amplab
UC BERKELEY

FairRide: Near-Optimal Fair Cache Sharing

Qifan Pu,
Haoyuan Li,
Matei Zaharia,
Ali Ghodsi,
Ion Stoica

Caches are crucial

Caches are crucial

Caches are crucial

memCached

Caches are crucial

ALLUXIO

Spark

memCached

Cache sharing

Cache sharing

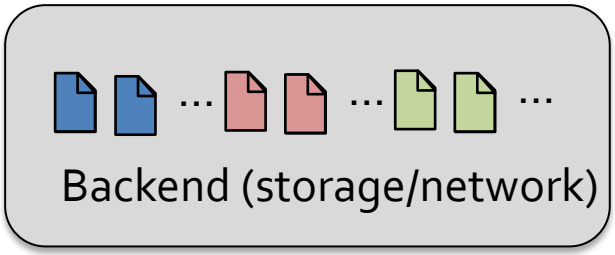
- Increasingly, caches are shared among multiple users

Cache sharing

- Increasingly, caches are shared among multiple users
  - Especially with the advent of cloud

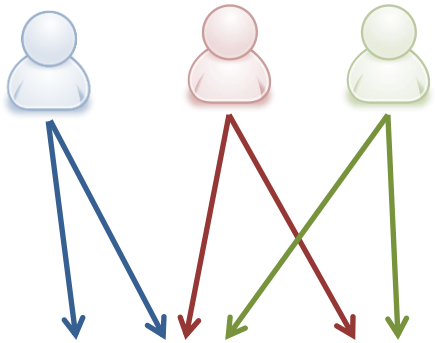
Cache sharing

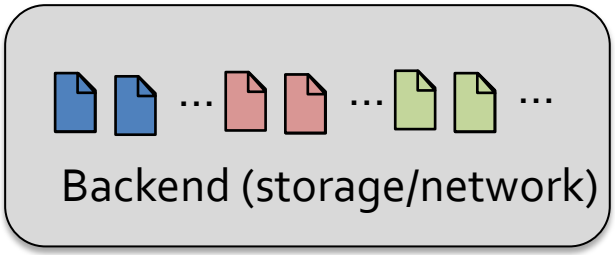
- Increasingly, caches are shared among multiple users
  - Especially with the advent of cloud



Cache sharing

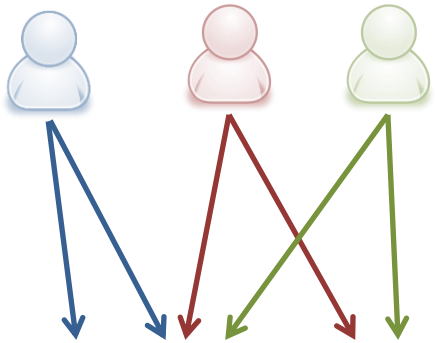
- Increasingly, caches are shared among multiple users
  - Especially with the advent of cloud





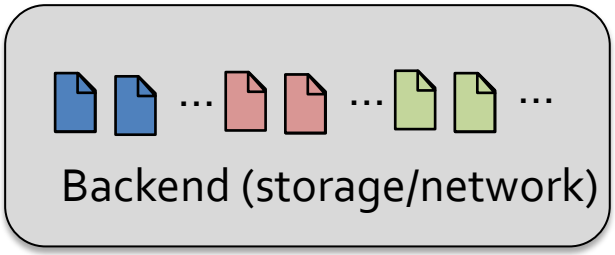
Cache sharing

- Increasingly, caches are shared among multiple users
  - Especially with the advent of cloud



Benefits:

- Provide low latency
- Reduce backend load

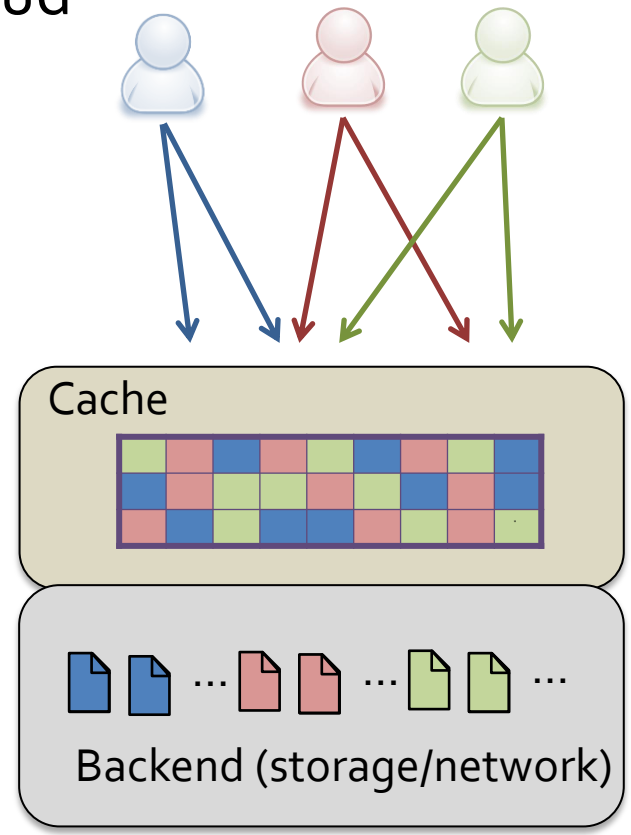


Cache sharing

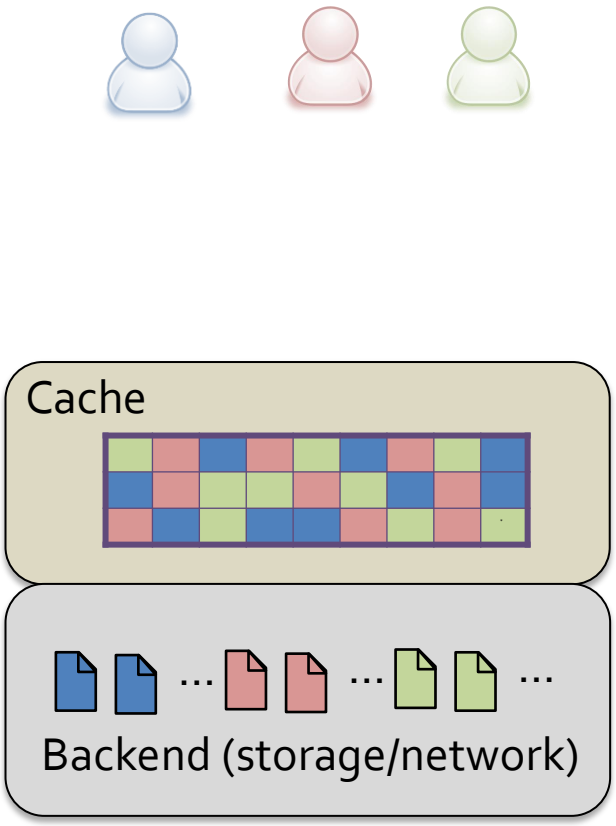
- Increasingly, caches are shared among multiple users
  - Especially with the advent of cloud

Benefits:

