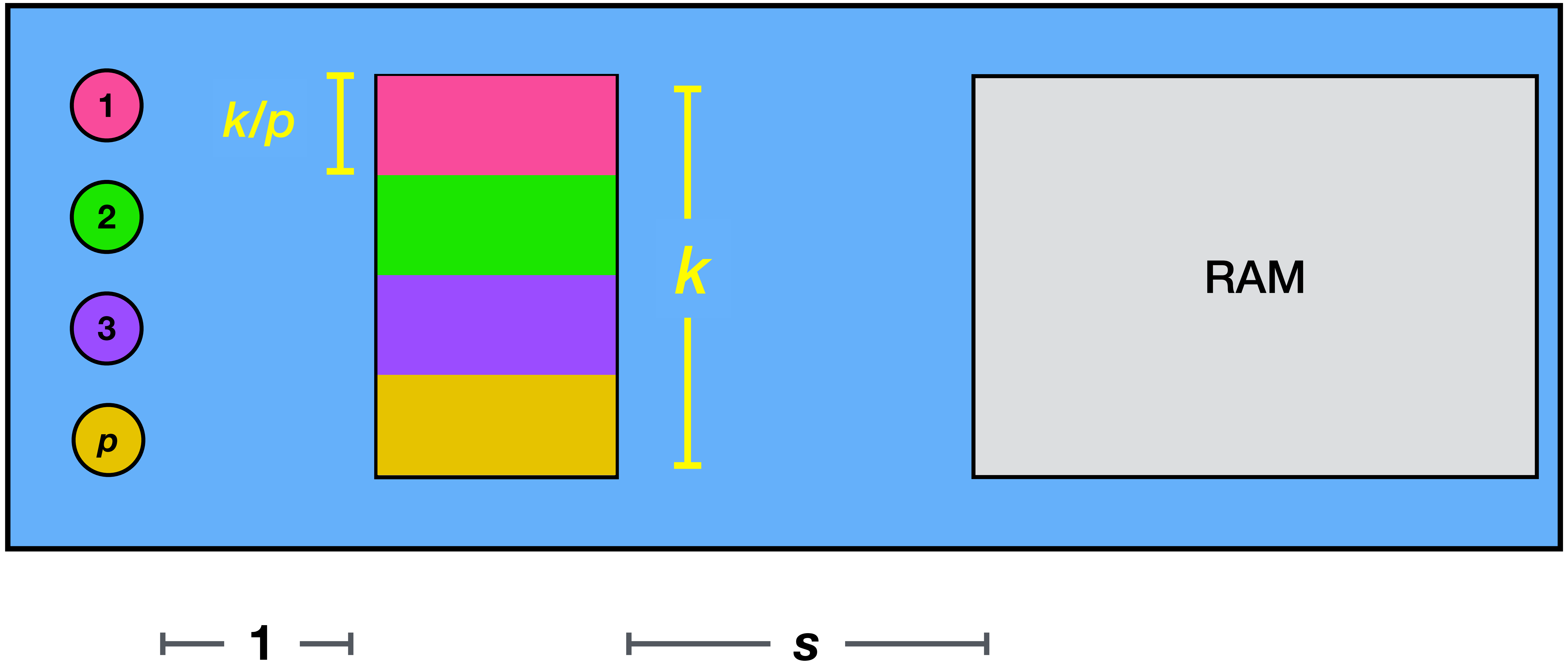
p threads
threads access (disjoint) blocks

$r_{11},\ r_{12},\ r_{13},\ r_{14},\ r_{15},\ r_{16},\ \ldots.$

$r_{21},\ r_{22},\ r_{23},\ r_{24},\ r_{25},\ r_{26},\ \ldots.$

$r_{31},\ r_{32},\ r_{33},\ r_{34},\ r_{35},\ r_{36},\ \ldots.$

$r_{p1},\ r_{p2},\ r_{p3},\ r_{p4},\ r_{p5},\ r_{p6},\ \ldots.$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Challenge 1: how to partition the cache among the threads?

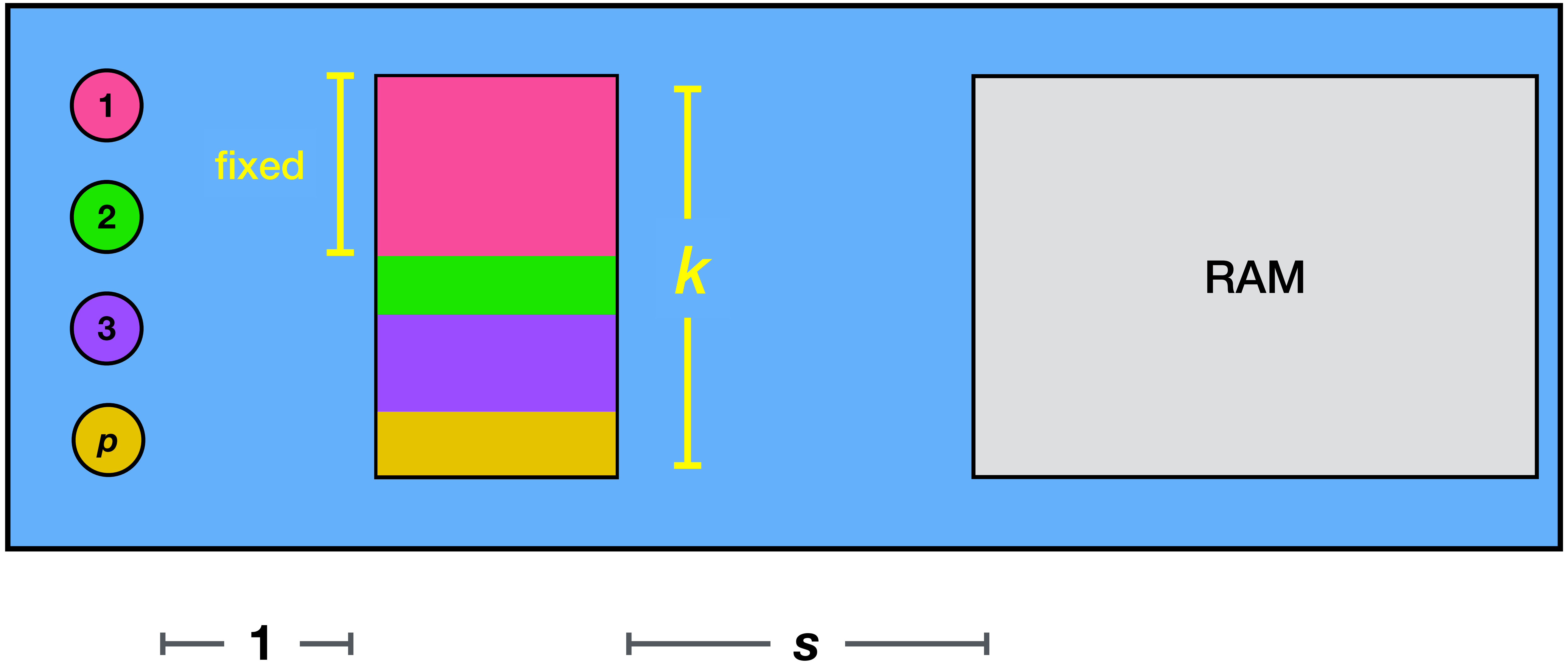
p threads
threads access (disjoint) blocks

$r_{11},\ r_{12},\ r_{13},\ r_{14},\ r_{15},\ r_{16},\ ....$

$r_{21},\ r_{22},\ r_{23},\ r_{24},\ r_{25},\ r_{26},\ ....$

$r_{31},\ r_{32},\ r_{33},\ r_{34},\ r_{35},\ r_{36},\ ....$

$r_{p1},\ r_{p2},\ r_{p3},\ r_{p4},\ r_{p5},\ r_{p6},\ ....$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge 1: how to partition the cache among the threads?

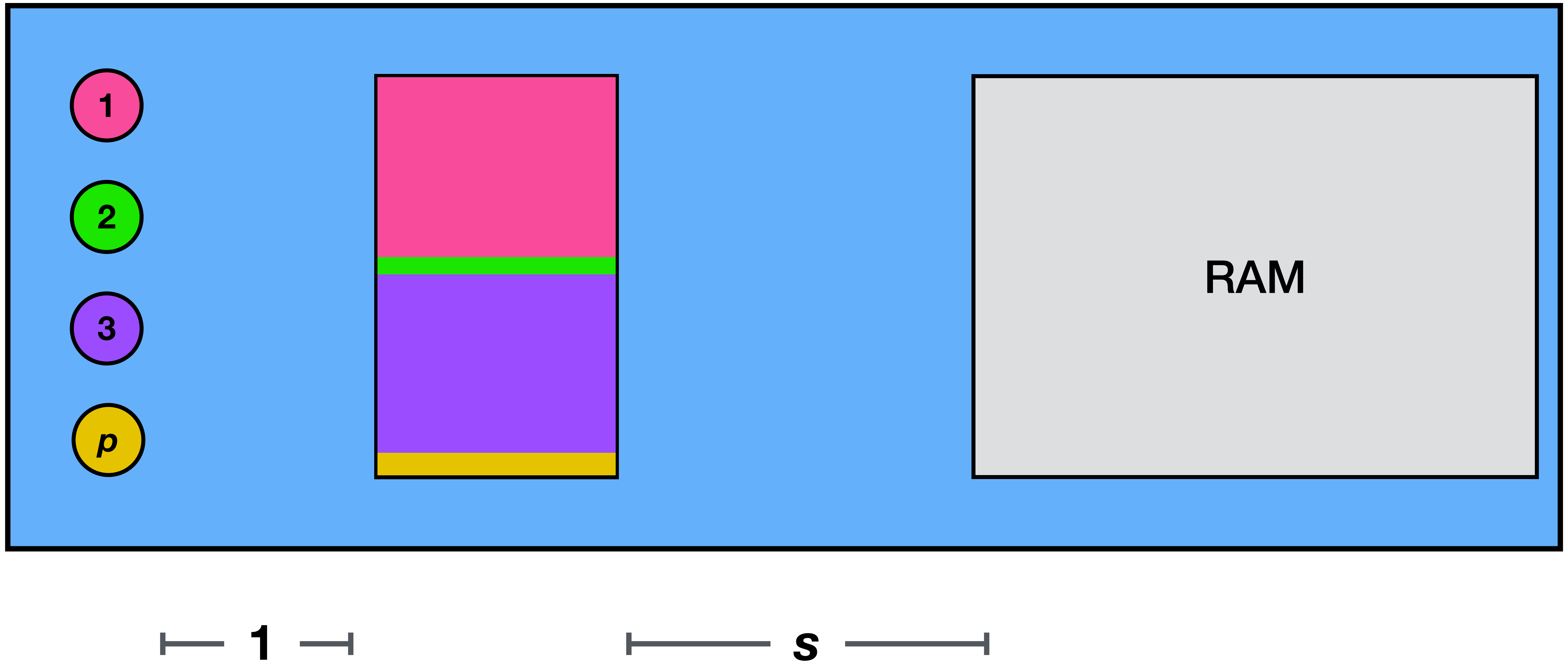
p threads
threads access (disjoint) blocks

$r_{11},\ r_{12},\ r_{13},\ r_{14},\ r_{15},\ r_{16},\ \ldots.$

$r_{21},\ r_{22},\ r_{23},\ r_{24},\ r_{25},\ r_{26},\ \ldots.$

$r_{31},\ r_{32},\ r_{33},\ r_{34},\ r_{35},\ r_{36},\ \ldots.$

$r_{p1},\ r_{p2},\ r_{p3},\ r_{p4},\ r_{p5},\ r_{p6},\ \ldots.$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

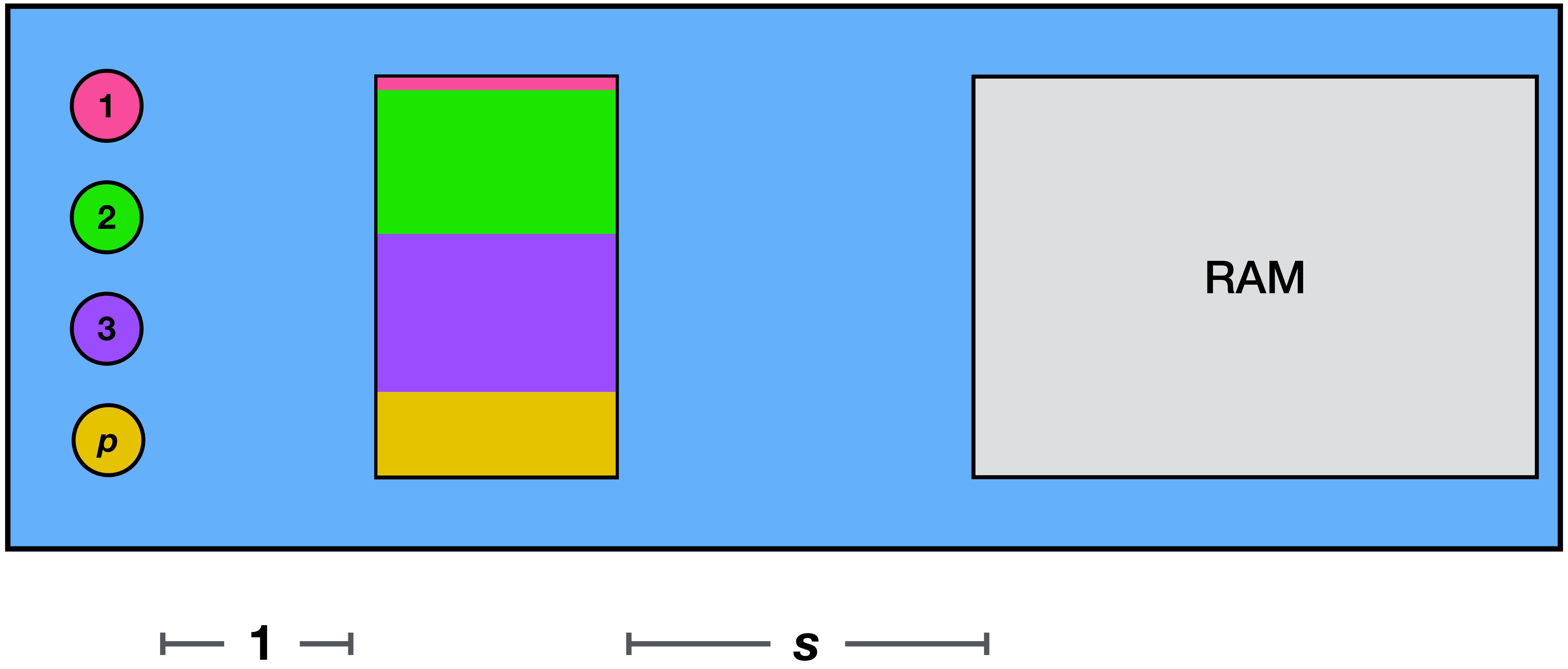
p threads
threads access (disjoint) blocks

$r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, \ldots$

$r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, \ldots$

$r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{36}, \ldots$

$r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, \ldots$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

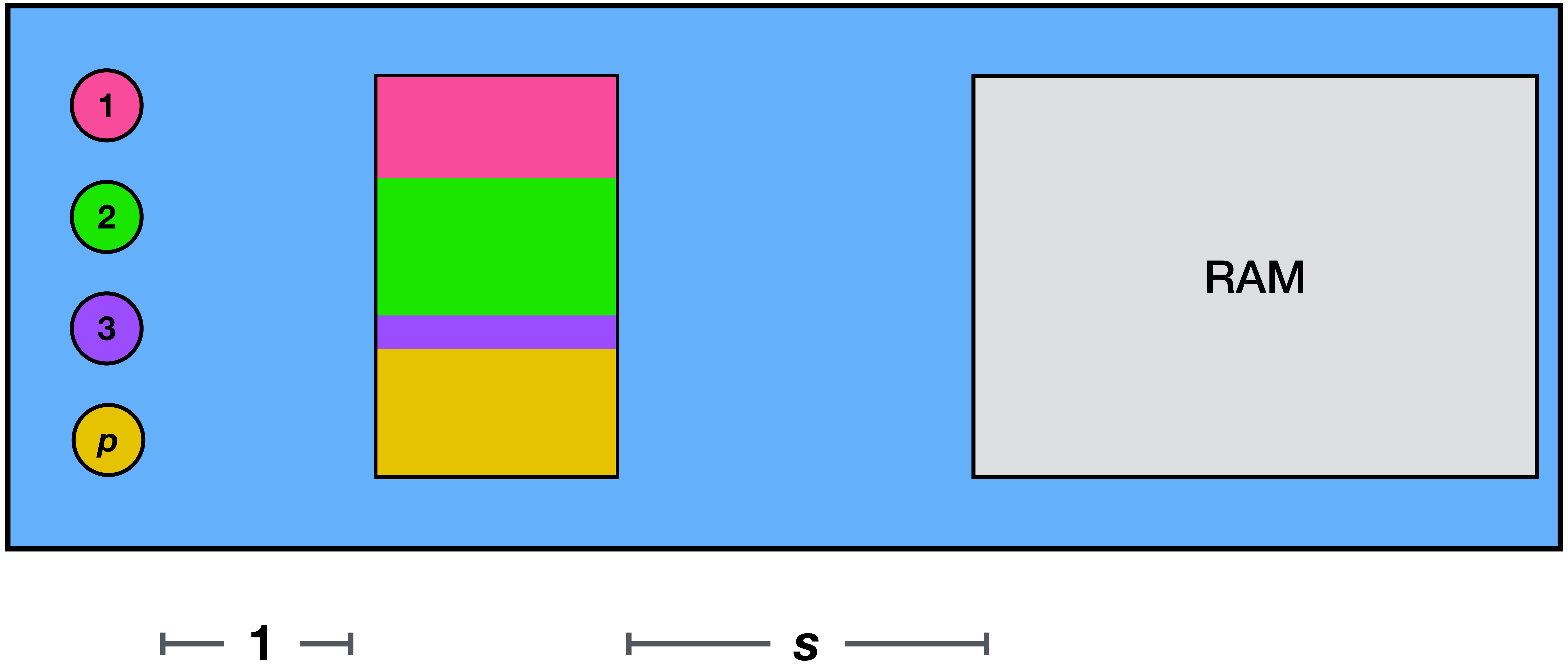
p threads
threads access (disjoint) blocks

r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, ...

r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, ...

r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{36}, ...

r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, ...



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

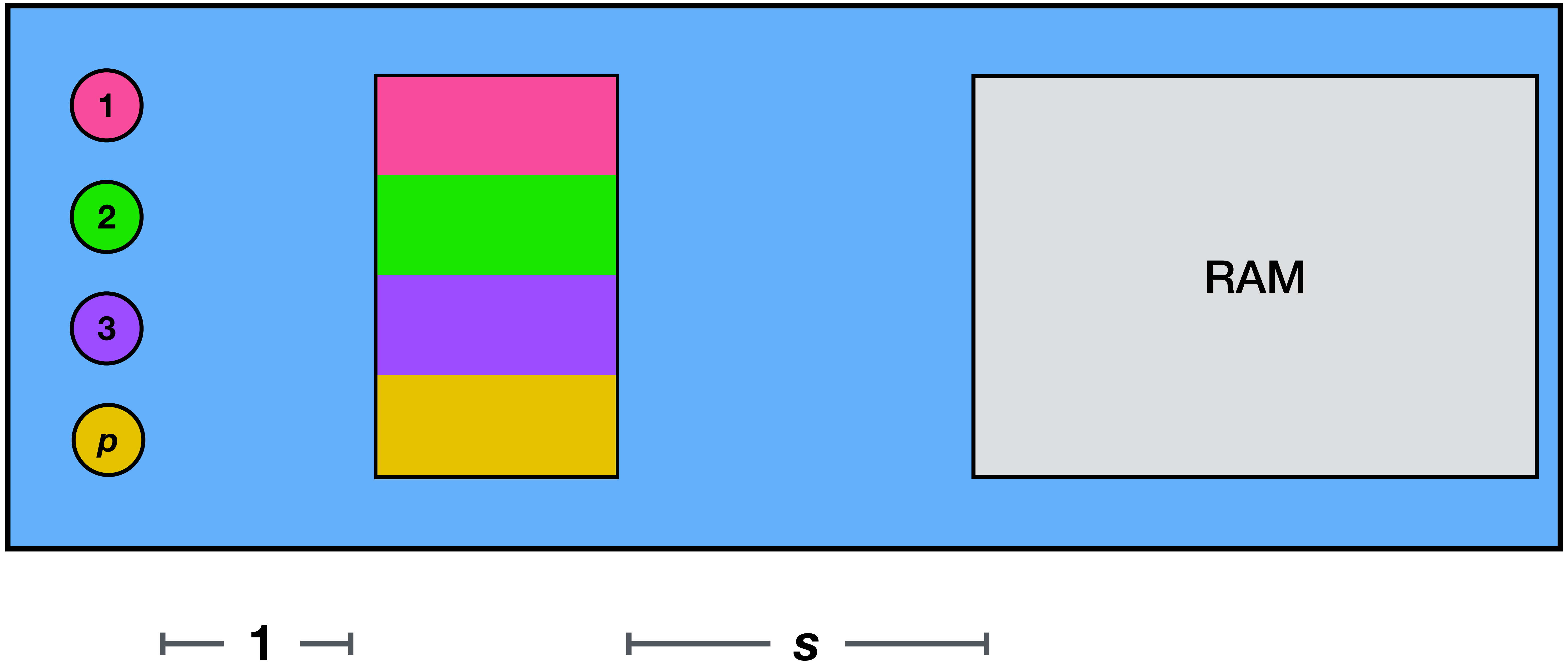
p threads
threads access (disjoint) blocks

$r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, ...$

$r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, ...$

$r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{36}, ...$

$r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, ...$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

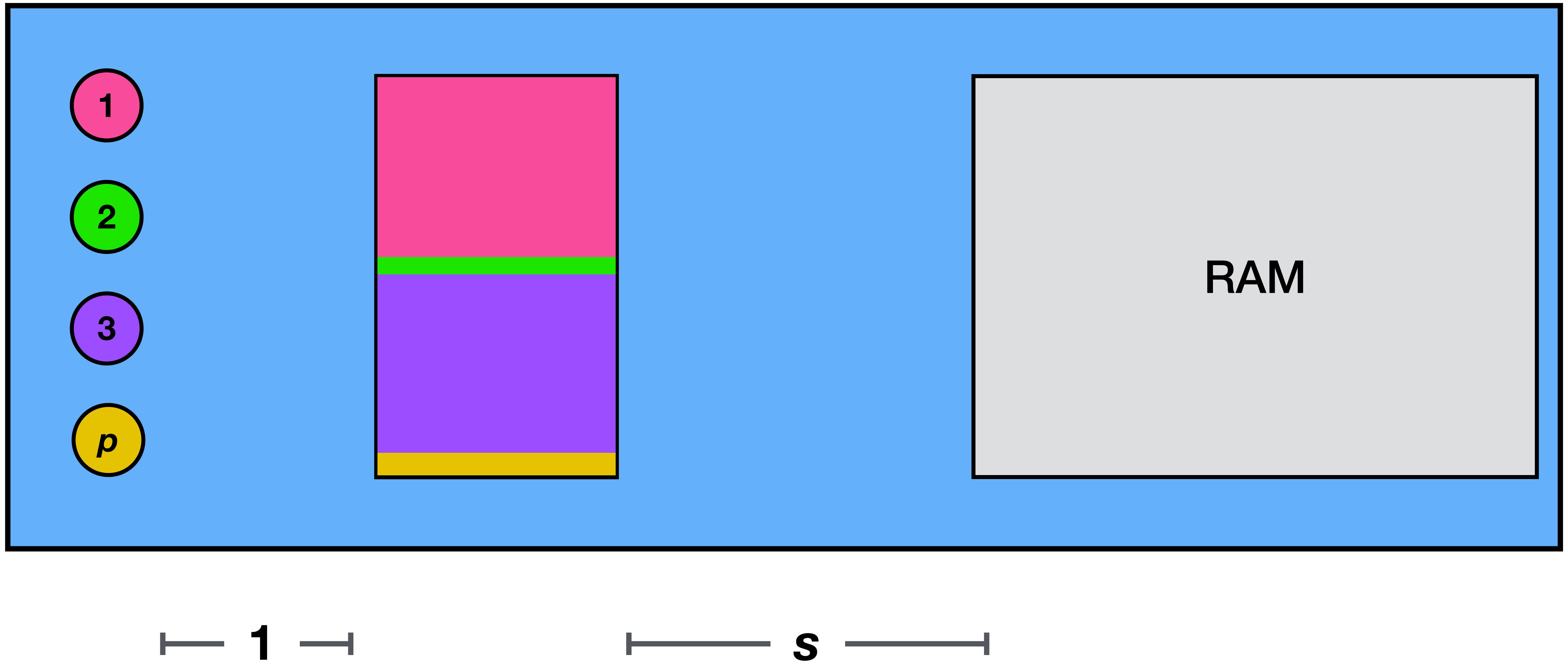
p threads
threads access (disjoint) blocks

$r_{11},\ r_{12},\ r_{13},\ r_{14},\ r_{15},\ r_{16},\ ....$

$r_{21},\ r_{22},\ r_{23},\ r_{24},\ r_{25},\ r_{26},\ ....$

$r_{31},\ r_{32},\ r_{33},\ r_{34},\ r_{35},\ r_{36},\ ....$

$r_{p1},\ r_{p2},\ r_{p3},\ r_{p4},\ r_{p5},\ r_{p6},\ ....$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

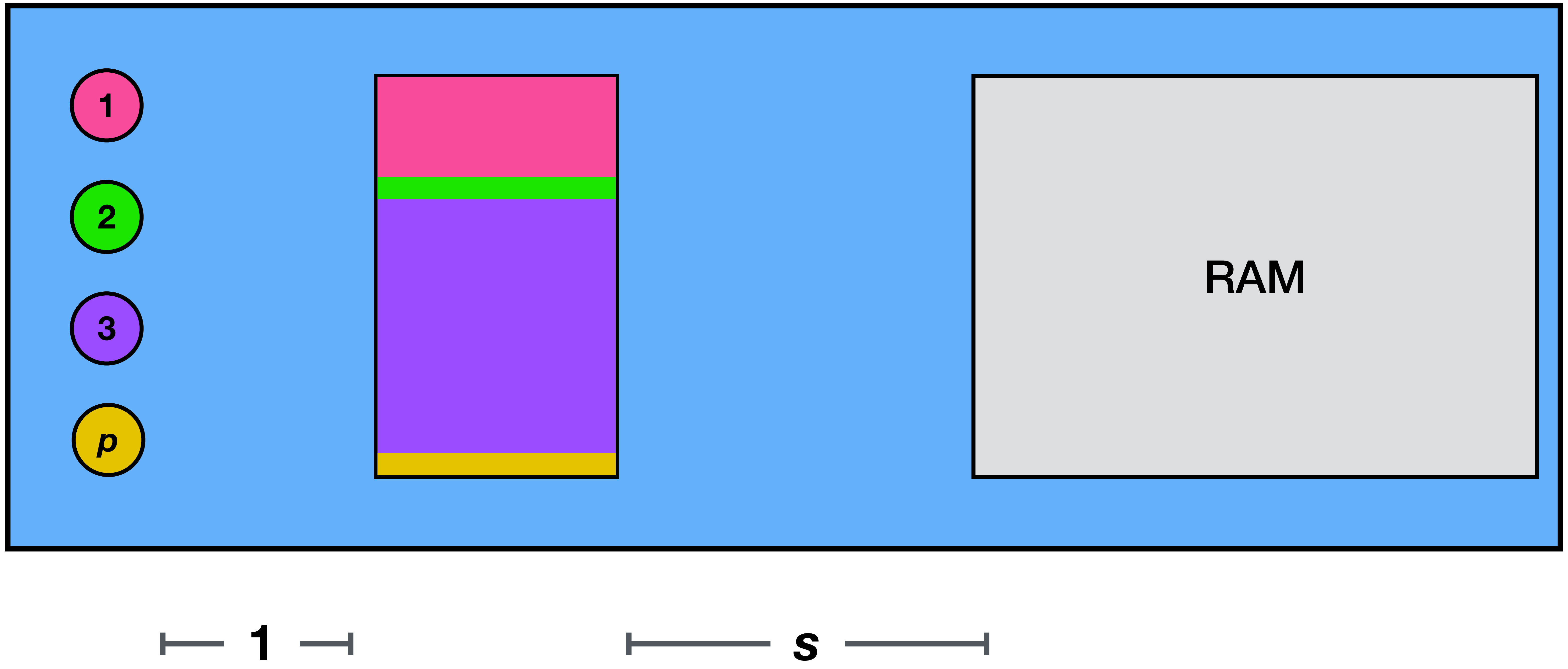
p threads
threads access (disjoint) blocks

$r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, \ldots.$

$r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, \ldots.$

$r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{36}, \ldots.$

$r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, \ldots.$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

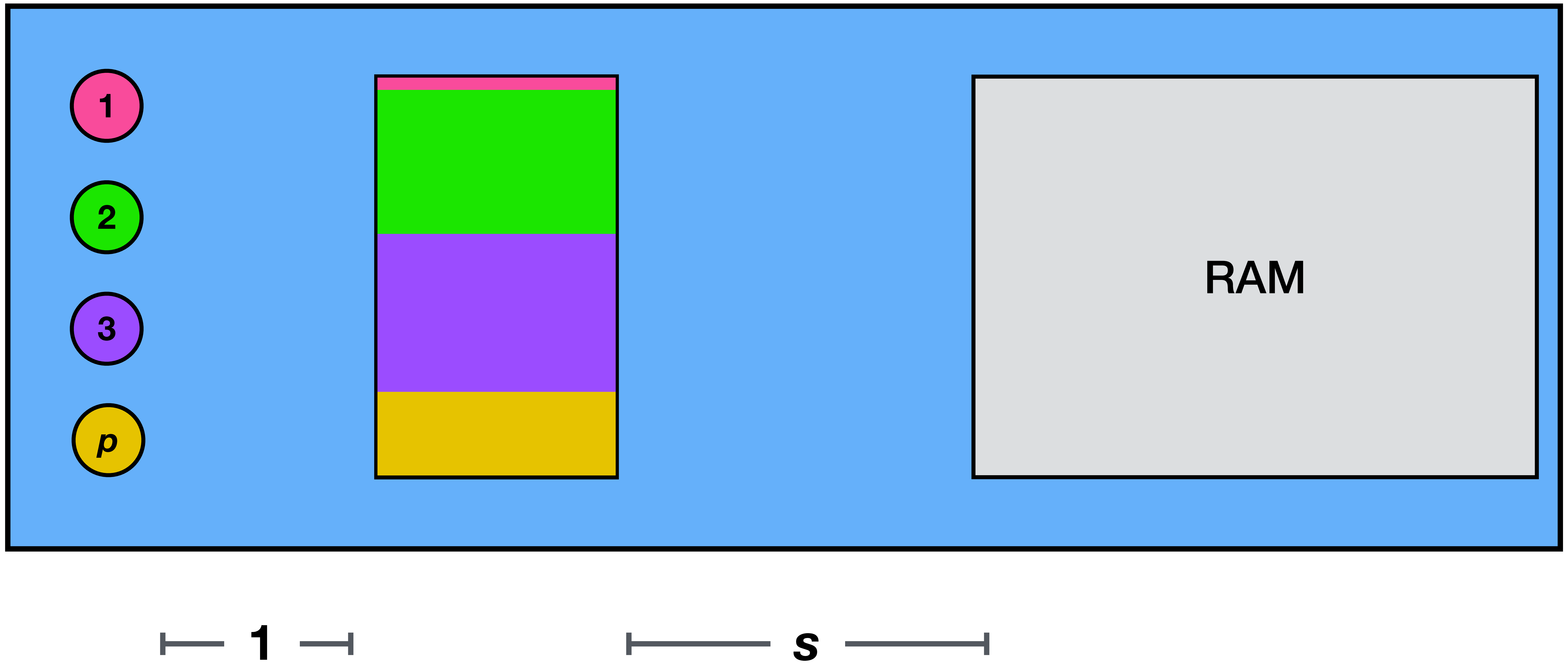
p threads
threads access (disjoint) blocks

$r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, \ldots$

$r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, \ldots$

$r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{36}, \ldots$

$r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, \ldots$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

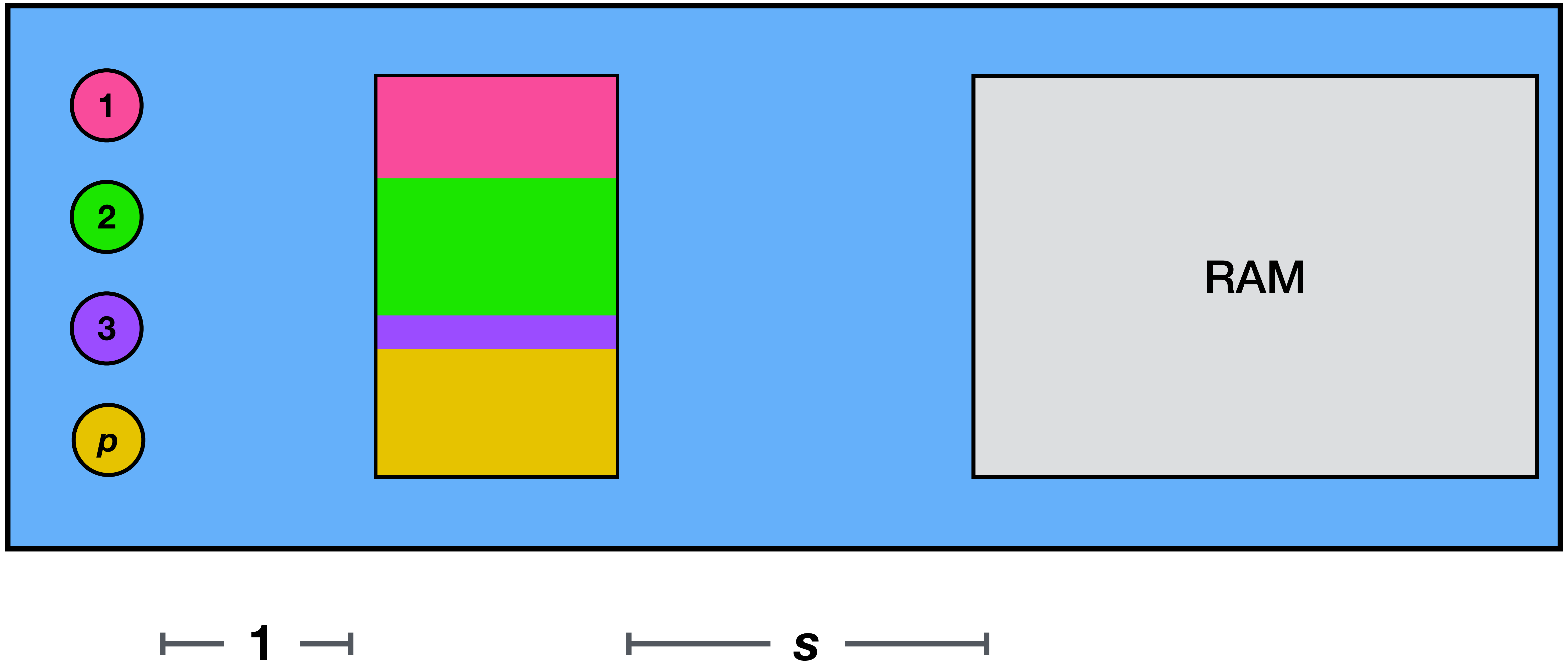
p threads
threads access (disjoint) blocks

$r_{11},\ r_{12},\ r_{13},\ r_{14},\ r_{15},\ r_{16},\ \ldots.$

$r_{21},\ r_{22},\ r_{23},\ r_{24},\ r_{25},\ r_{26},\ \ldots.$

$r_{31},\ r_{32},\ r_{33},\ r_{34},\ r_{35},\ r_{36},\ \ldots.$

$r_{p1},\ r_{p2},\ r_{p3},\ r_{p4},\ r_{p5},\ r_{p6},\ \ldots.$



Dividing the cache evenly among the threads is bad: $\Omega(s \bullet \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

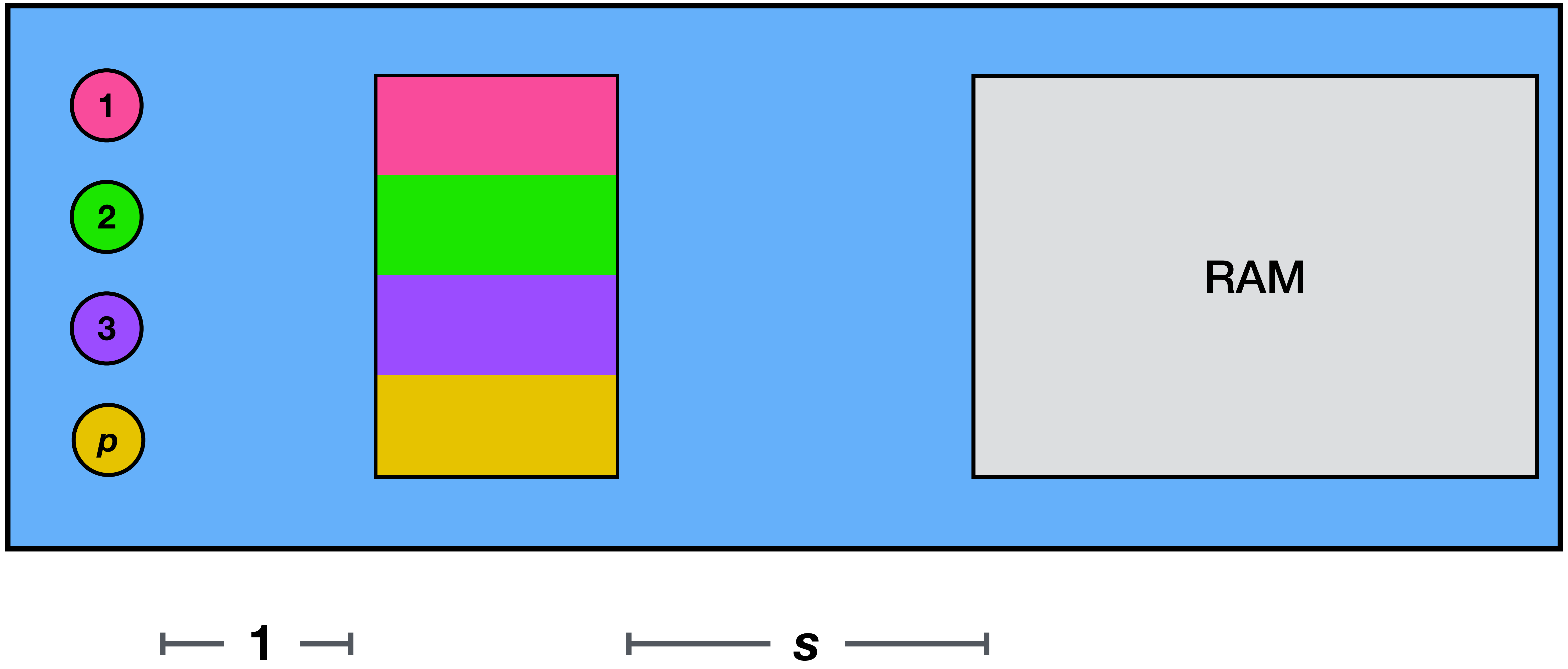
p threads
threads access (disjoint) blocks

r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, ...

r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, ...

r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{36}, ...

r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, ...