- Provide low latency
- Reduce backend load

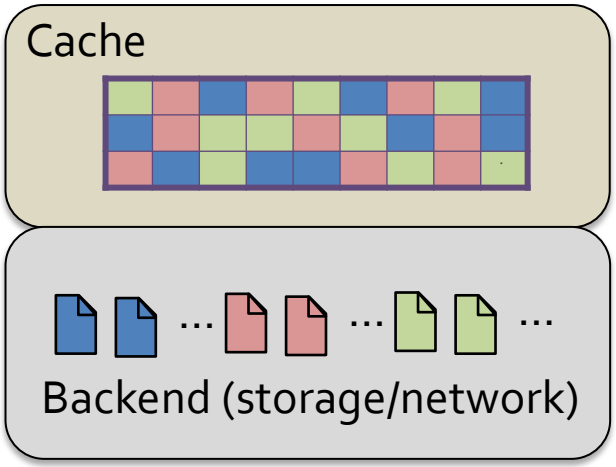


Problems with cache algorithms



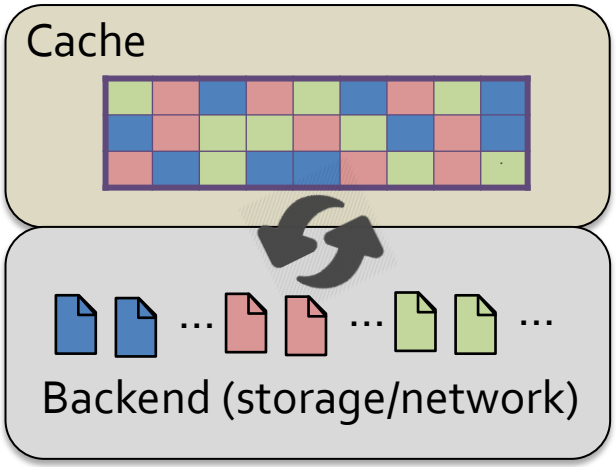
Problems with cache algorithms

- LRU, LFU, LRU-K…

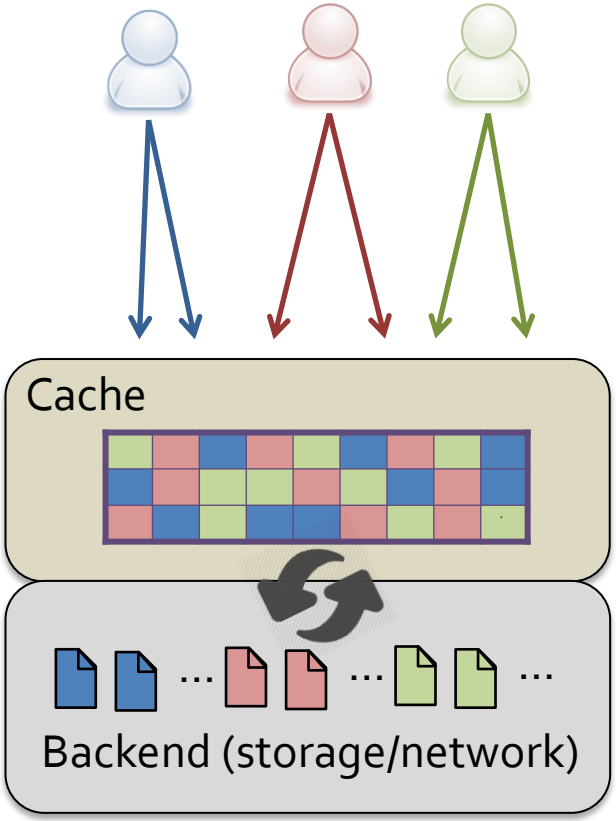


Problems with cache algorithms

- LRU, LFU, LRU-K…

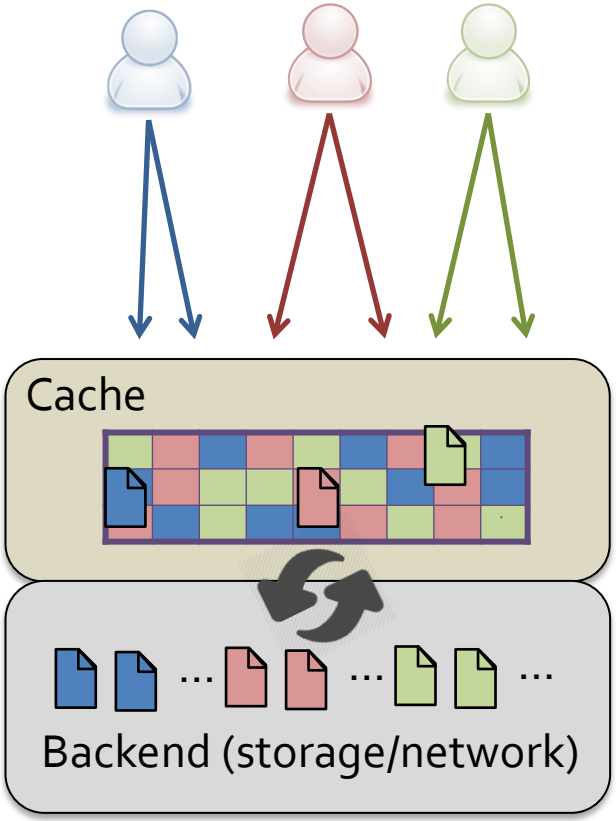


Problems with cache algorithms



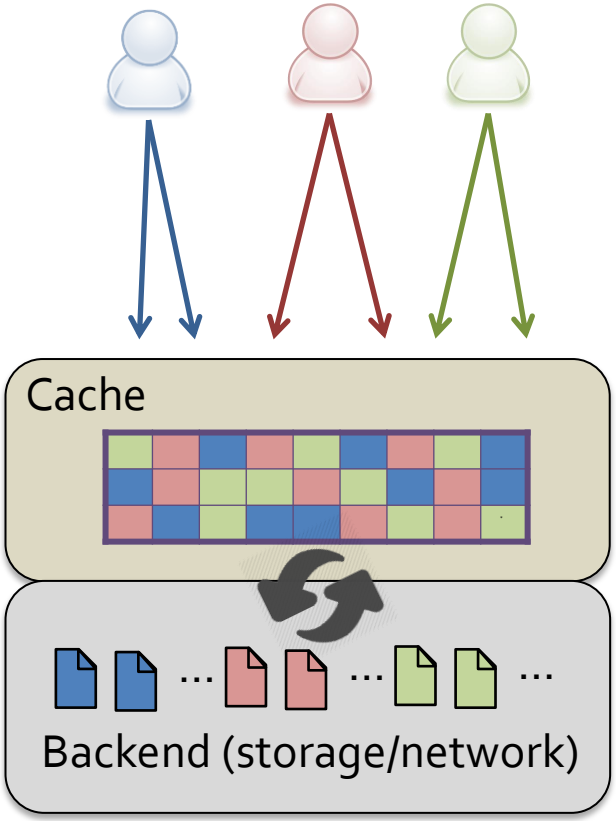
- LRU, LFU, LRU-K…

Problems with cache algorithms



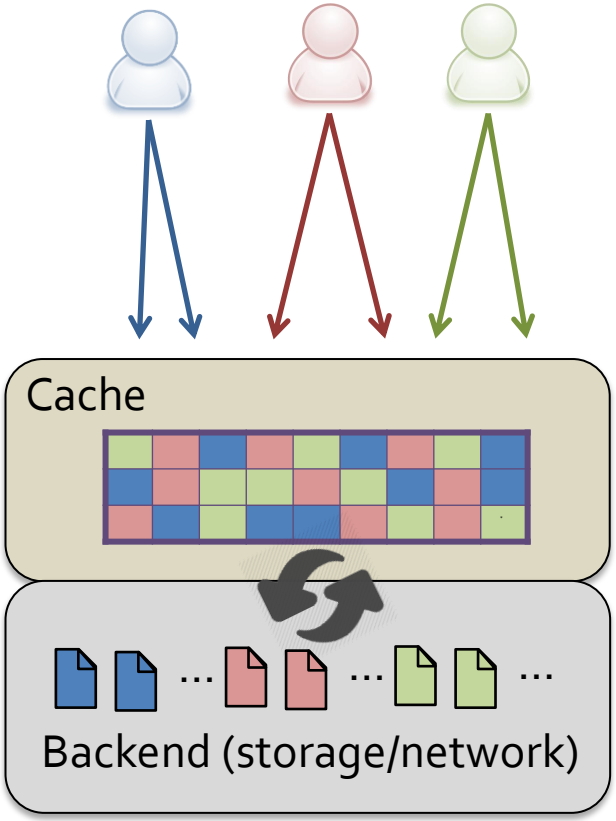
- LRU, LFU, LRU-K…

Problems with cache algorithms



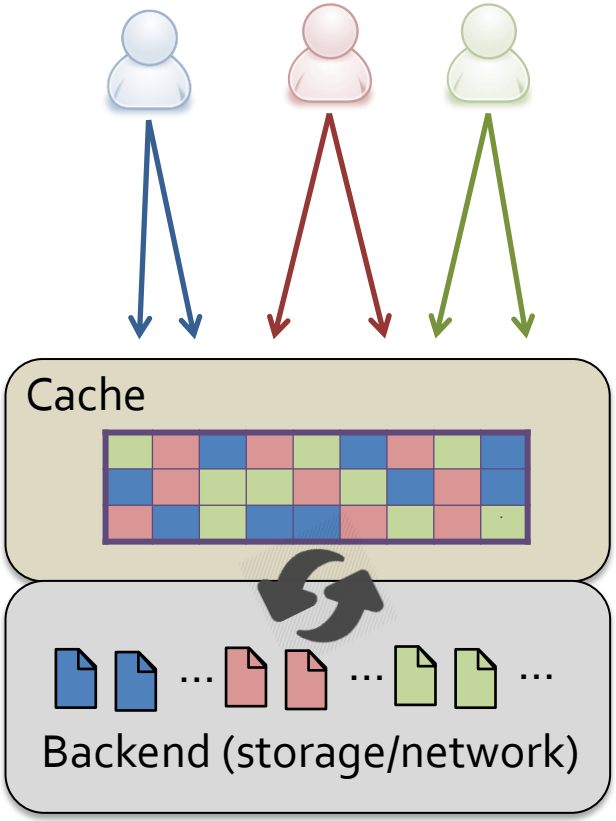
- LRU, LFU, LRU-K…
  - Cache data likely to be accessed in the future

Problems with cache algorithms



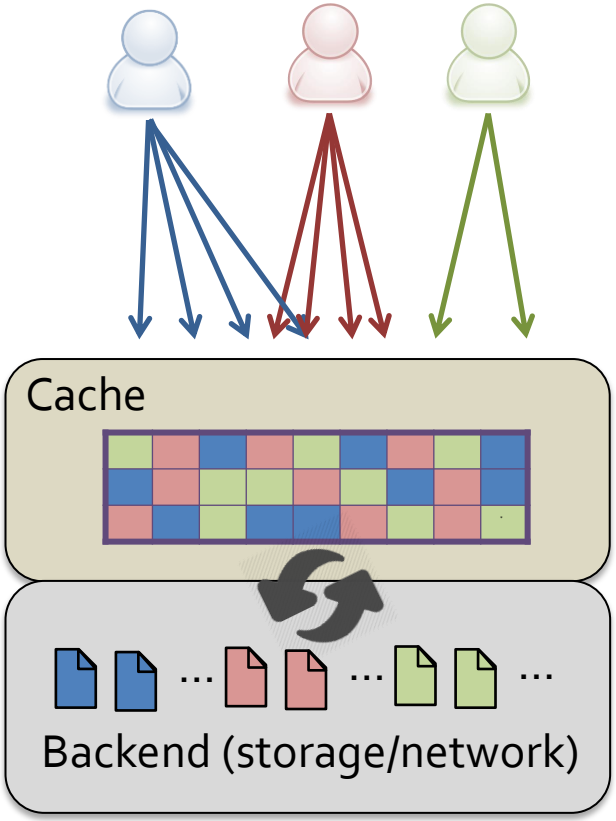