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

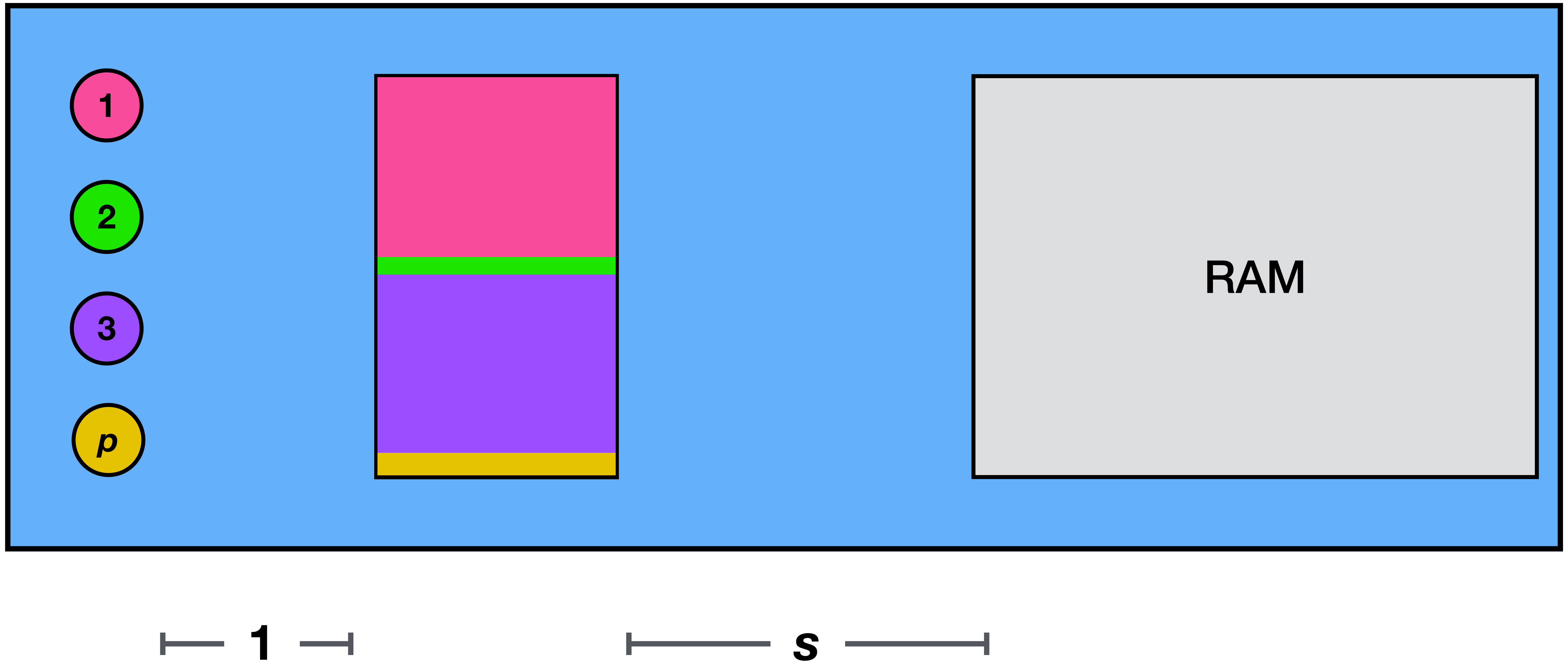
p threads
threads access (disjoint) blocks

$r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, \ldots.$

$r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, \ldots.$

$r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{36}, \ldots.$

$r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, \ldots.$



Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \text{OPT})$.

Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 1: how to partition the cache among the threads?

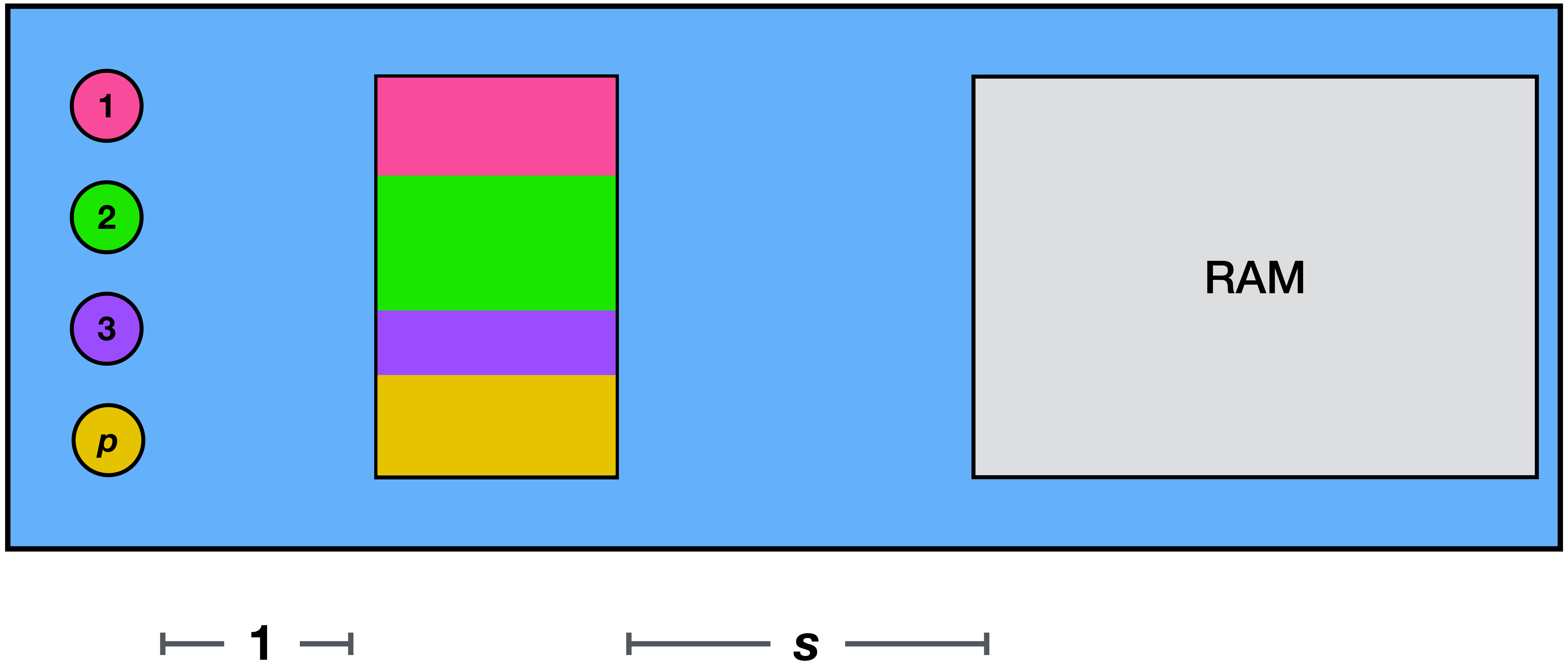
p threads
threads access (disjoint) blocks

$r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, \ldots$

$r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}, \ldots$

$r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{36}, \ldots$

$r_{p1}, r_{p2}, r_{p3}, r_{p4}, r_{p5}, r_{p6}, \ldots$



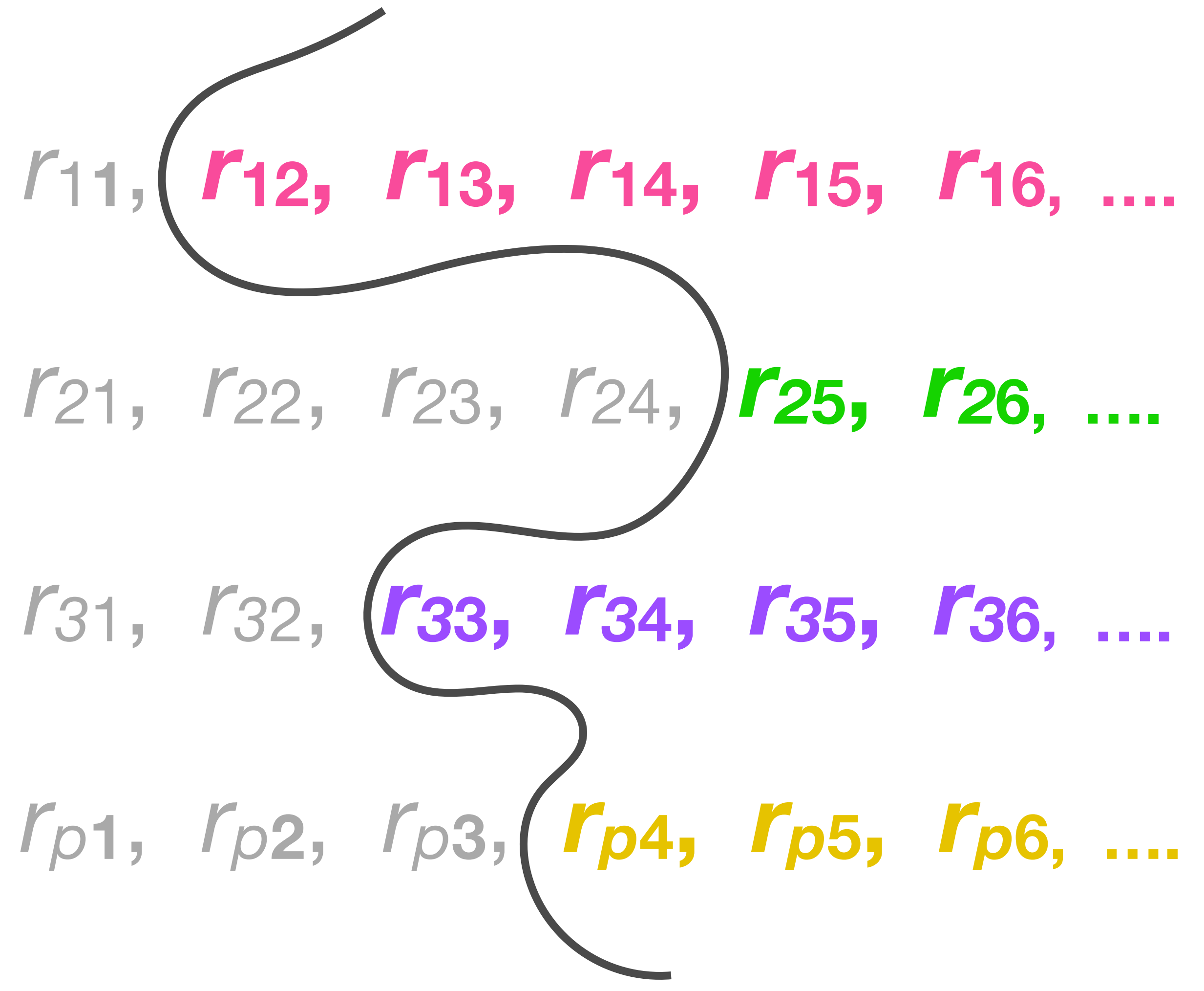
Dividing the cache evenly among the threads is bad: $\Omega(s \cdot \mathrm{OPT})$.

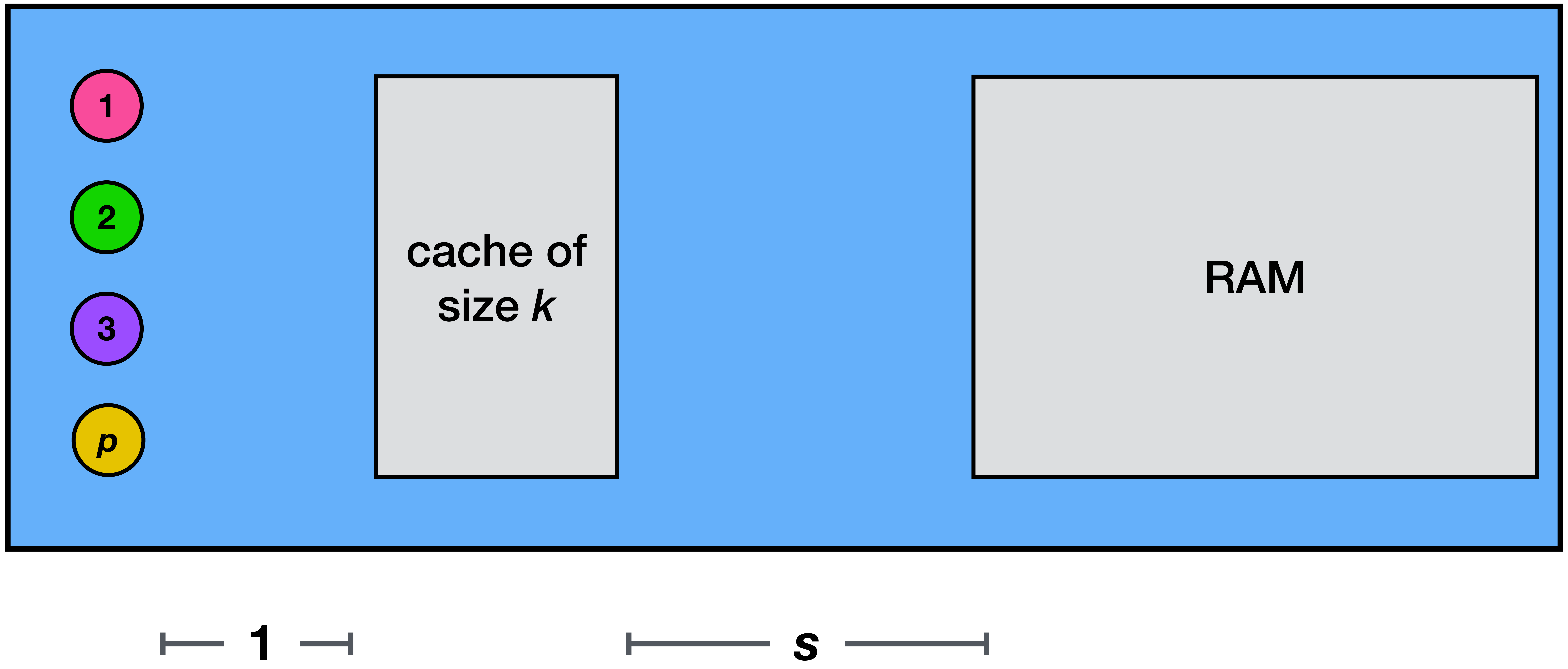
Any fixed allocation is similarly bad.

Challenge: how to *dynamically* partition the cache among the threads?

Challenge 2: how to interleave/schedule the individual threads?

p threads
threads access (disjoint) blocks

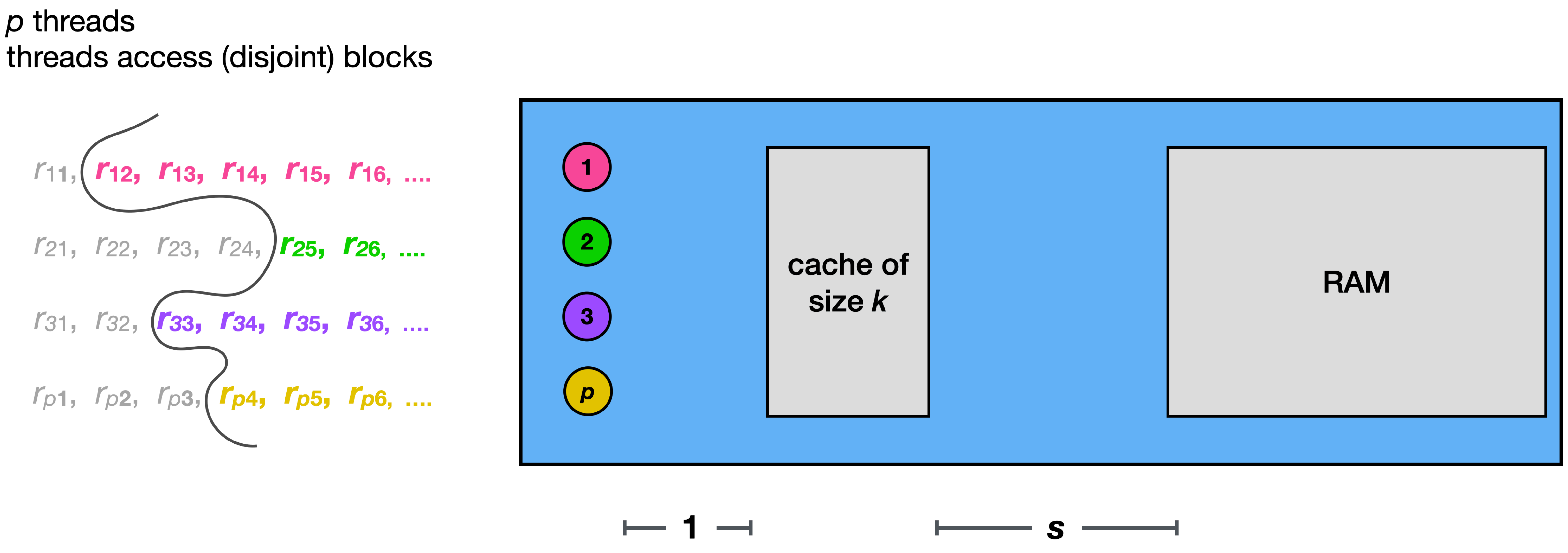




Our decisions cause processors to move at different speeds.

In general, threads cannot be scheduled in lock-step.

Challenge 2: how to interleave/schedule the individual threads?



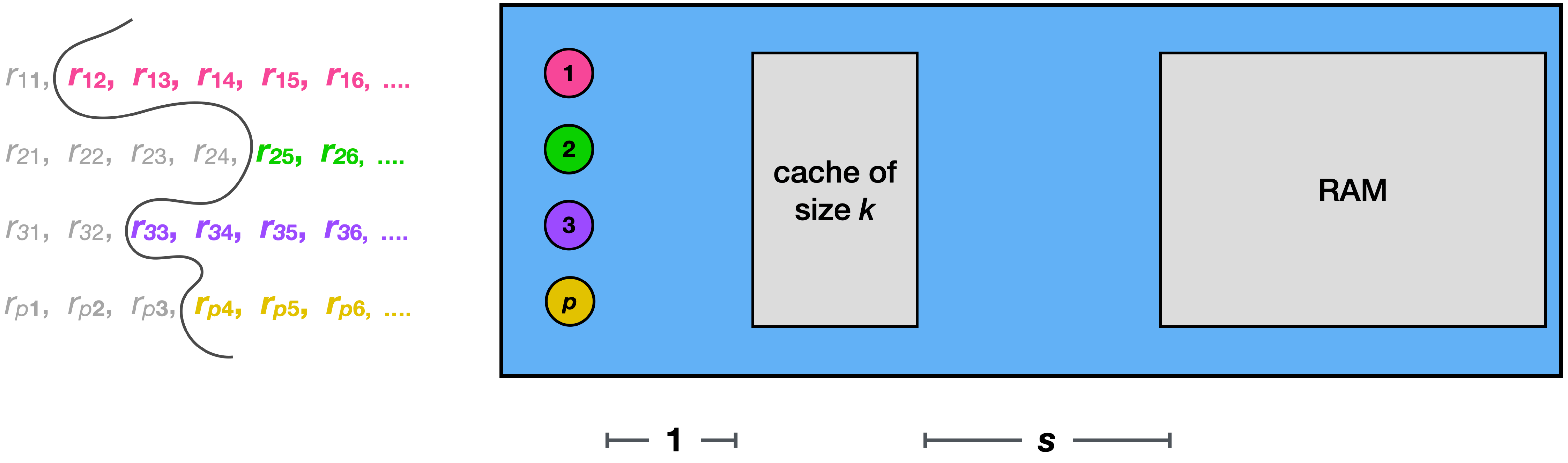
In general, threads cannot be scheduled in lock-step.

Ex:

- p_1 accesses pages (round robin) $\in [0, k/2)$
- p_2 " $\in [k/2, k)$
- p_3 " $\in [k, 3k/2)$
- p_4 " $\in [3k/2, 2k)$

Challenge 2: how to interleave/schedule the individual threads?

p threads
threads access (disjoint) blocks



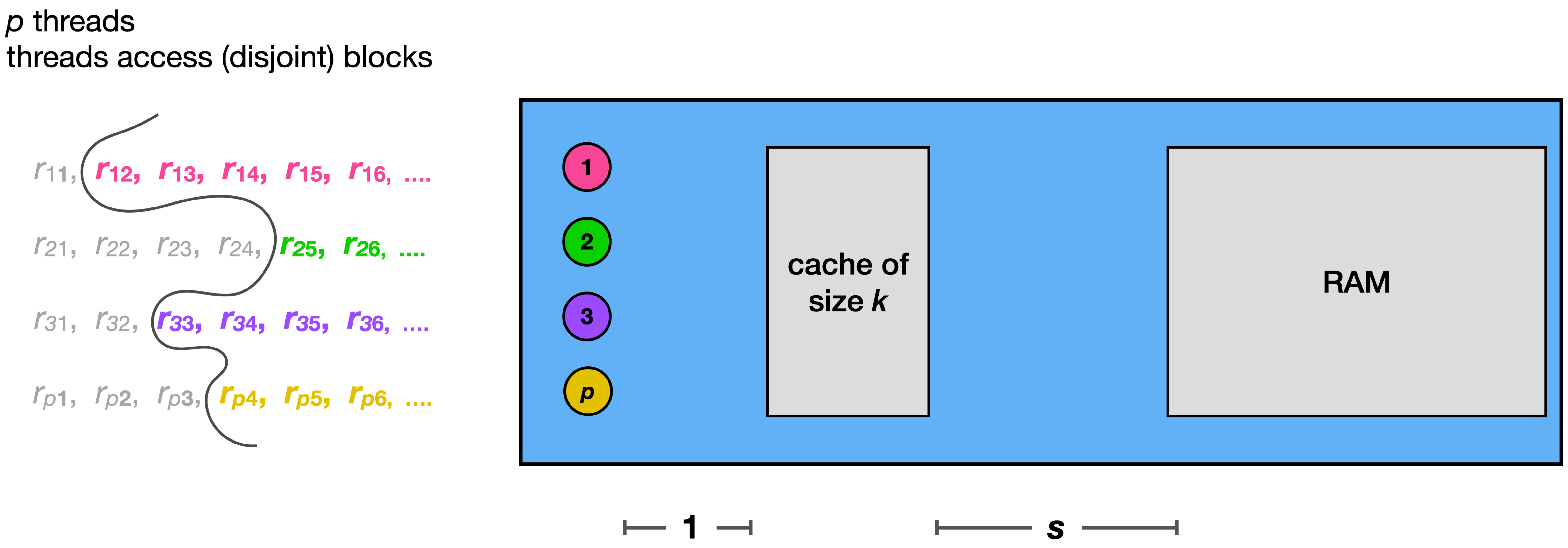
In general, threads cannot be scheduled in lock-step.

Ex:

- p_1 accesses pages (round robin) $\in [0, k/2)$
- p_2 " $\in [k/2, k)$
- p_3 " $\in [k, 3k/2)$
- p_4 " $\in [3k/2, 2k)$

working sets of 2 threads fit in cache, but not 3

Challenge 2: how to interleave/schedule the individual threads?



In general, threads cannot be scheduled in lock-step.

Ex:

- p_1 accesses pages (round robin) $\in [0, k/2)$
- p_2 " $\in [k/2, k)$
- p_3 " $\in [k, 3k/2)$
- p_4 " $\in [3k/2, 2k)$

working sets of 2 threads fit in cache, but not 3

OPT:

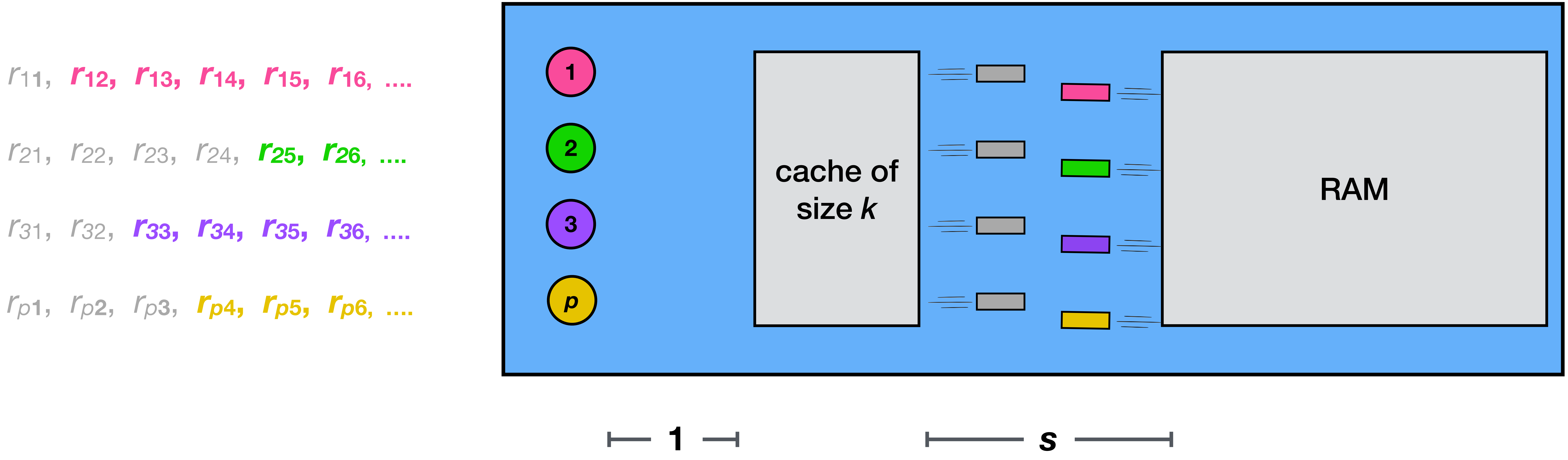
- Run p_1 and p_2 to completion.
- Then run p_3 and p_4 to completion.

Lock step:

- Makespan is $\Omega(s \cdot \mathrm{OPT})$.

Non-Challenge: What Eviction Policy Should Each Processor Use?

p threads
threads access (disjoint) blocks



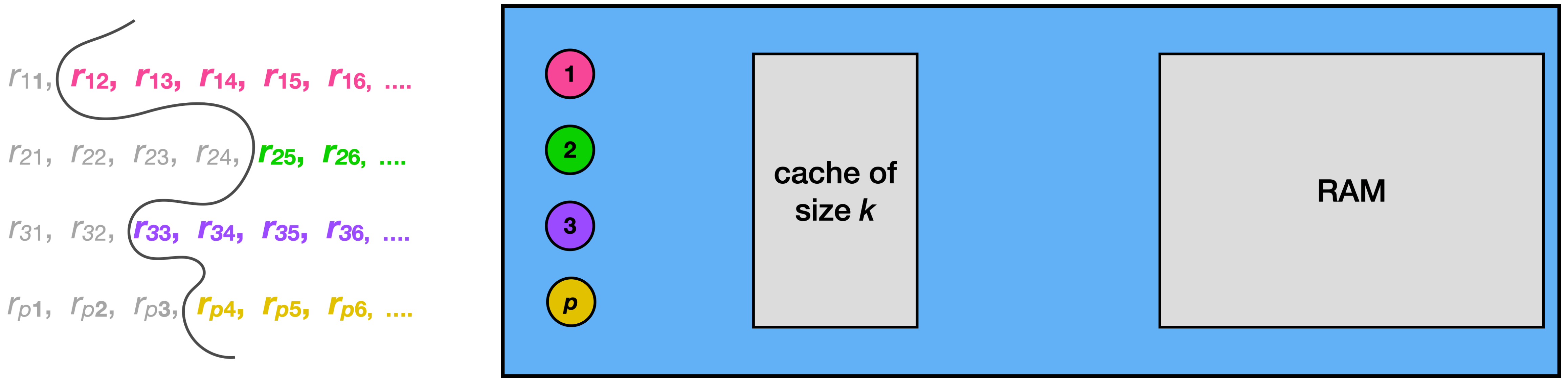
Question: What are the eviction policies of individual threads, given that a thread's allotment of cache changes over time?

Answer: Each processor should still just use LRU.

[Bender, Ebrahimi, Fineman, Ghasemiesfeh, Johnson, McCauley SODA 2014]

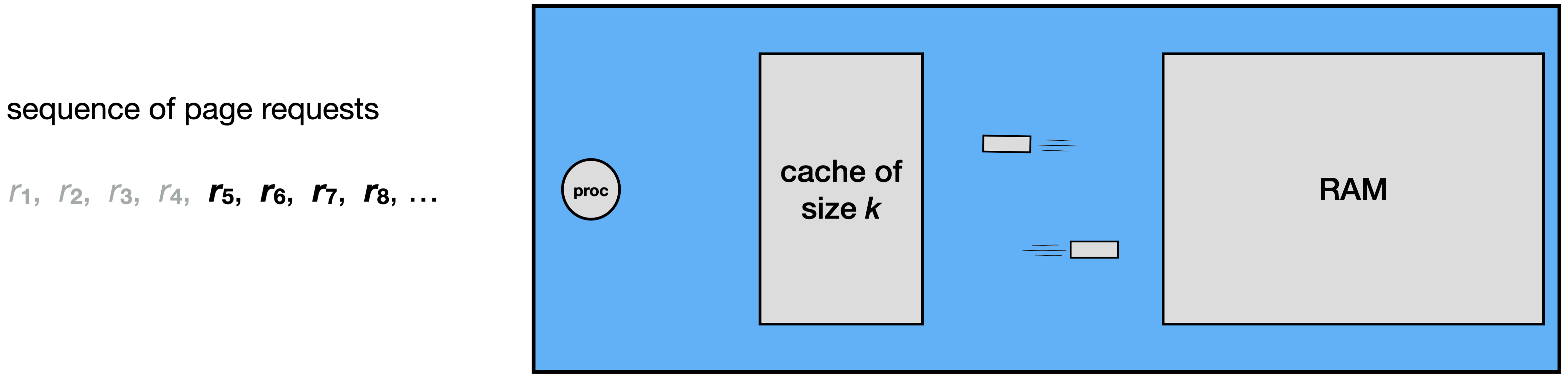
Summary of parallel-paging challenges

Parallel paging



- How to partition the cache among the threads?
- How to interleave/schedule the individual threads?
- What are the eviction policies of each individual threads?

Sequential paging



- The (single) thread gets all of cache.
- There is a total order in which all page requests are serviced.
- There is a single eviction policy.

Online parallel paging was open for 25 years

Previous results are for the offline problem.

[Hassidim ICS 2010]
[López-Ortiz & Salinger ITCS 2012
& WAOA 2012]

- NP-hardness
- Existing ***offline*** algs w/ run times exponential in # procs *p* and cache size *k*

The online problem has been open since 1995.

[Fiat and Karlin, STOC 1995]

This Talk: O(log *p*)-competitive algs for parallel paging

Deterministic online parallel paging algorithm that is

- O(log *p*) competitive for *average completion time + makespan* with
- O(1) resource augmentation.

No deterministic online algorithm can do better.

[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SODA 2021]

[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SPAA 2022]

The cache is a scarce resource.

Each thread should use as little cache as it can.

Cache as a scarce resource

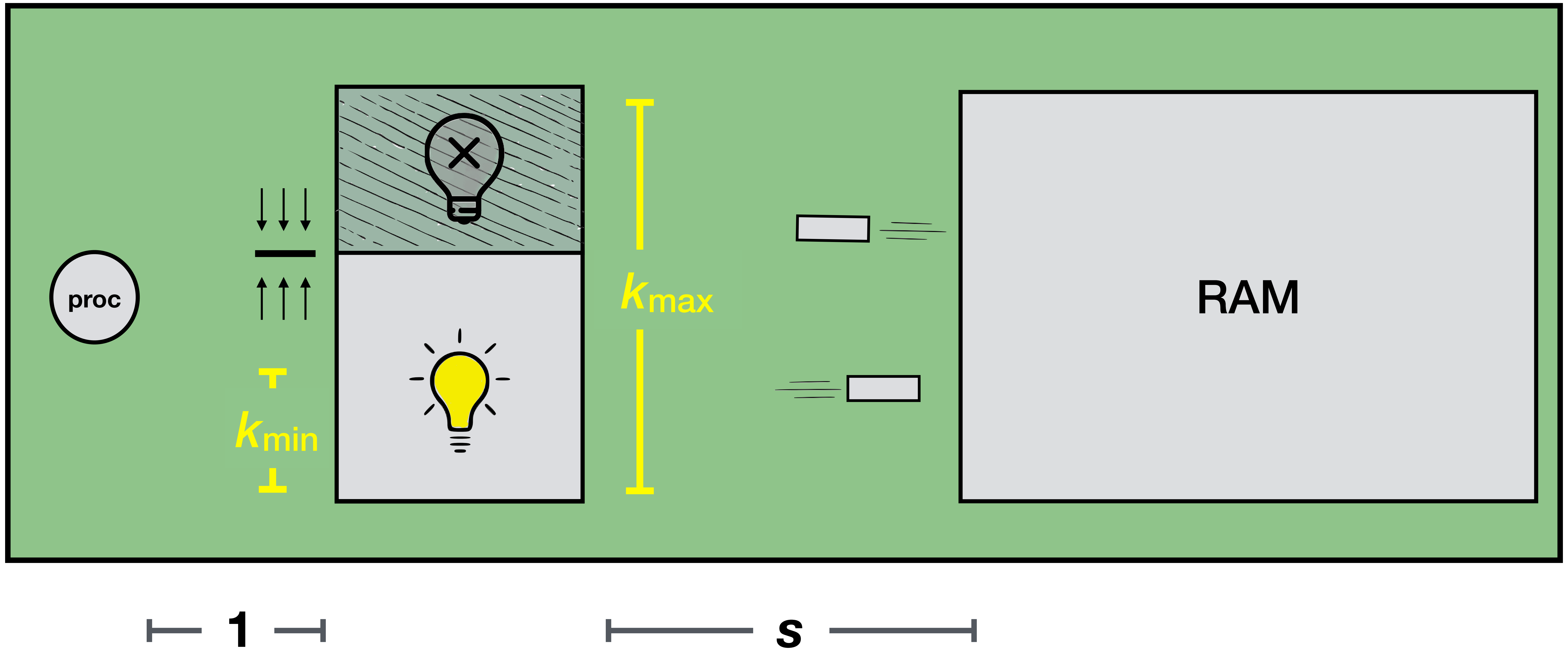
The cache is a scarce resource.

Each thread should use as little cache as it can.

This motivates the very different problem of green paging.

The Green Paging Problem

r_1, r_2, r_3, $\boldsymbol{r_4}$, $\boldsymbol{r_5}$, $\boldsymbol{r_6}$, $\boldsymbol{r_7}$, $\boldsymbol{r_8}$, . . .



•Single processor.

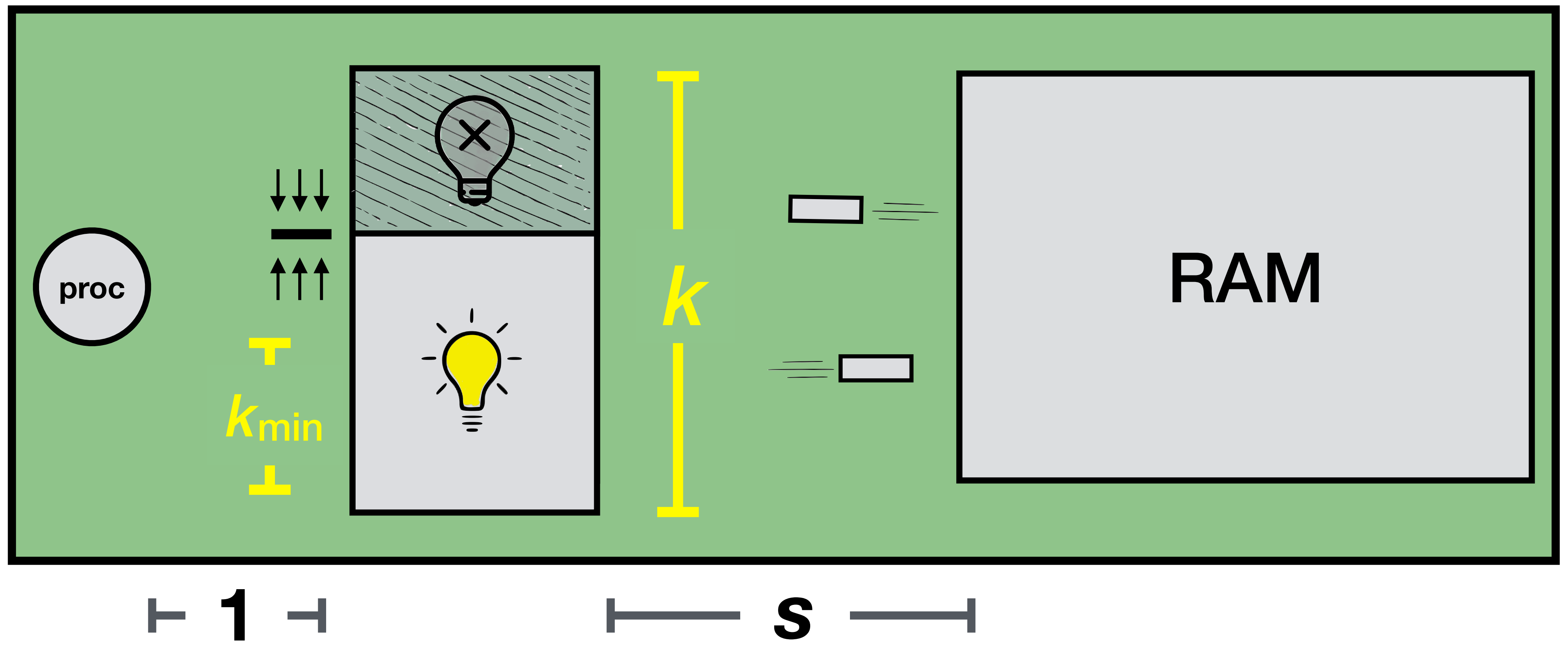
•Slots in the cache can be ***powered off*** to save energy.