- LRU, LFU, LRU-K…
  - Cache data likely to be accessed in the future
- Optimize global efficiency

Problems with cache algorithms



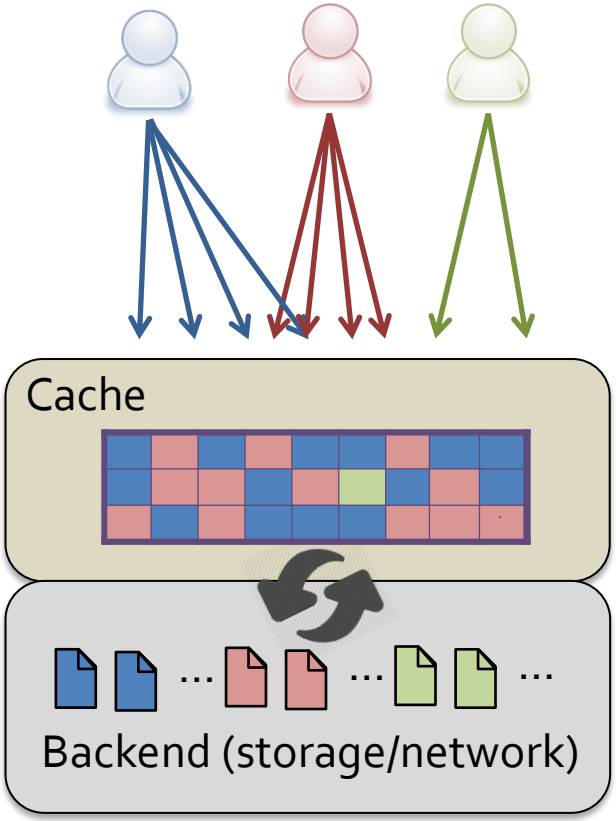
- LRU, LFU, LRU-K…
  - Cache data likely to be accessed in the future
- Optimize global efficiency
- Single user gets arbitrarily small cache

Problems with cache algorithms



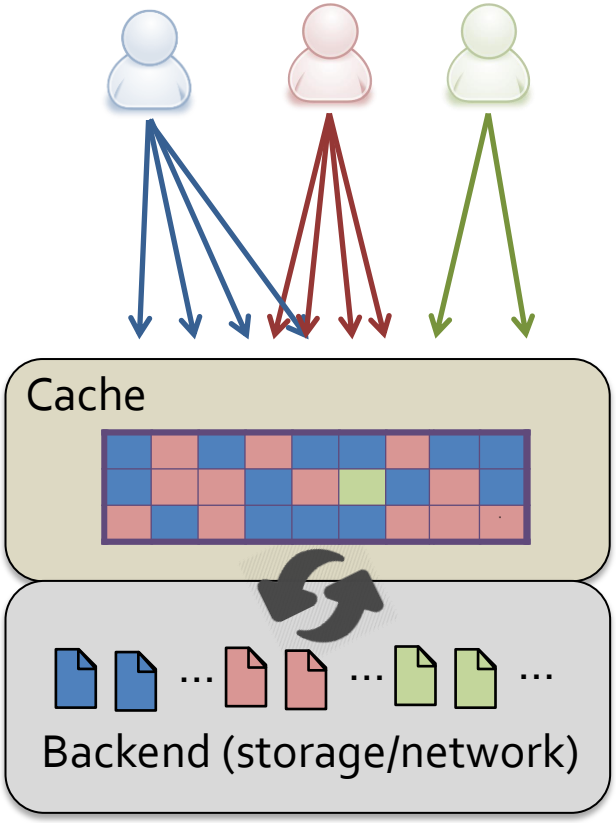
- LRU, LFU, LRU-K…
  - Cache data likely to be accessed in the future
- Optimize global efficiency
- Single user gets arbitrarily small cache

Problems with cache algorithms



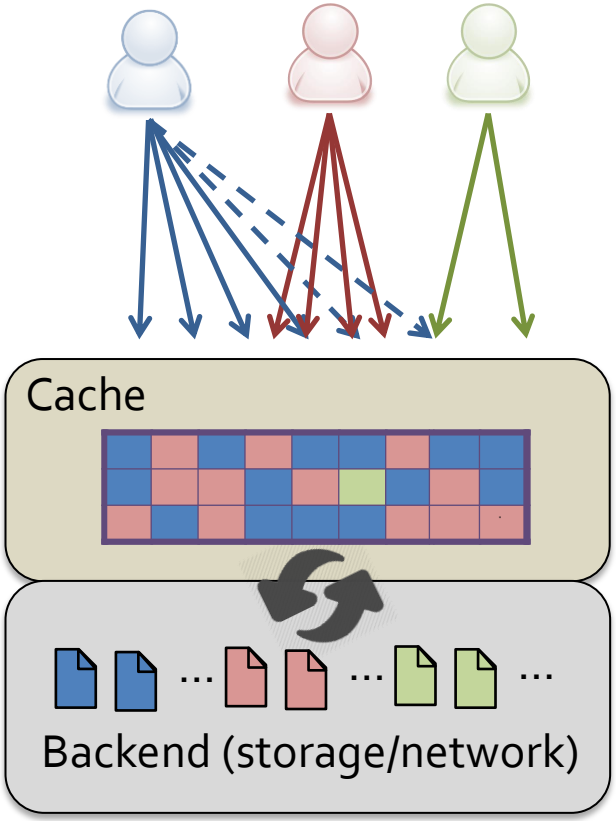
- LRU, LFU, LRU-K…
  - Cache data likely to be accessed in the future
- Optimize global efficiency
- Single user gets arbitrarily small cache