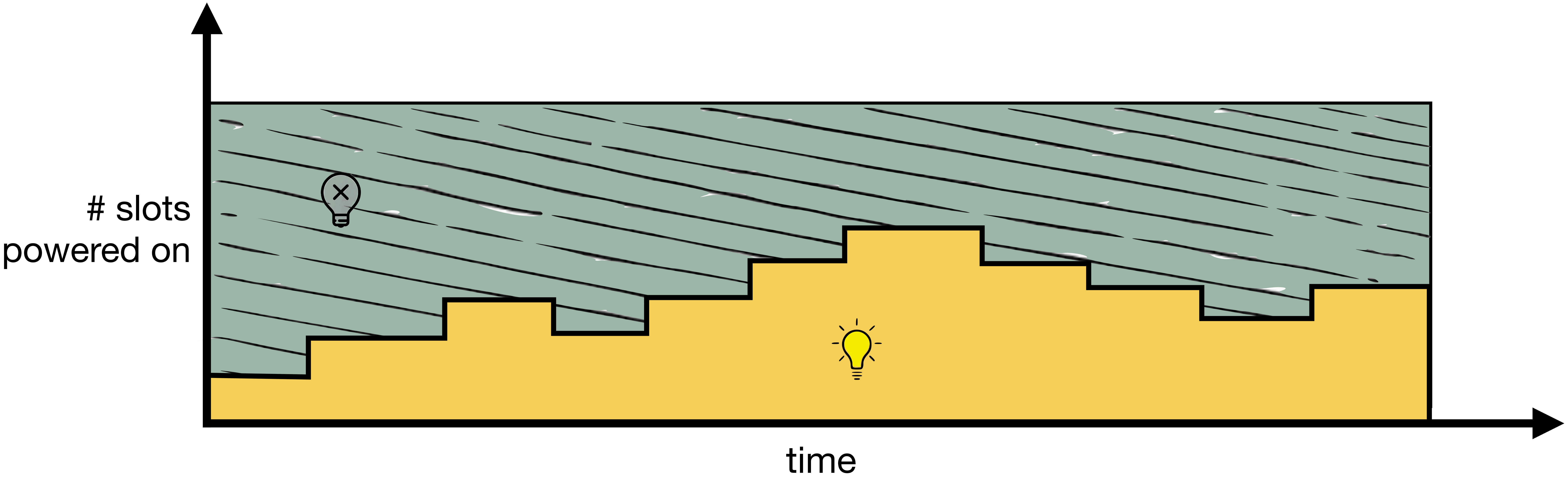
•Energy consumption in a timestep = Θ(cache slots that are turned on).

Objective: service a request sequence online with minimal energy.

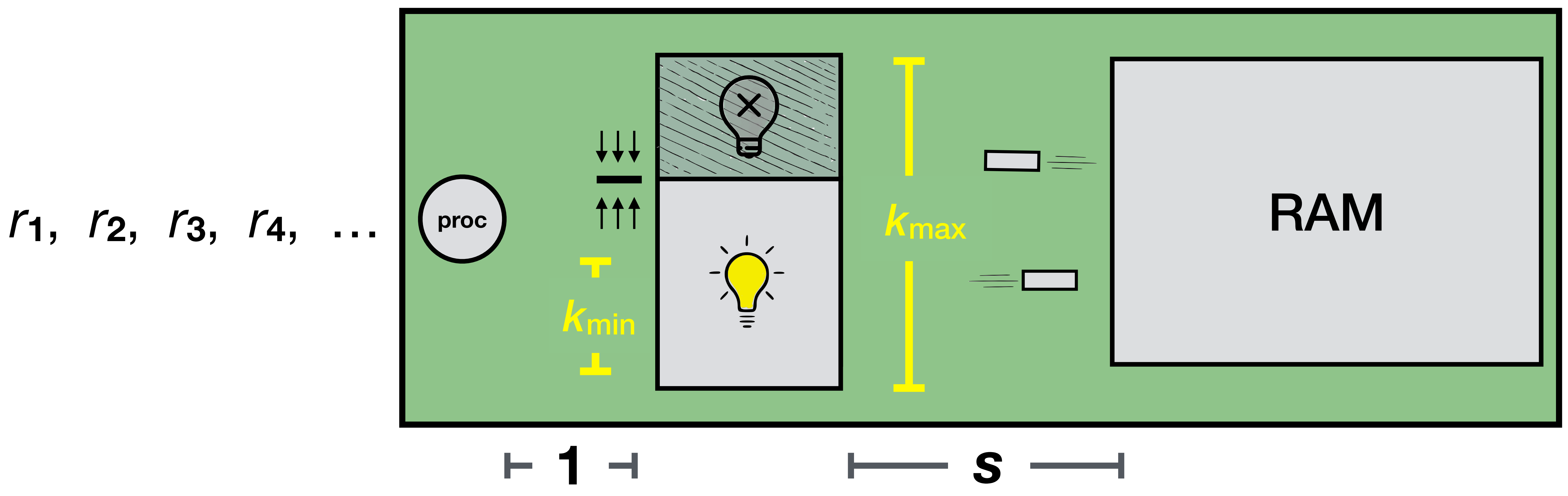
cache impact ("energy") = $\Theta(\sum_{\text{time } t}$ # cache slots powered on $)$

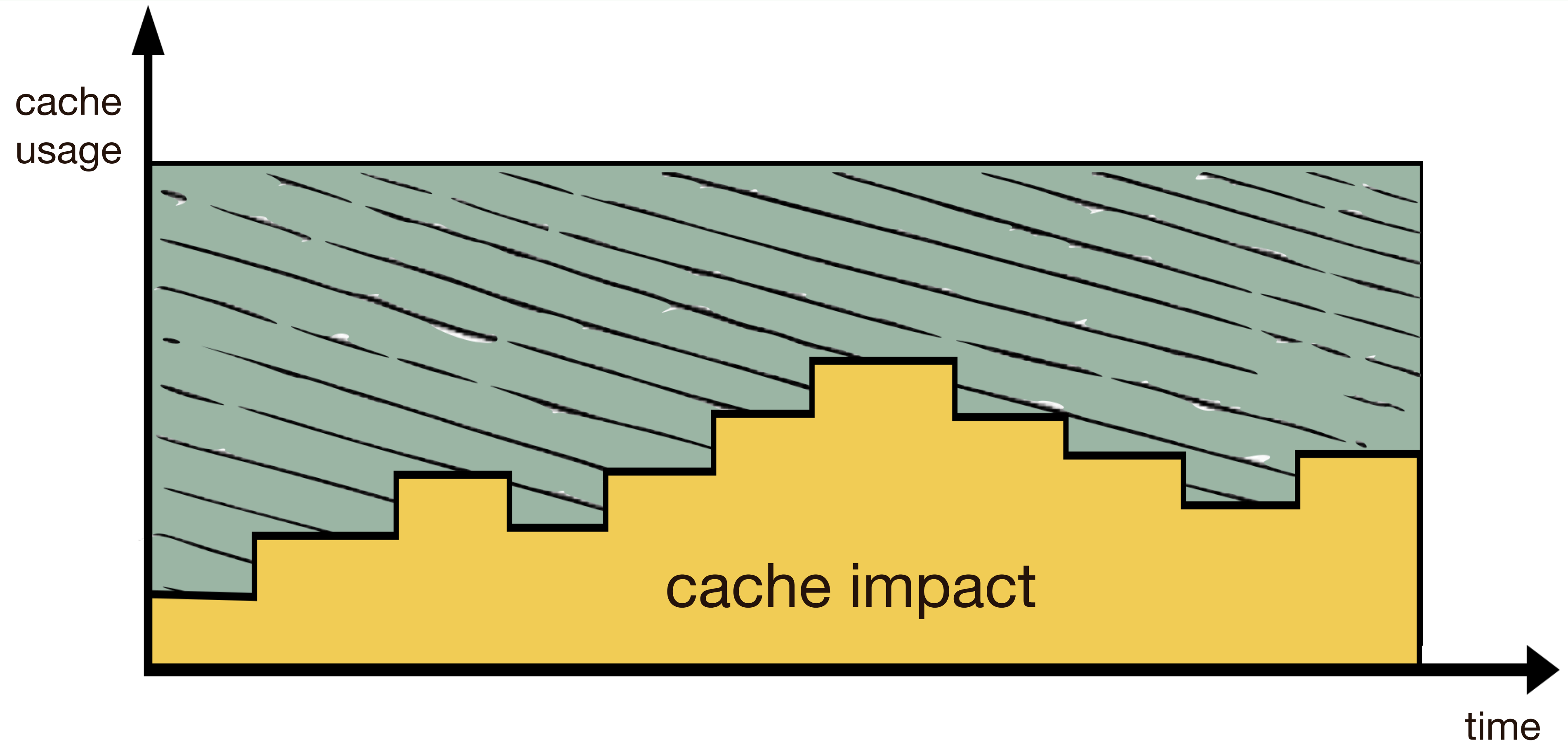
$\boldsymbol{R} = r_1,\ r_2,\ r_3,\ r_4,\ \ldots$





Green paging





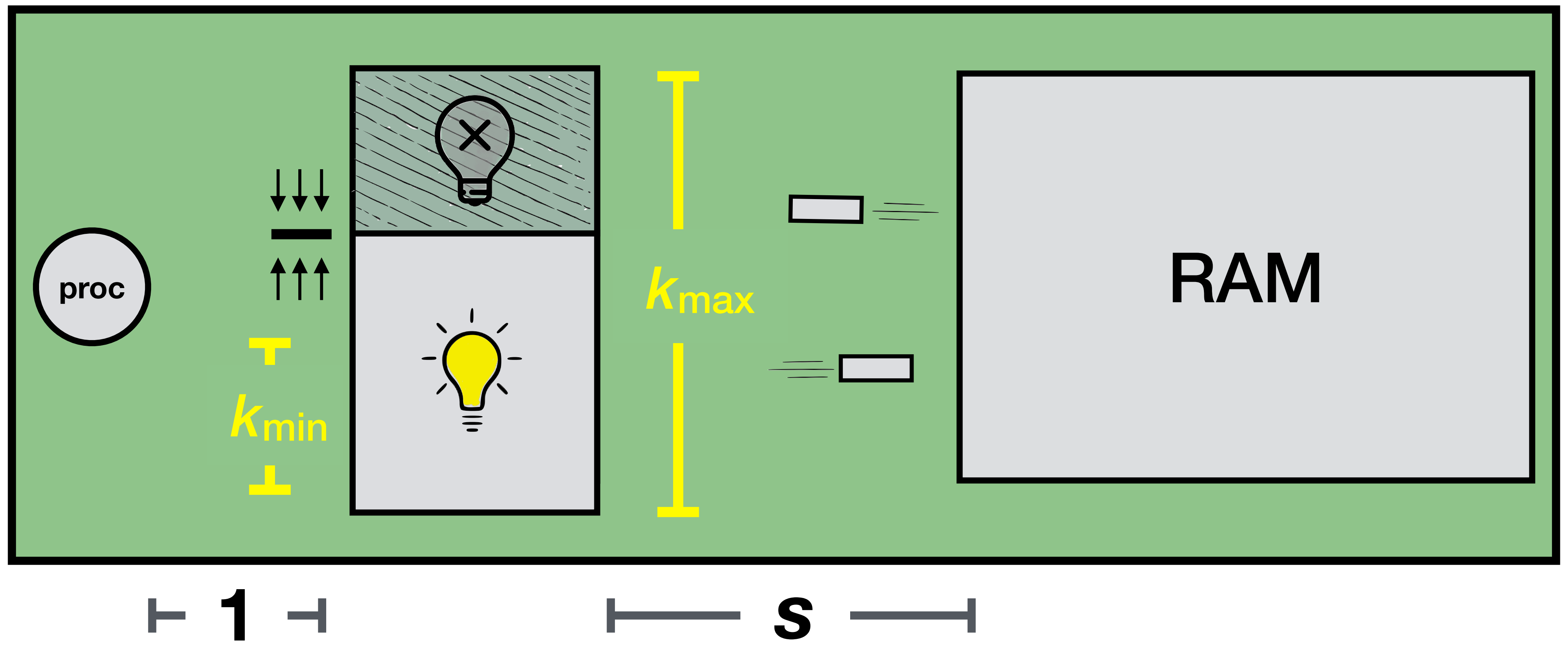
Given a page request sequence $R = r_1, r_2, r_3, \ldots$ design

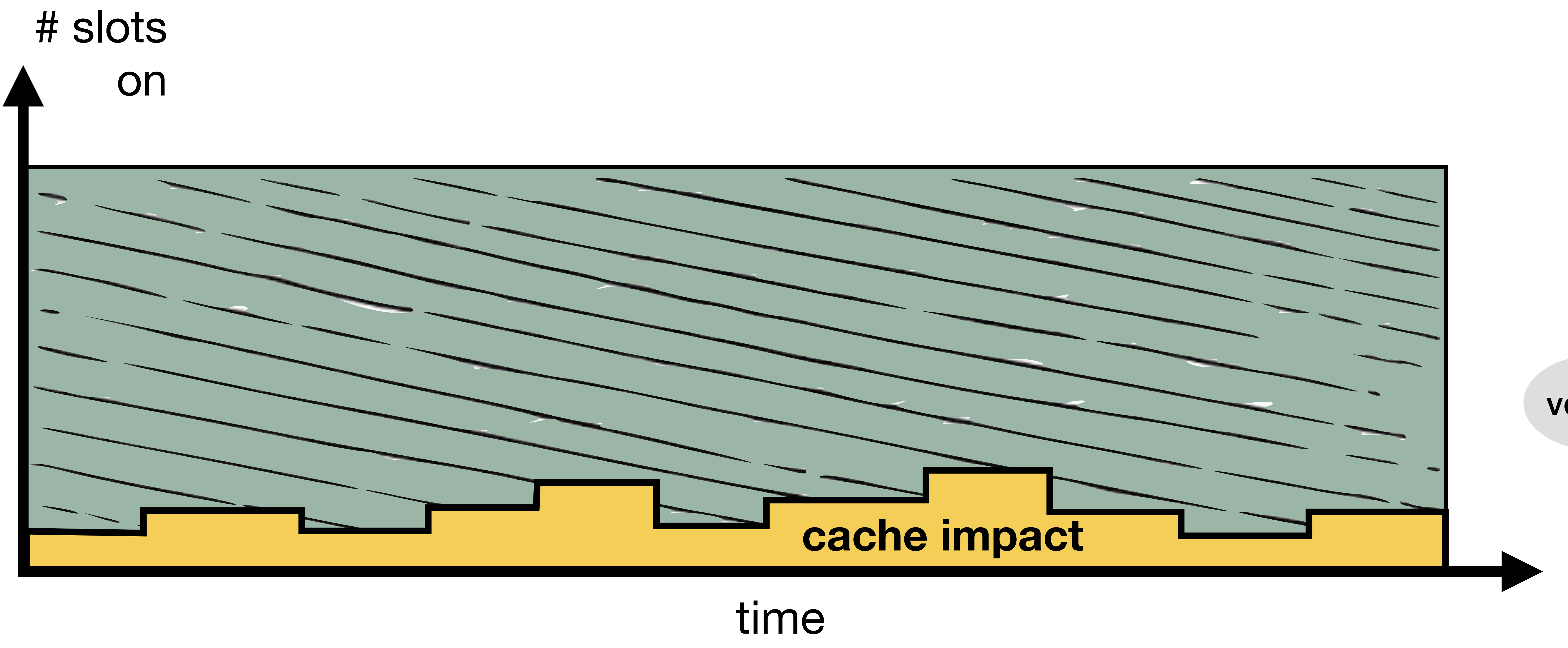
- **cache-space allocation over time, and**
- **page-replacement policy,**

to minimize the cache impact to serve R.

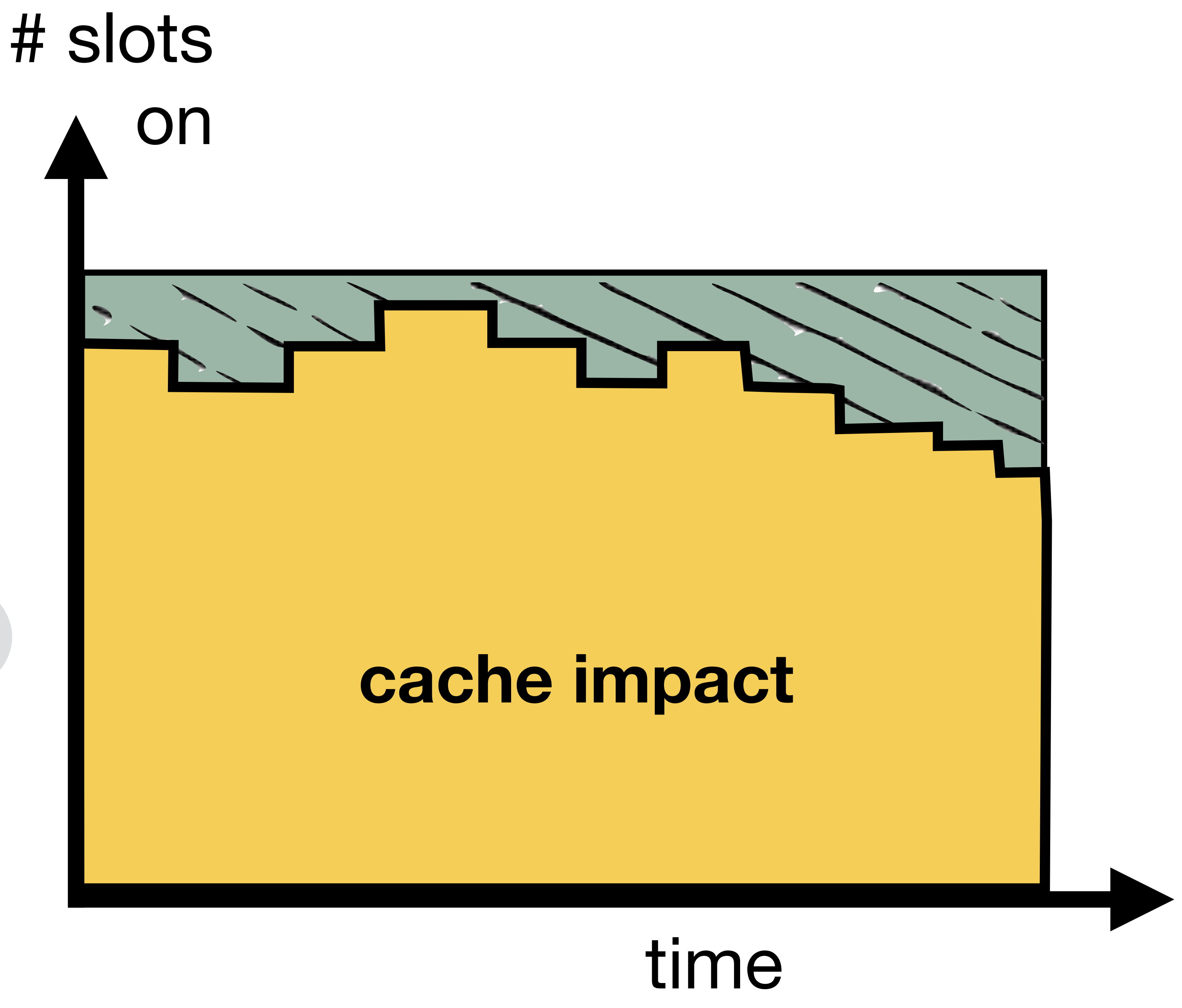
Green paging dilemma

$\boldsymbol{R} = r_1,\ r_2,\ r_3,\ r_4,\ \ldots$





versus



?

Online green paging ⟷ Online parallel paging

[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SODA 2021, SPAA 2022]

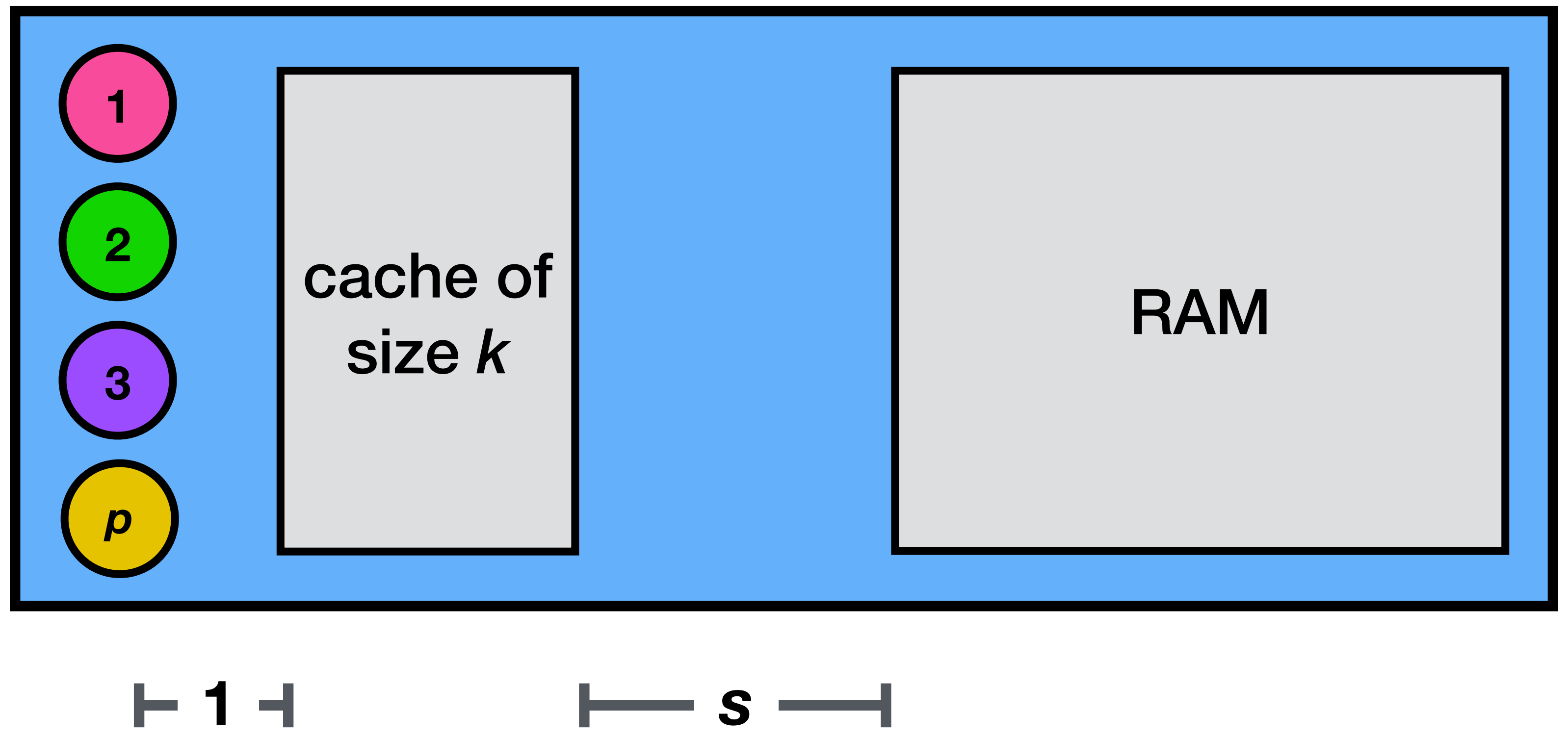
Solve the green-paging problem independently for each thread

$R_1 = r_{11}, r_{12}, r_{13}, r_{14}, \ldots$

$R_2 = r_{21}, r_{22}, r_{23}, r_{24}, \ldots$

$R_3 = r_{31}, r_{32}, r_{33}, r_{34}, \ldots$

$R_p = r_{p1}, r_{p2}, r_{p3}, r_{p4}, \ldots$



Online green paging $\longleftrightarrow$ Online parallel paging

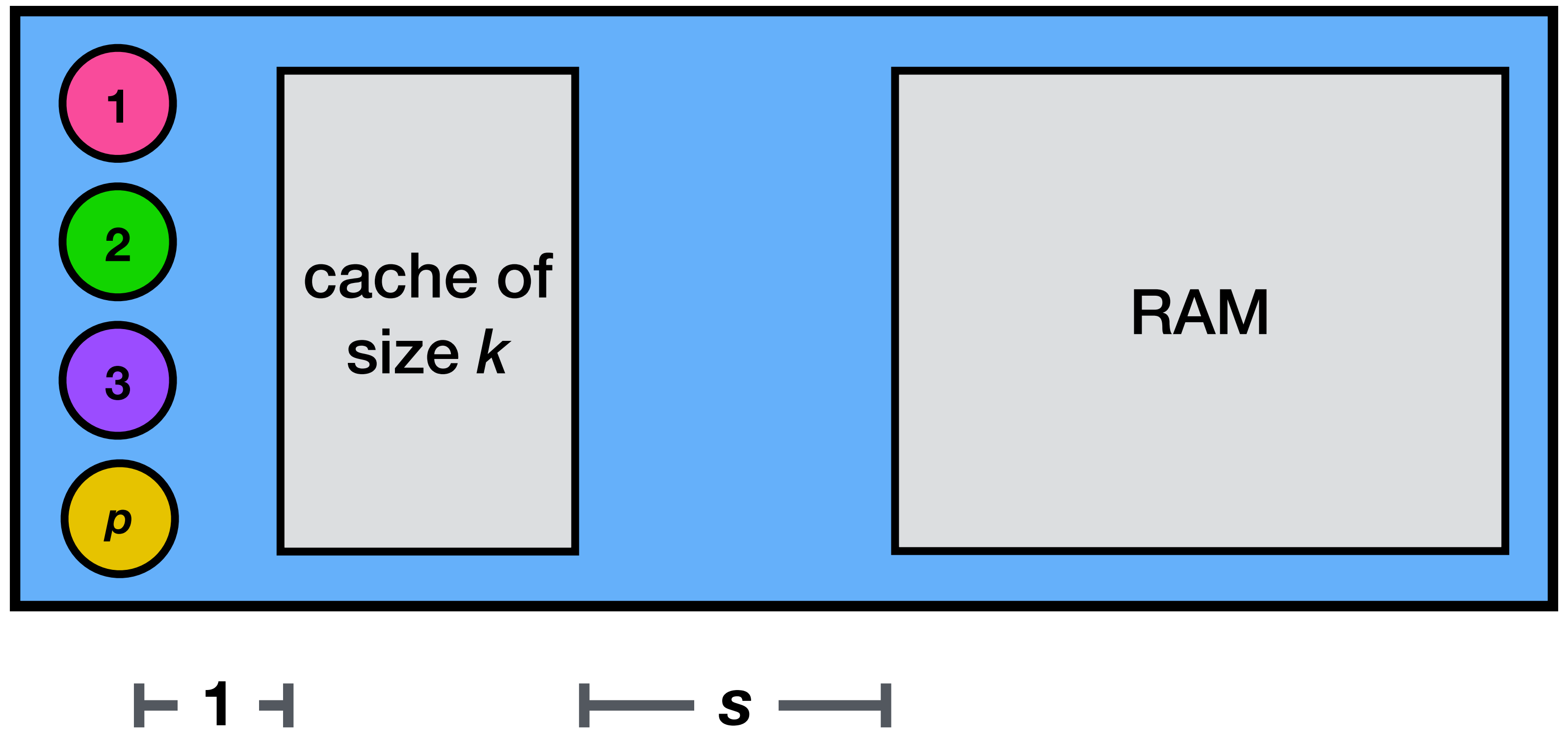
[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SODA 2021, SPAA 2022]

$R_1 = r_{11},\ r_{12},\ r_{13},\ r_{14},\ \ldots$

$R_2 = r_{21},\ r_{22},\ r_{23},\ r_{24},\ \ldots$

$R_3 = r_{31},\ r_{32},\ r_{33},\ r_{34},\ \ldots$

$R_p = r_{p1},\ r_{p2},\ r_{p3},\ r_{p4},\ \ldots$

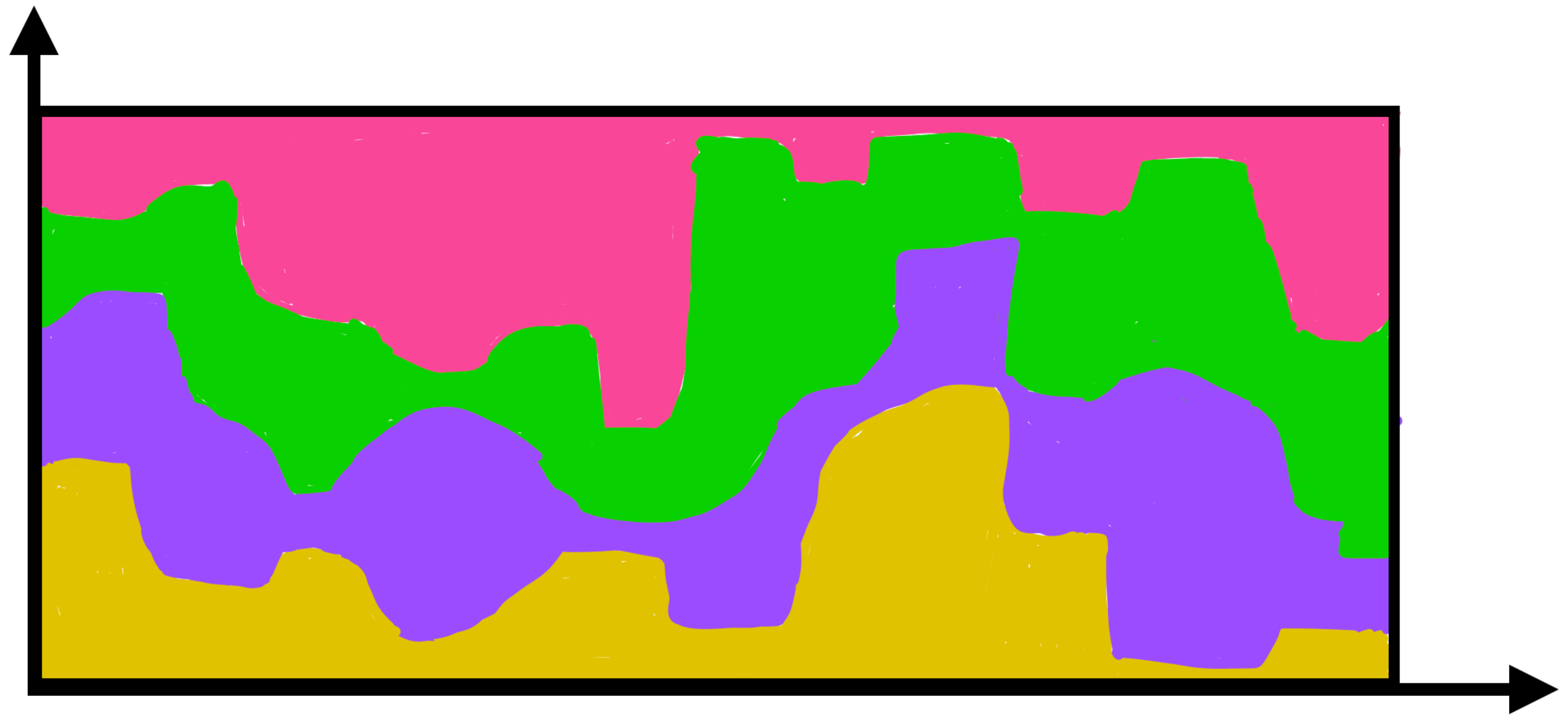


Solve the green-paging problem independently for each thread

$\Downarrow$

Stitch/pack the solutions together

$\Downarrow$

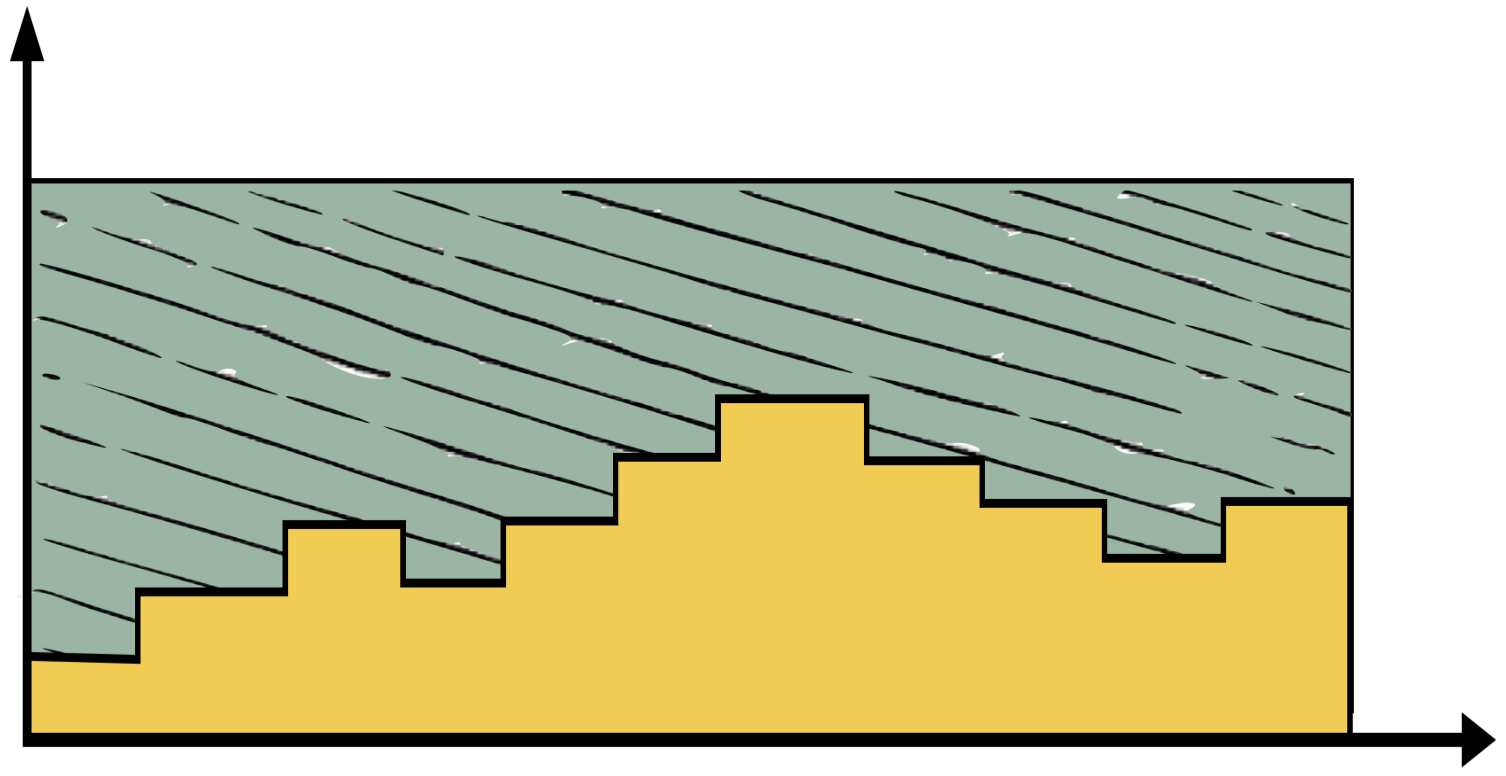


Theorem [Green paging **upper bound** → Parallel paging **upper bound**]

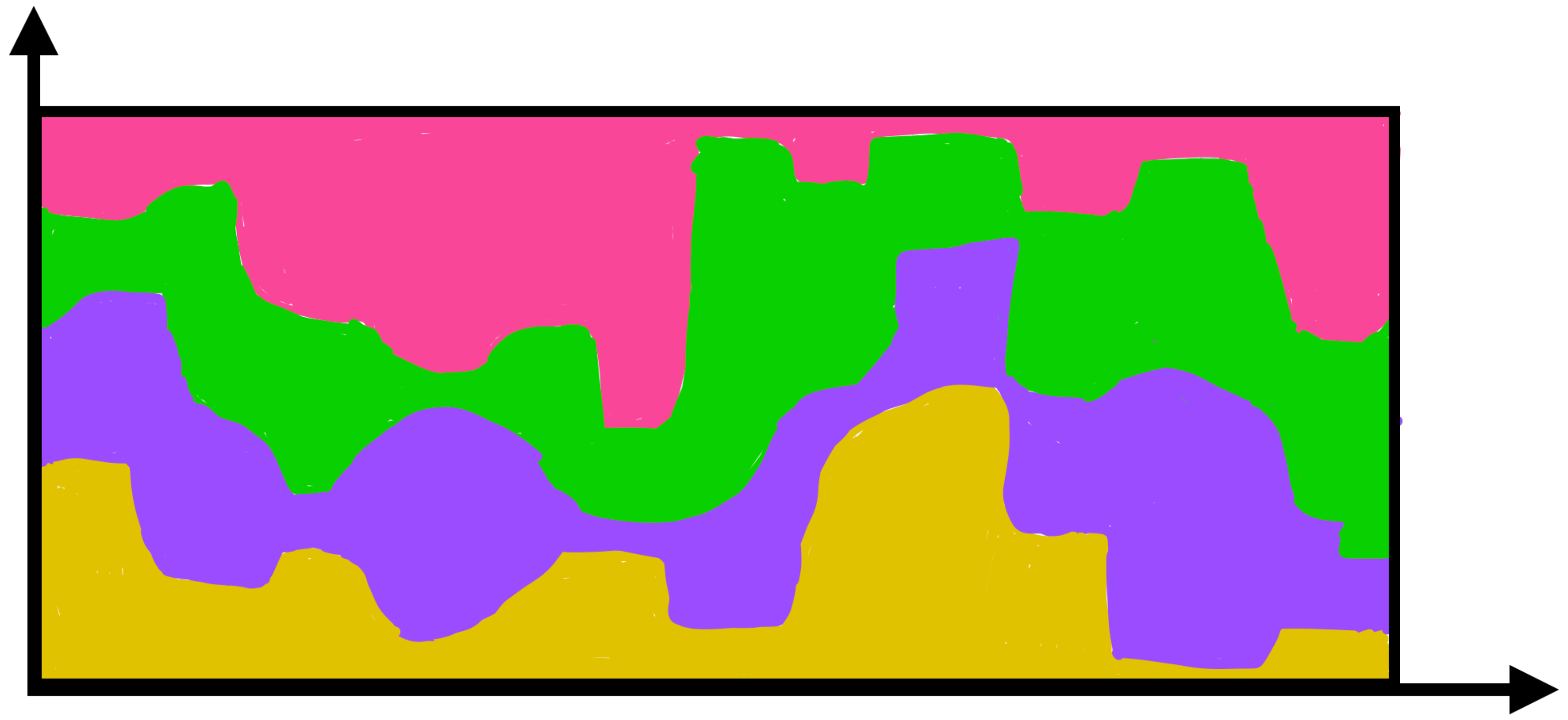
Theorem [Green paging **lower bound** → Parallel paging **lower bound**]

online green-paging alg w/ comp ratio $\Theta(\beta)$ $\Longleftrightarrow$ online ||-paging alg w/ comp ratio $\Theta(\beta)$

(with O(1) resource augmentation)



$\Longleftrightarrow$



Bounds for Green Paging and Parallel Paging

[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SODA 2021, SPAA 2022]

Theorem: No deterministic online green-paging algorithm ($k_{max}=k$ and $k_{min}=k/p$) can be o(log P)-competitive.