Problems with cache algorithms



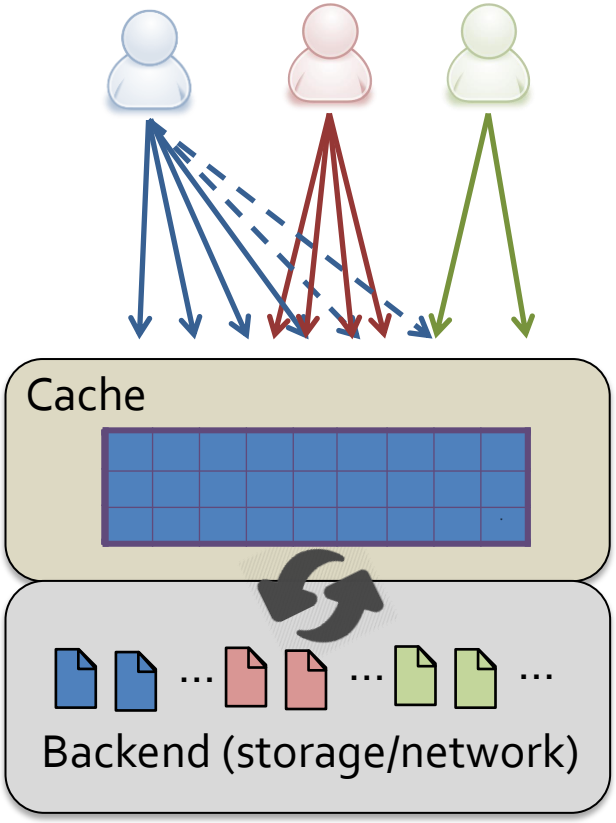
- LRU, LFU, LRU-K…
  - Cache data likely to be accessed in the future
- Optimize global efficiency
- Single user gets arbitrarily small cache
- Prone to strategic behavior

Problems with cache algorithms



- LRU, LFU, LRU-K…
  - Cache data likely to be accessed in the future
- Optimize global efficiency
- Single user gets arbitrarily small cache
- Prone to strategic behavior

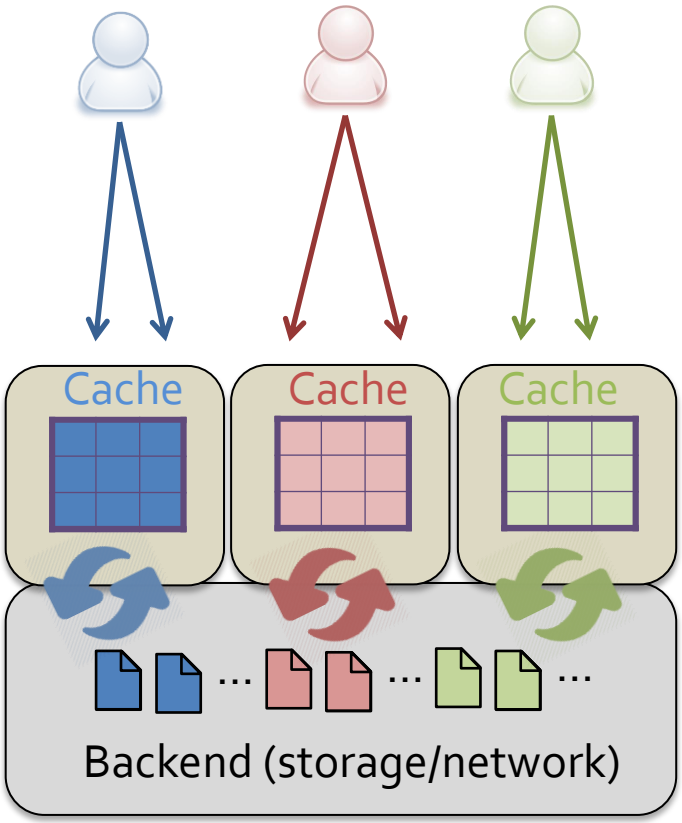
Problems with cache algorithms



- LRU, LFU, LRU-K…
  - Cache data likely to be accessed in the future
- Optimize global efficiency
- Single user gets arbitrarily small cache
- Prone to strategic behavior

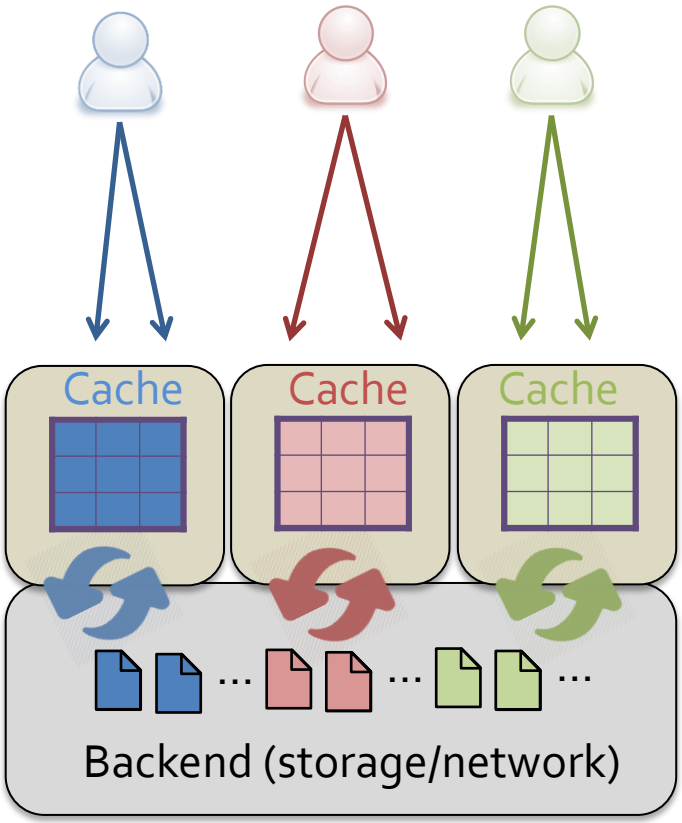
Solution?

Statically allocated



Solution?

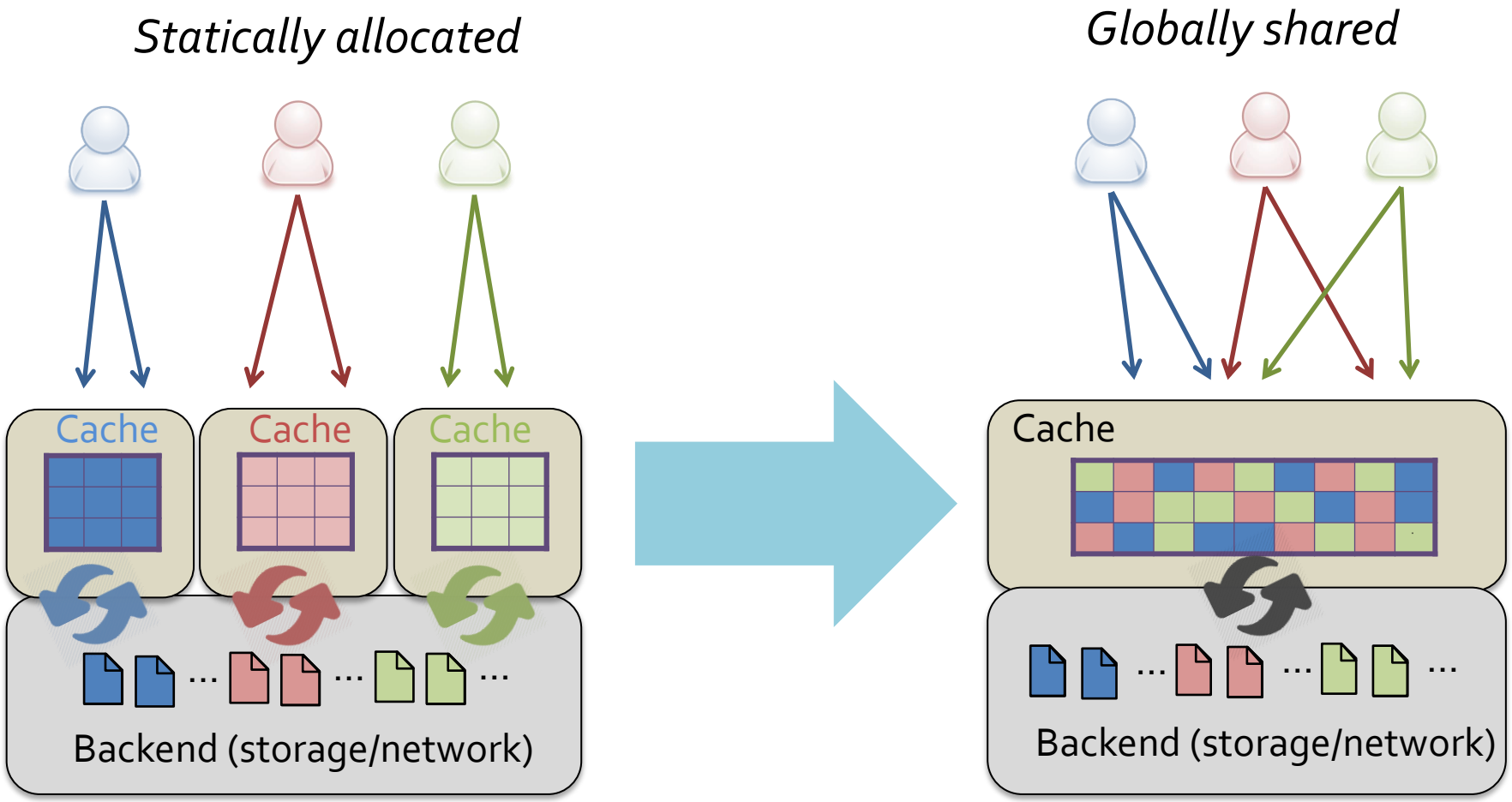
Statically allocated



Isolation

Strategy-proof

What we want

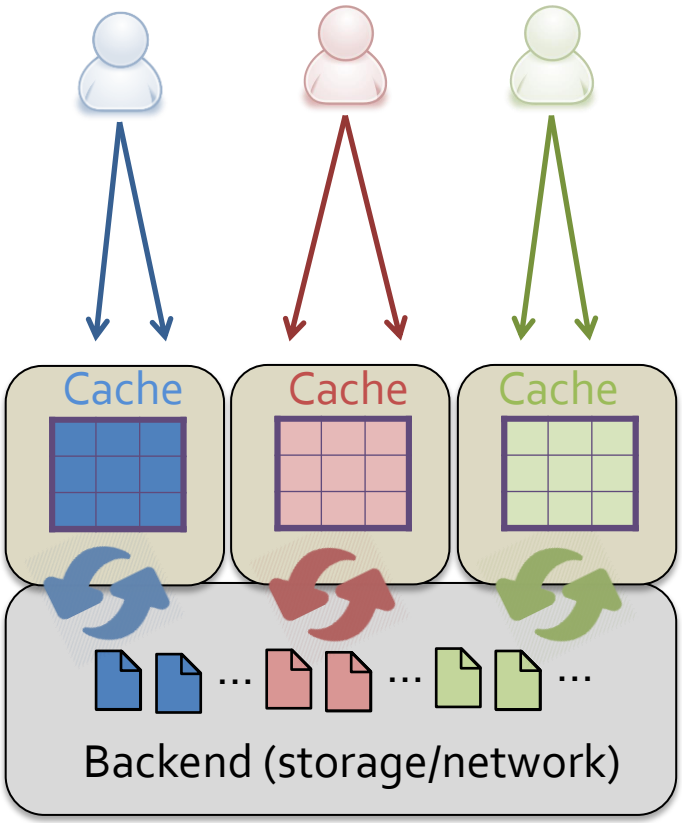


Isolation

Strategy-proof

What we want

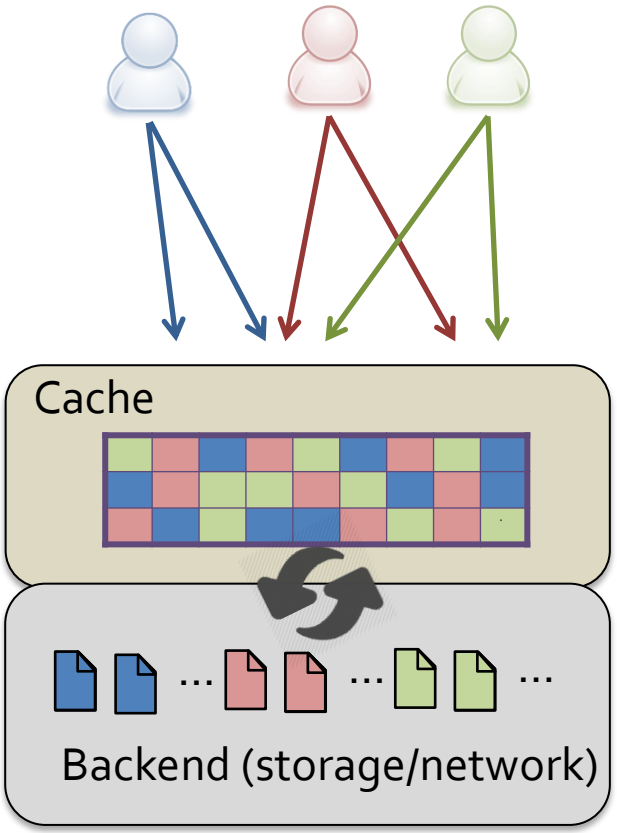
Statically allocated



Isolation

Strategy-proof

Globally shared

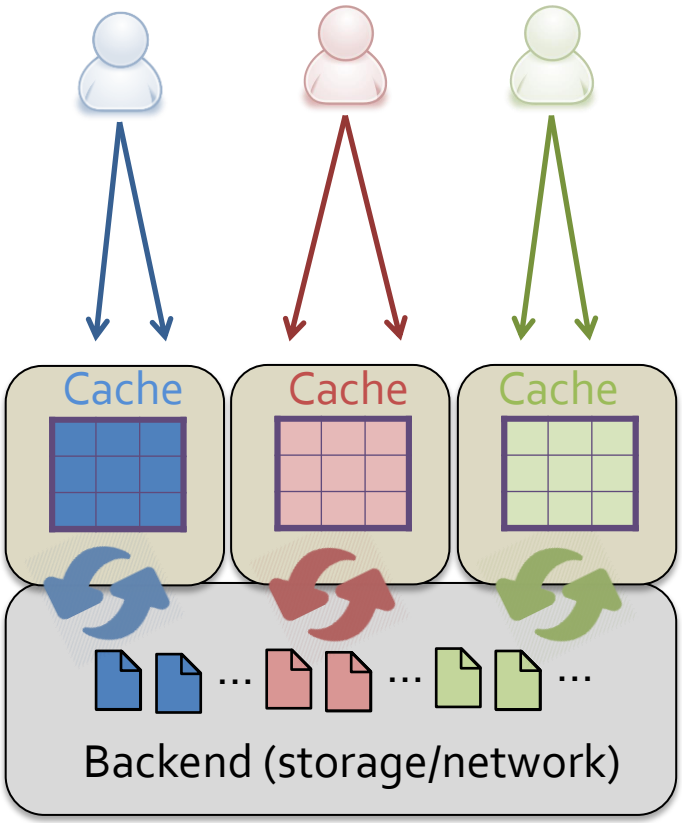


Higher utilization

Share data

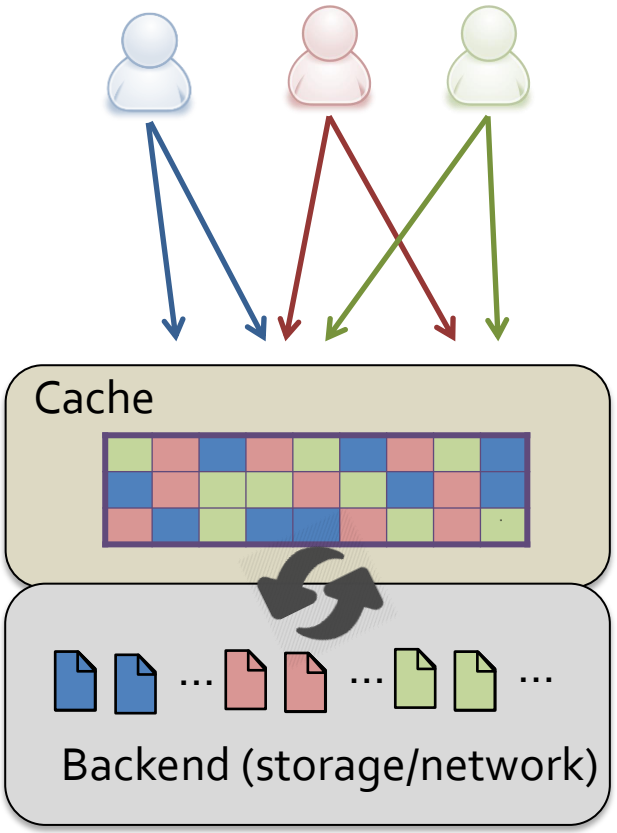
What we want

Statically allocated



Isolation
Strategy-proof

Globally shared



Higher utilization
Share data

Our contribution

Our contribution

- First analysis of cache allocation policies with well defined resource-sharing properties.

Our contribution

- First analysis of cache allocation policies with well defined resource-sharing properties.

- **Impossibility result:**
  no policy achieves all good properties!

Our contribution

- First analysis of cache allocation policies with well defined resource-sharing properties.

- **Impossibility result:**
  no policy achieves all good properties!

- **A new policy that is near-optimal and outperforms other policies when users cheat.**

A simple model

A simple model

- Users access equal-sized files at constant rates
  - r_{ij} the rate user *i* accesses file *j*

A simple model

- Users access equal-sized files at constant rates
  - r_{ij} the rate user *i* accesses file *j*
- A allocation **policy** decides which files to cache
  - p_j the % of file *j* put in cache

A simple model

- Users access equal-sized files at constant rates
  - r_{ij} the rate user *i* accesses file *j*
- A allocation **policy** decides which files to cache
  - p_j the % of file *j* put in cache
- Users care their hit ratio
  - user *i*'s hit ratio:

A simple model

- Users access equal-sized files at constant rates