Bounds for Green Paging and Parallel Paging

[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SODA 2021, SPAA 2022]

Theorem: **No deterministic online green-paging algorithm ($k_{max}=k$ and $k_{min}=k/p$) can be o(log P)-competitive.**

$\Longrightarrow$ **no parallel-paging algorithm can be o(log P)-competitive.**

Bounds for Green Paging and Parallel Paging

[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SODA 2021, SPAA 2022]

Theorem: No deterministic online green-paging algorithm ($k_{max}=k$ and $k_{min}=k/p$) can be o(log P)-competitive.

$\Longrightarrow$ **no parallel-paging algorithm can be o(log P)-competitive.**

Theorem: ∃ a *universal green-paging solution* that is O(log P)-competitive for all request sequences.

Bounds for Green Paging and Parallel Paging

[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SODA 2021, SPAA 2022]

Theorem: **No deterministic online green-paging algorithm ($k_{max}=k$ and $k_{min}=k/p$) can be o(log P)-competitive.**

$\Longrightarrow$ no parallel-paging algorithm can be o(log P)-competitive.

Theorem: **∃ a *universal green-paging solution* that is O(log P)-competitive for all request sequences.**

$\Longrightarrow$ ∃ *universal* O(log P)-competitive parallel-paging algorithm.

Bounds for Green Paging and Parallel Paging

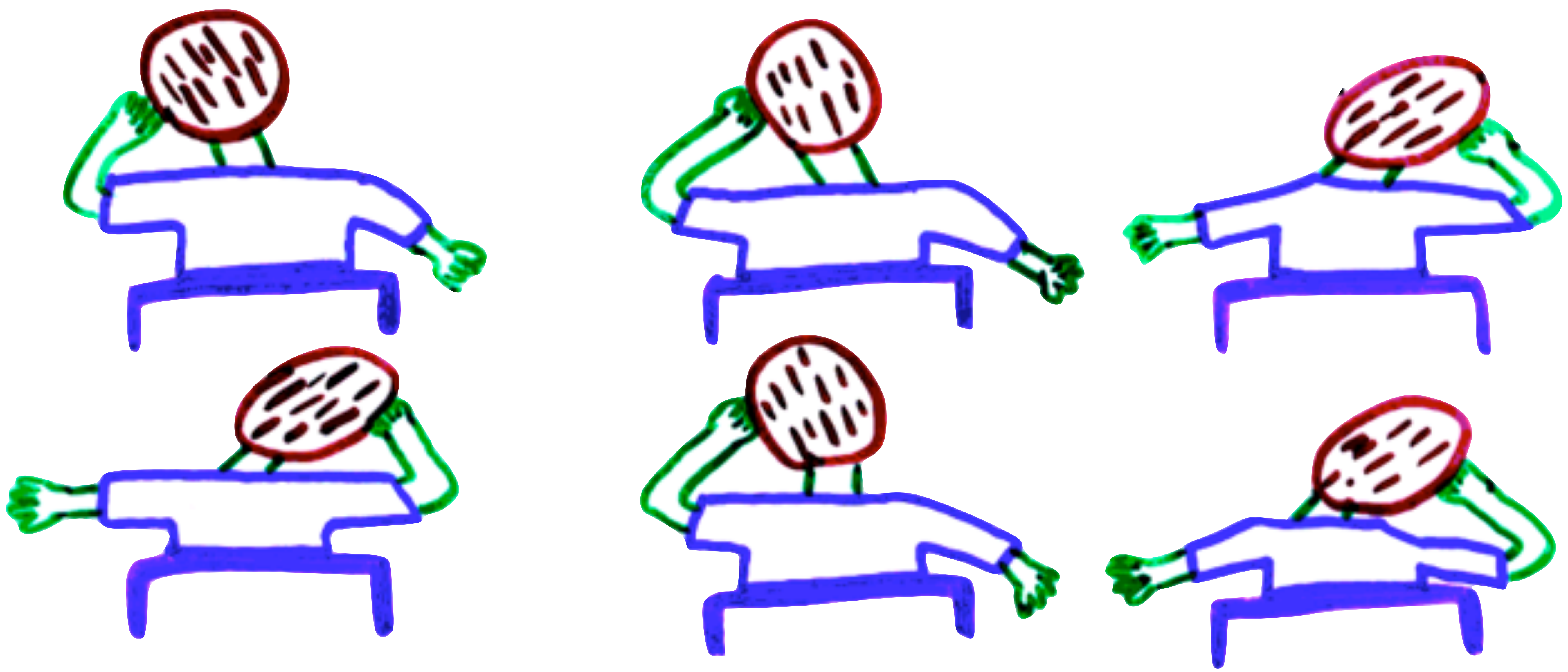
[Agrawal, Bender, Das, Kuszmaul, Peserico, Scquizzato SODA 2021, SPAA 2022]

Theorem: No online green-paging algorithm ($k_{max}=k$ and $k_{min}=k/p$) can be o(log *P*)-competitive.

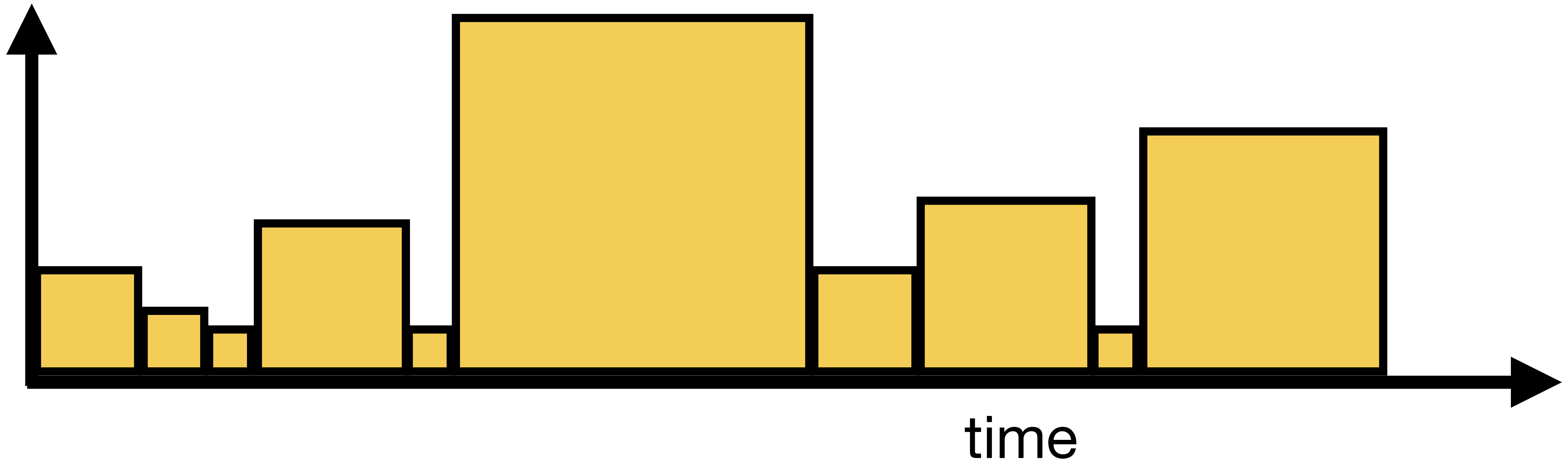
⟹ no parallel paging algorithm can be o(log P)-competitive.

Theorem: ∃ a *universal green-paging solution* that is O(log *P*)-competitive for all request sequences.

⟹ ∃ *universal* O(log P)-competitive parallel-paging algorithm.



Green-paging competitive ratio: O(log *p*)-competitive universal algorithm

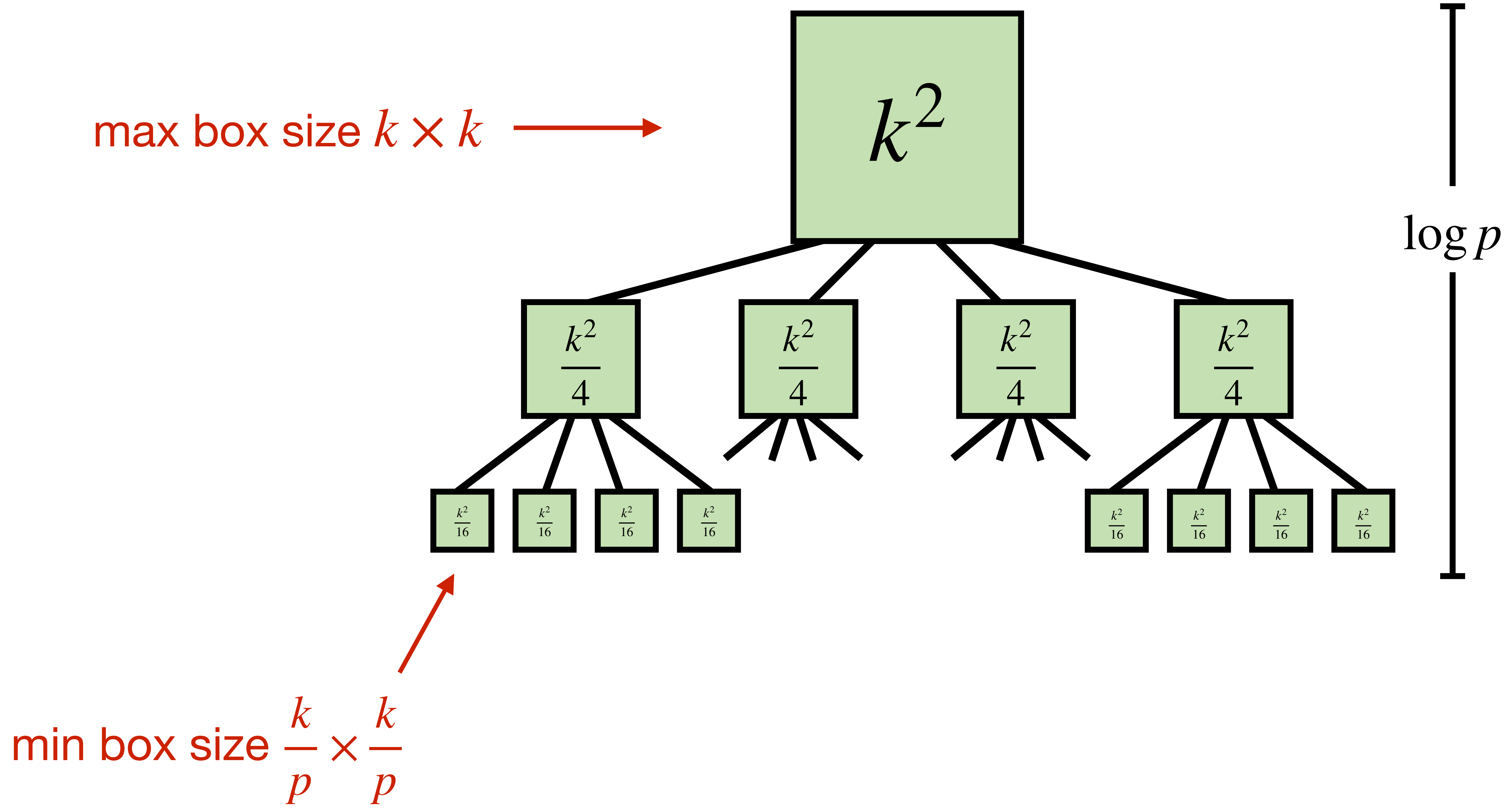


Thm: one can approximate any green-paging solution with a box-profile solution for the same asymptotic cost.

[Bender, Ebrahimi, Fineman, Ghasemiesfeh, Johnson, McCauley SODA 2014]

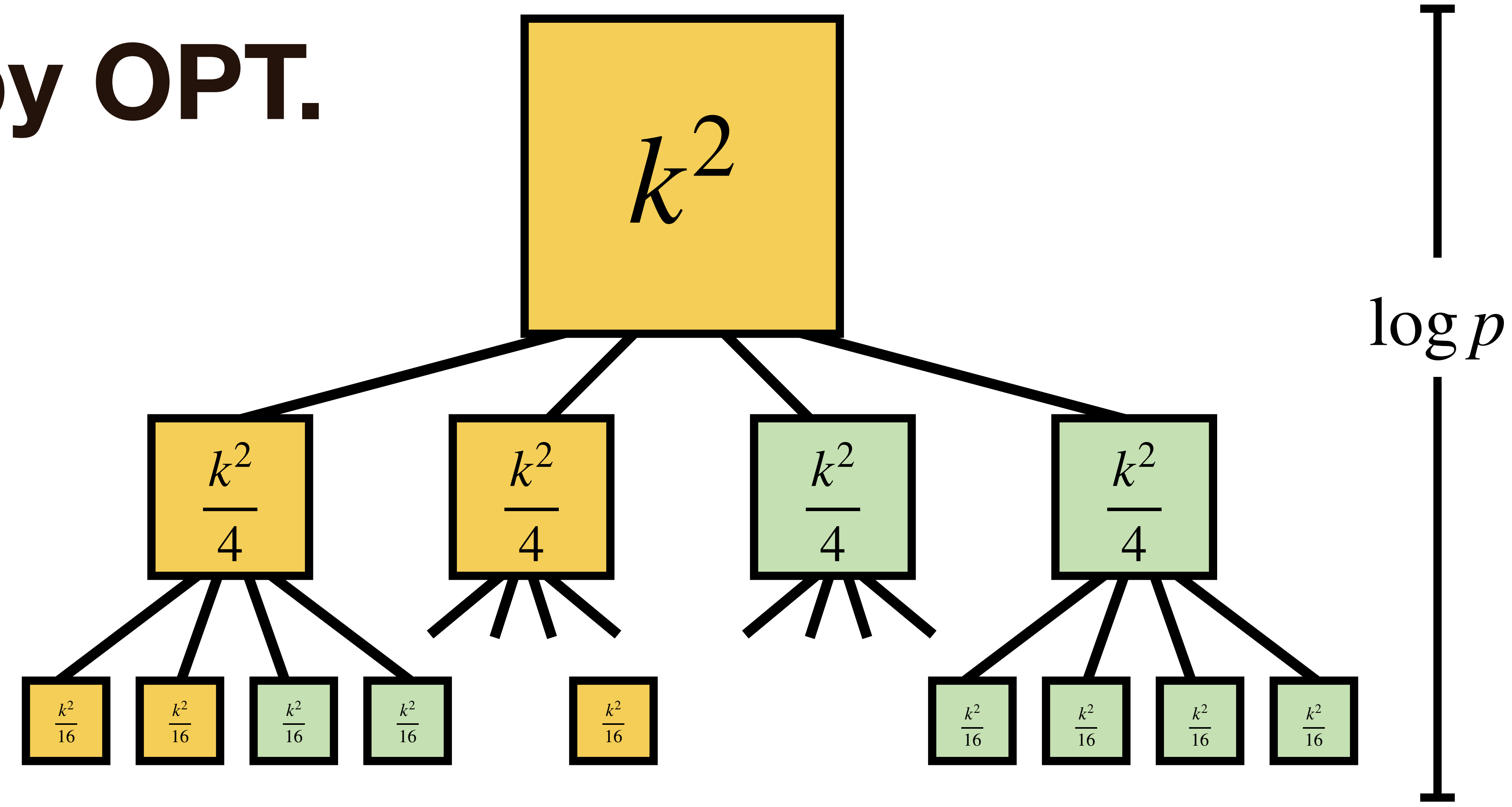
Green-paging competitive ratio: O(log *p*)-competitive universal algorithm

The universal green-paging memory profile is a *repeated* post-order traversal of this tree.