  - r_{ij} the rate user *i* accesses file *j*
- A allocation **policy** decides which files to cache
  - p_j the % of file *j* put in cache
- Users care their hit ratio $HR_i = \frac{total\_hits}{total\_accesses} = \frac{\sum_j p_j r_{ij}}{\sum_j r_{ij}}$
  - user *i*'s hit ratio:

A simple model

- Users access equal-sized files at constant rates
  - r_{ij} the rate user *i* accesses file *j*
- A allocation **policy** decides which files to cache
  - p_j the % of file *j* put in cache
- Users care their hit ratio $HR_i = \frac{total\_hits}{total\_accesses} = \frac{\sum_j p_j r_{ij}}{\sum_j r_{ij}}$
  - user *i*'s hit ratio:

◆ Results hold with varied file sizes, access partial files, p_j is binary, etc.

Outline

- What properties do we want?
- Can we extend max-min to solve the problem?
- How do we solve it? (FairRide)
- How well does FairRide work in practice?

Properties

- Isolation Guarantee
  - No user should be worse off than static allocation

Properties

- Isolation Guarantee
  - No user should be worse off than static allocation
- Strategy-Proofness
  - No user can improve by cheating

Properties

- Isolation Guarantee
  - No user should be worse off than static allocation
- Strategy-Proofness
  - No user can improve by cheating
- Pareto Efficiency
  - Can't improve a user without hurting others

Strategy proofness

Strategy proofness

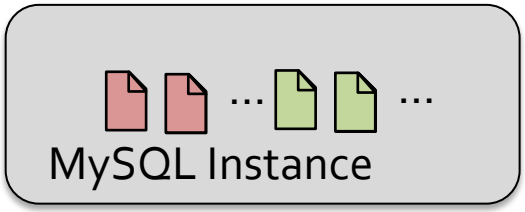
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses

Strategy proofness

- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice

Strategy proofness

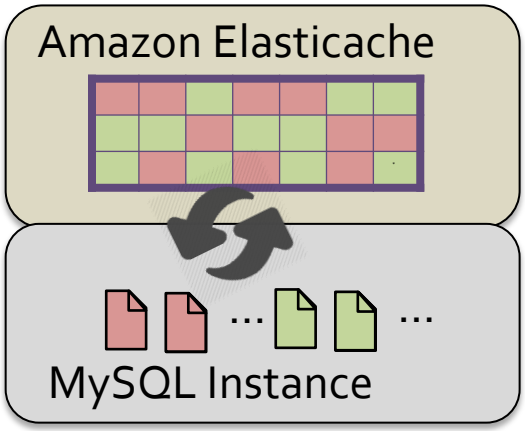
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice



Strategy proofness

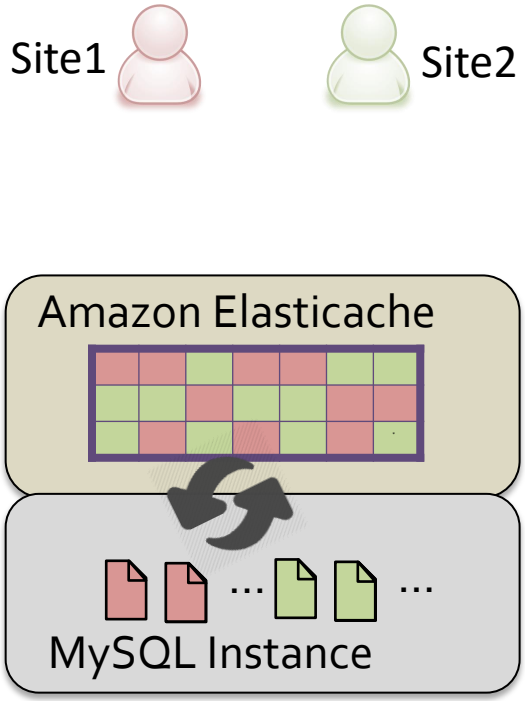
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice