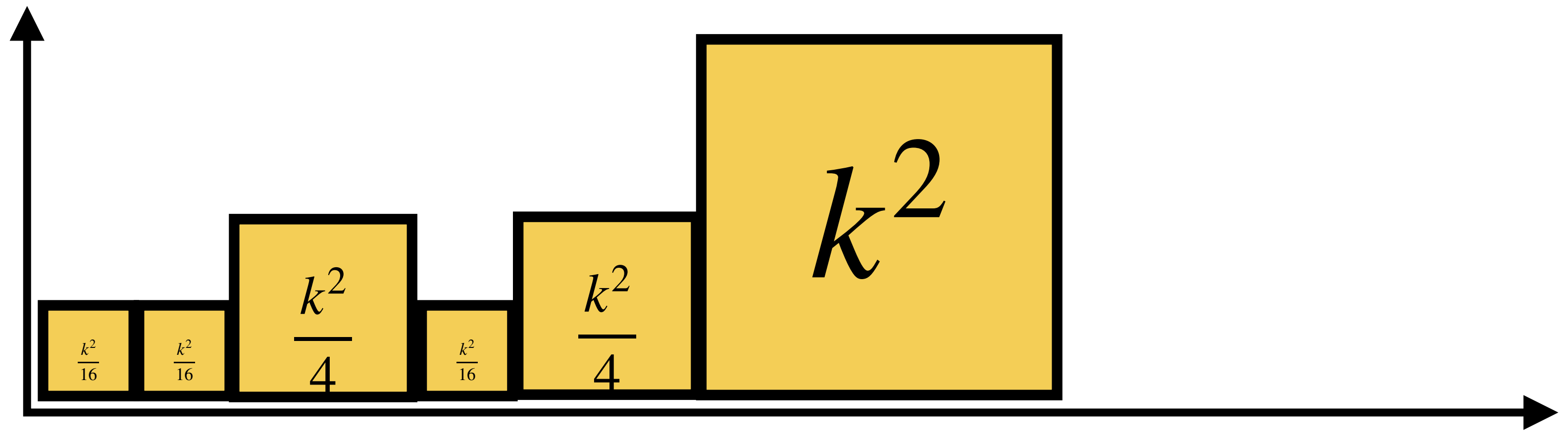


Green-paging competitive ratio: O(log *p*)-competitive universal algorithm

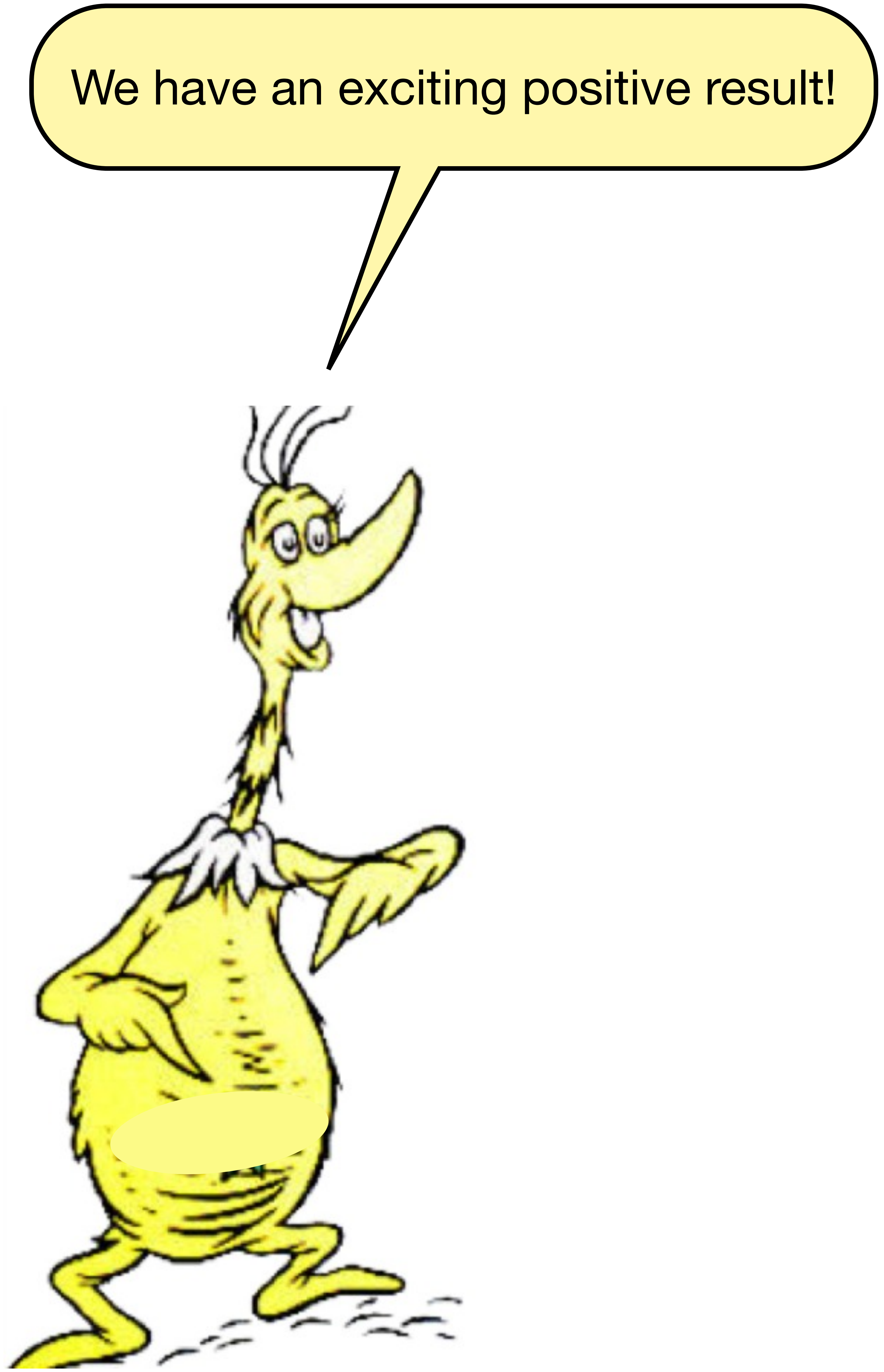
at least one box in root-to-leaf path is utilized by OPT.

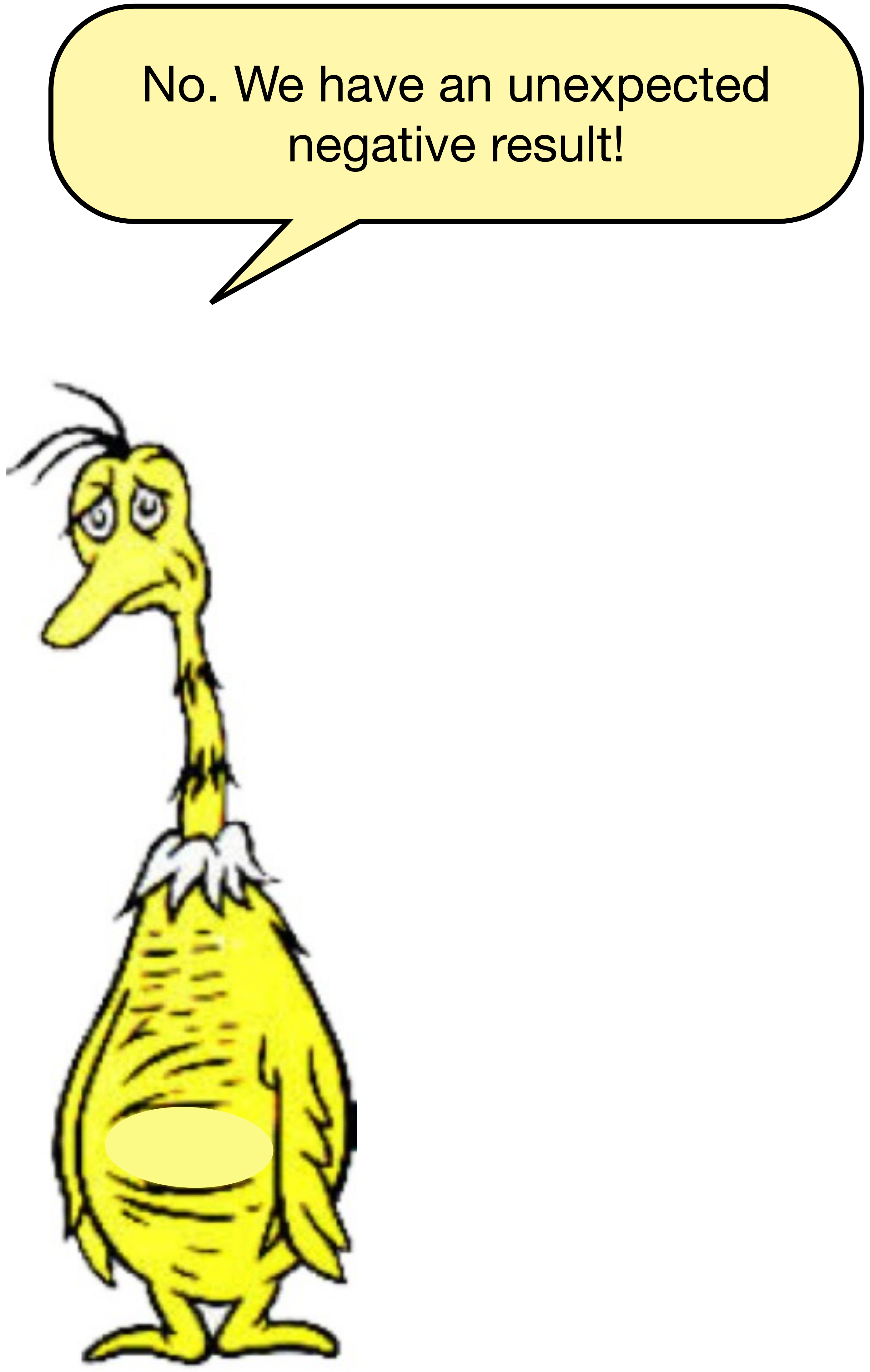


OPT

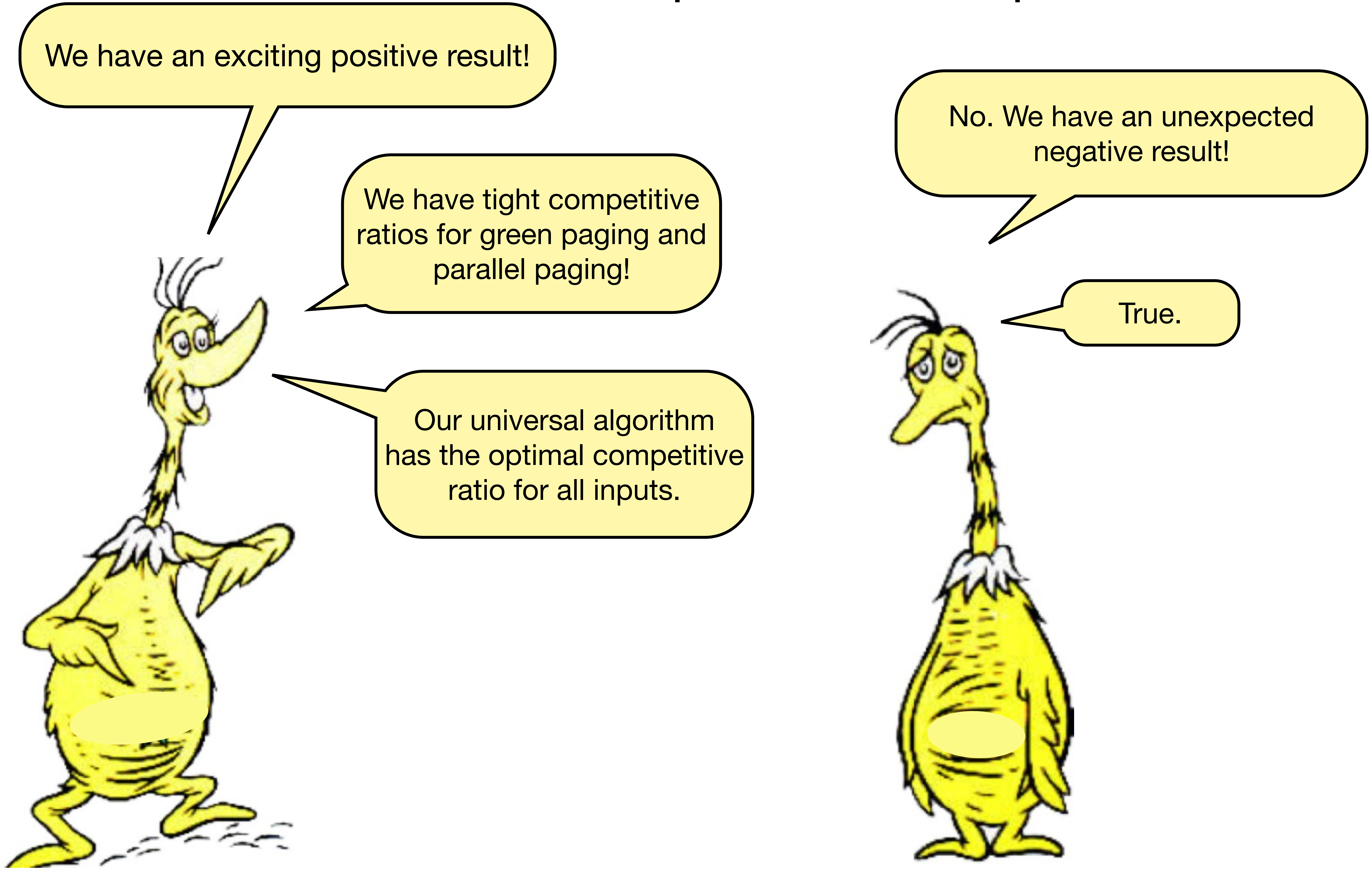


Optimist versus pessimist.

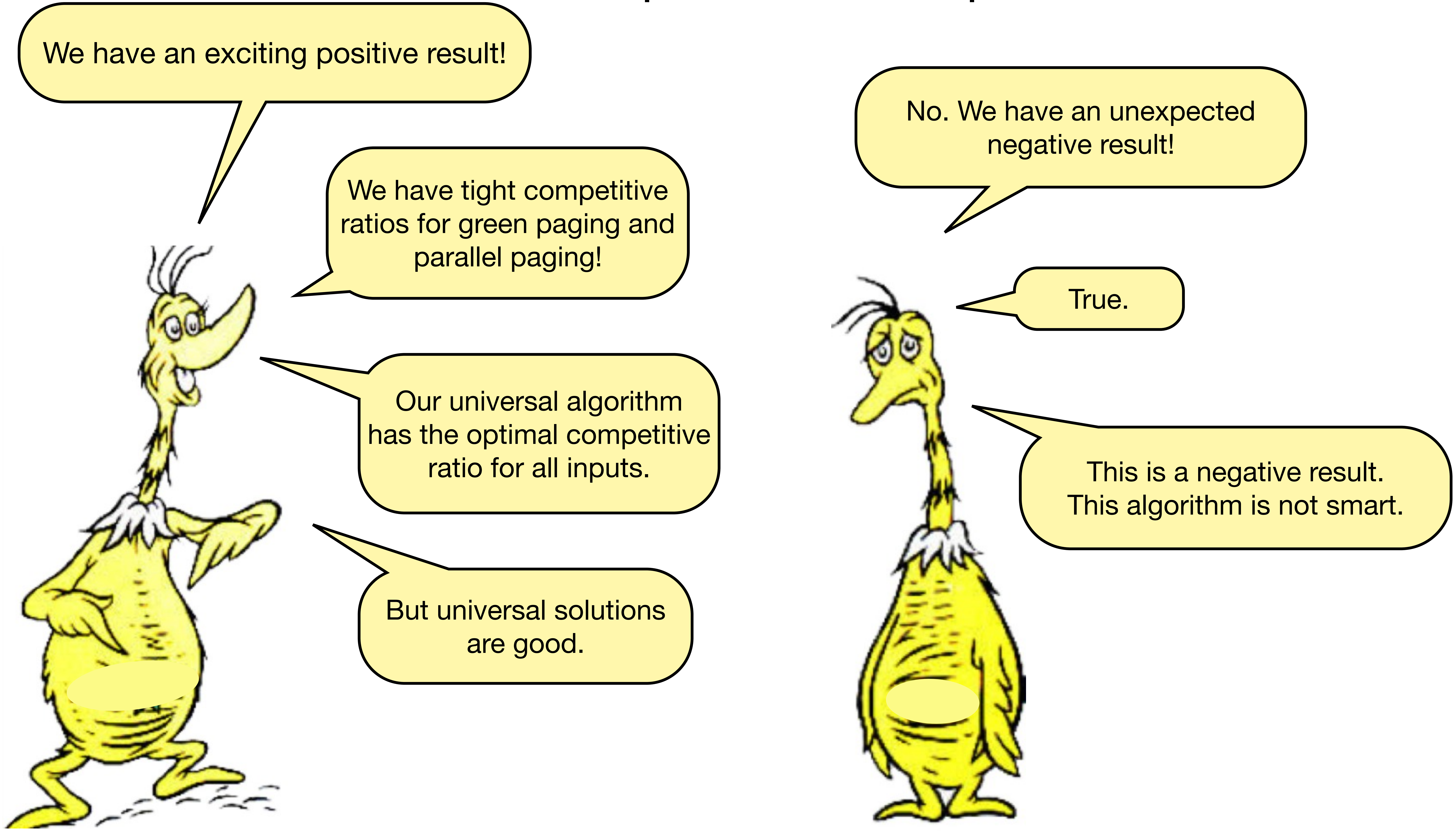




Optimist versus pessimist.



Optimist versus pessimist.



Optimist versus pessimist.

We have an exciting positive result!

We have tight competitive ratios for green paging and parallel paging!

Our universal algorithm has the optimal competitive ratio for all inputs.

But universal solutions are good.

Even so, we're much better than natural/naive solutions.

No. We have an unexpected negative result!

True.

This is a negative result. This algorithm is not smart.

Not all sequences have the same requirements. A good solution should be tailored to the problem instance. We provably cannot.

Optimist versus pessimist.

We have an exciting positive result!

No. We have an unexpected negative result!

We have tight competitive ratios for green paging and parallel paging!

True.

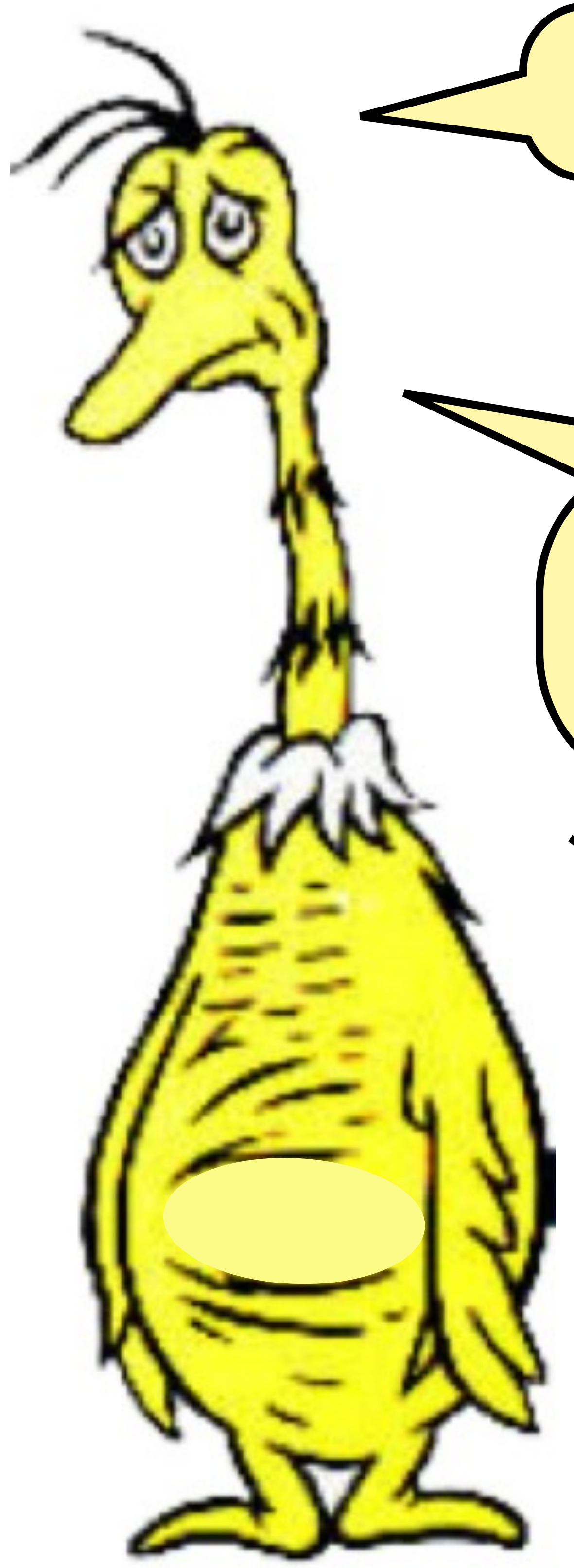
Our universal algorithm has the optimal competitive ratio for all inputs.

This is a negative result. This algorithm is not smart.

But universal solutions are good.

Not all sequences have the same requirements. A good solution should be tailored to the problem instance. We provably cannot.

Even so, we're much better than natural/naive solutions.



Conclusion

We now finally have the tools for reasoning about parallel paging.

- eg., green paging and the notion of cache impact

We should use these tools for beyond-worst-case analysis and in actual systems.