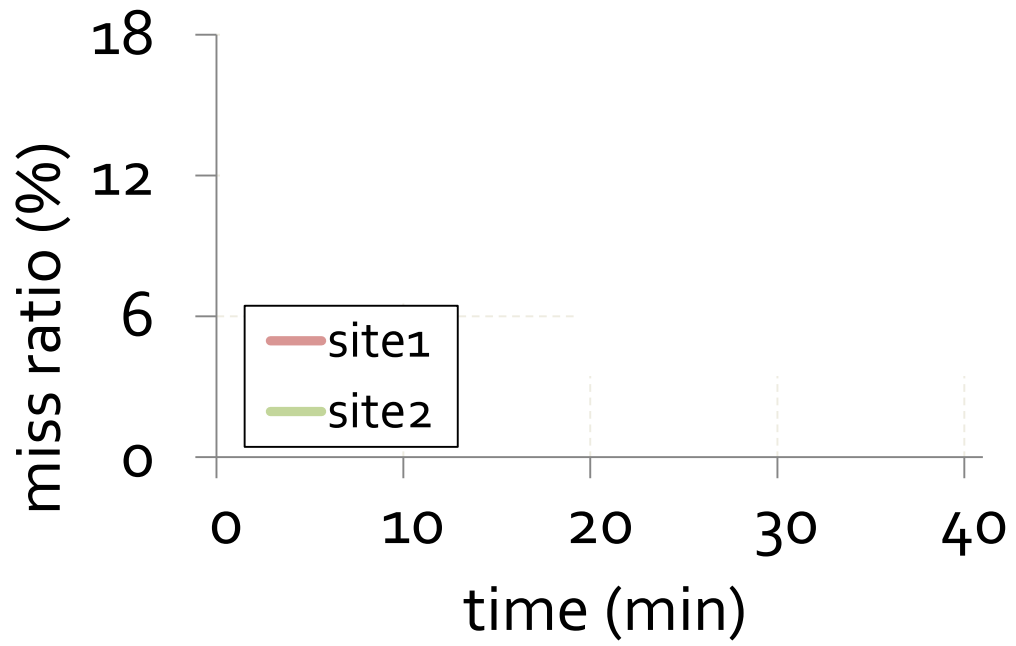
Site1 Site2



Strategy proofness

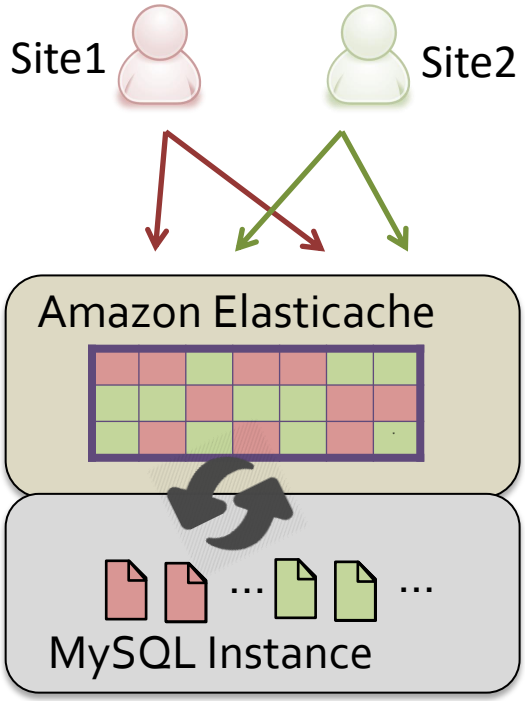
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice

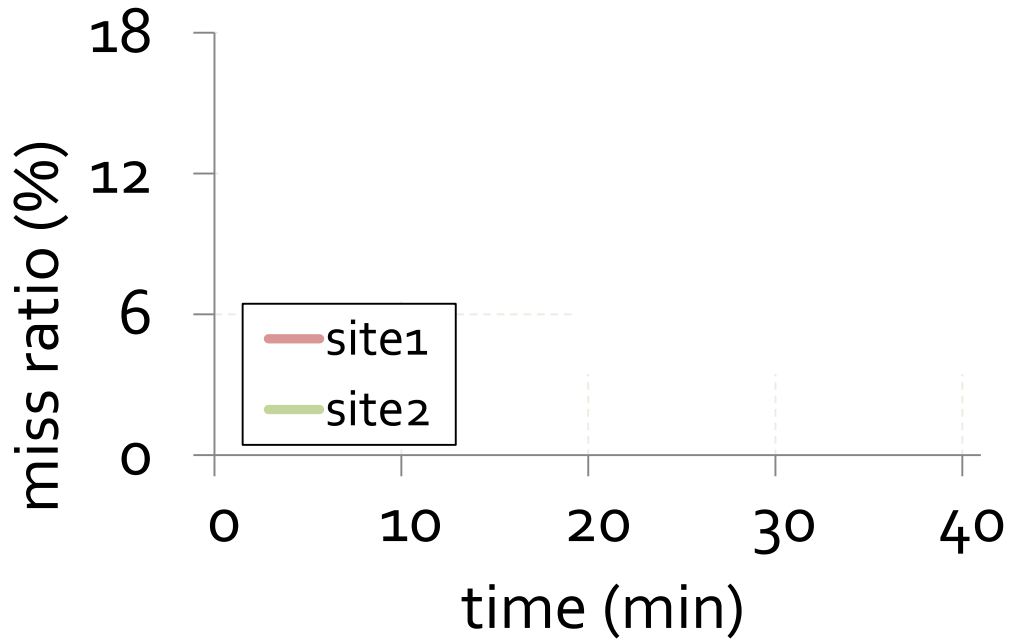




Strategy proofness

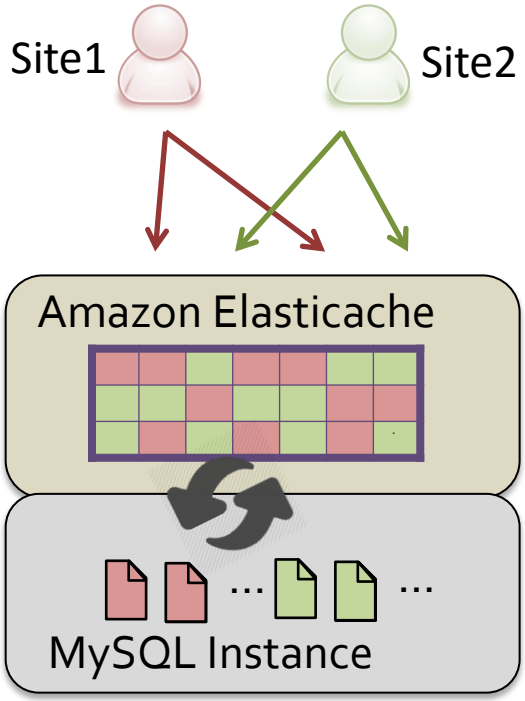
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice

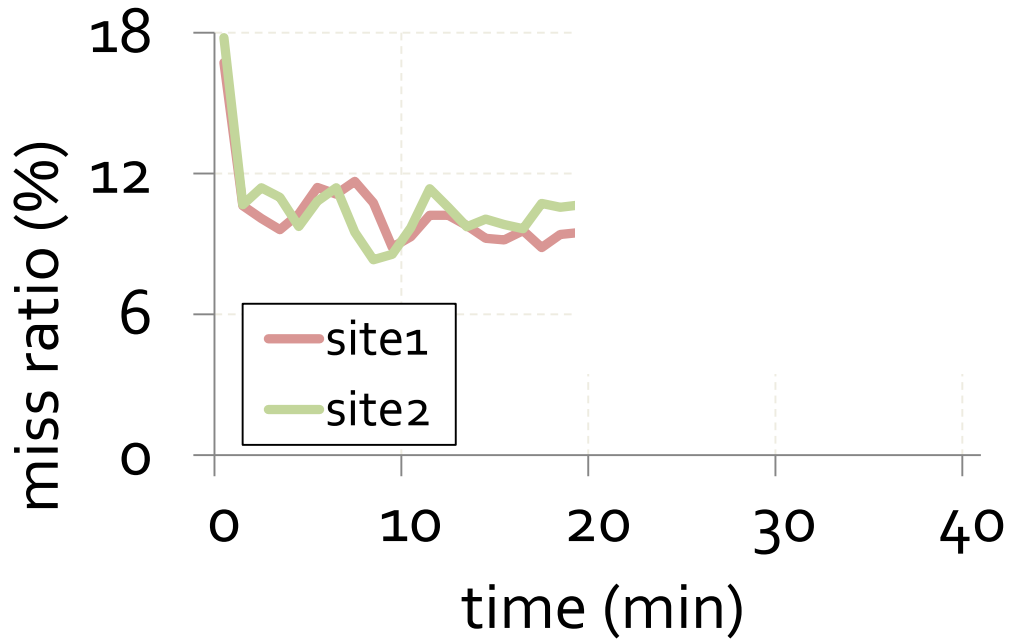




Strategy proofness

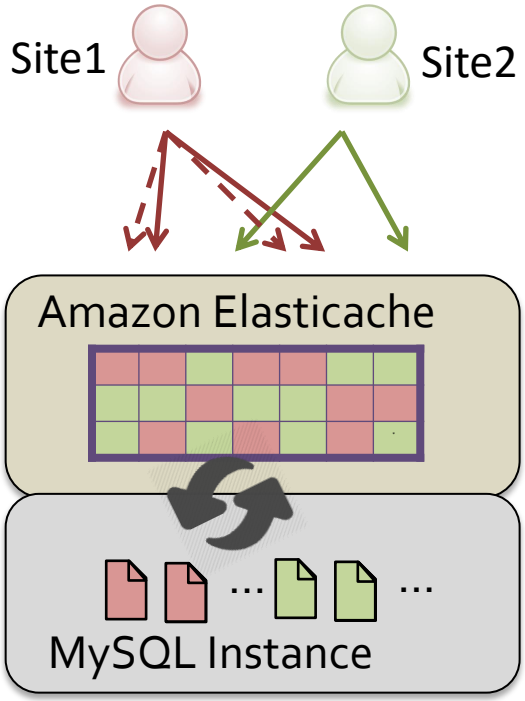
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice

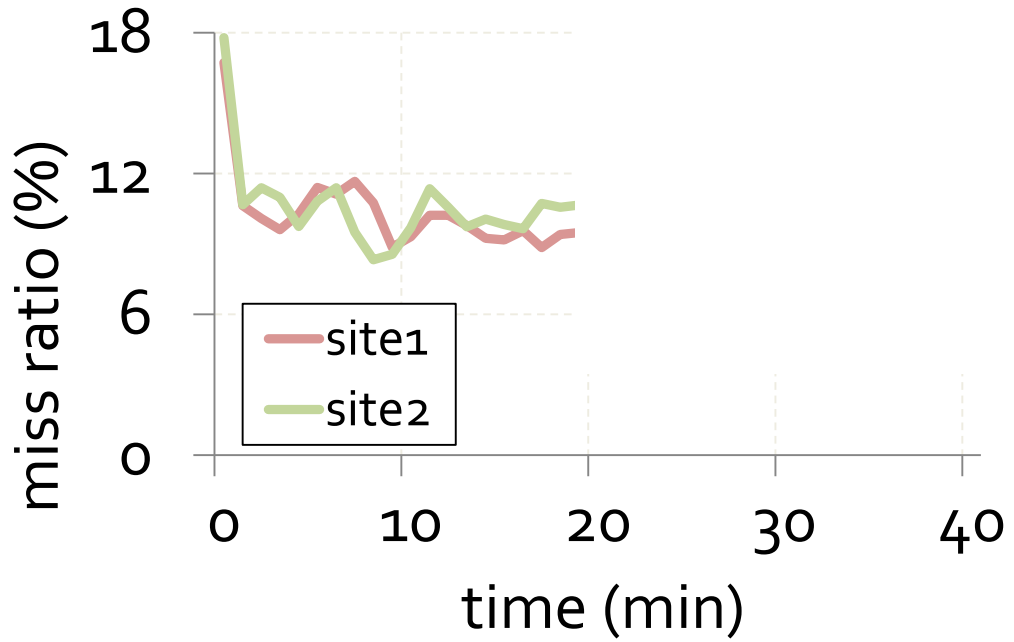




Strategy proofness

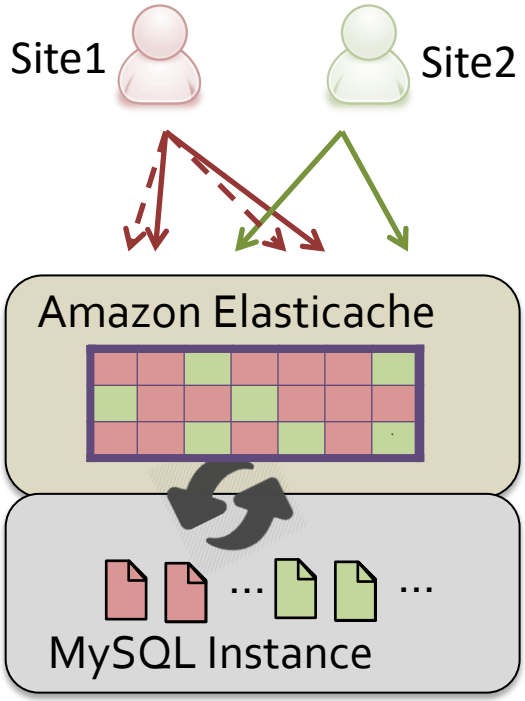
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice

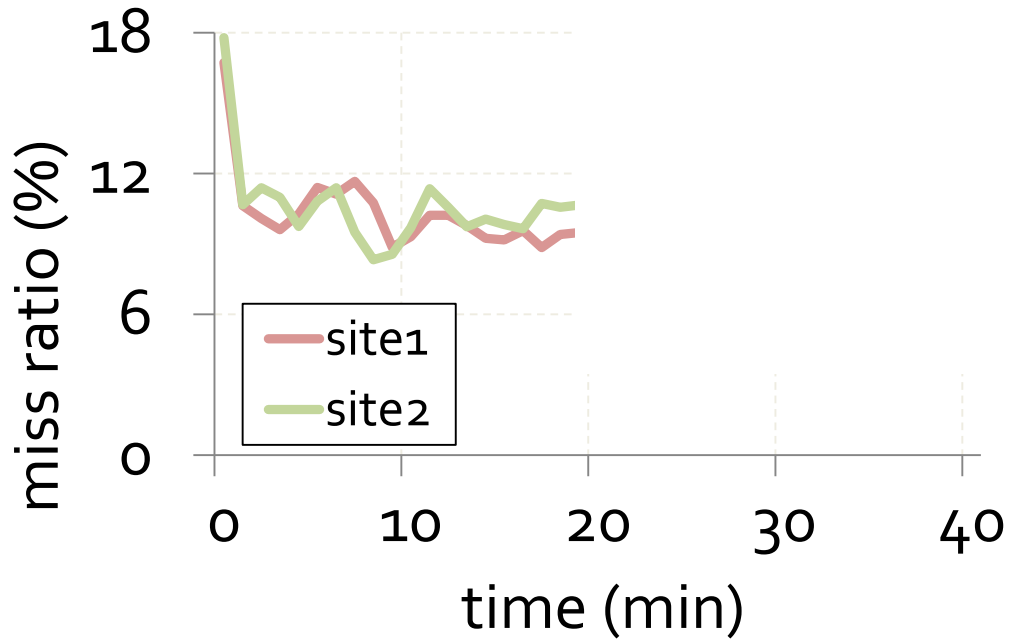




Strategy proofness

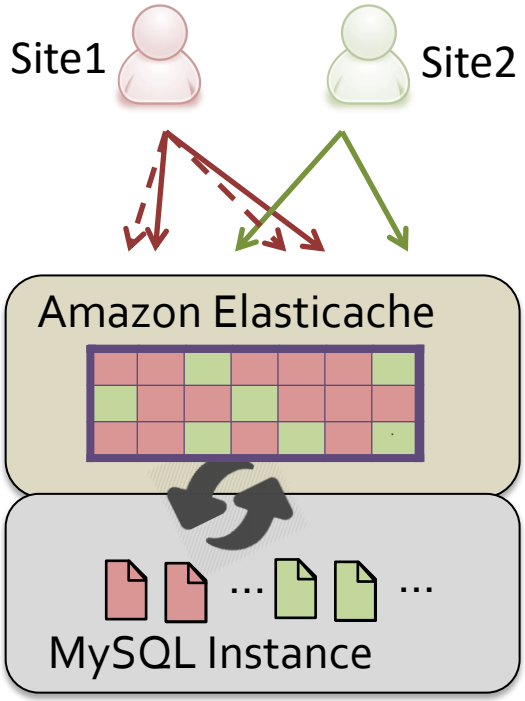
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice

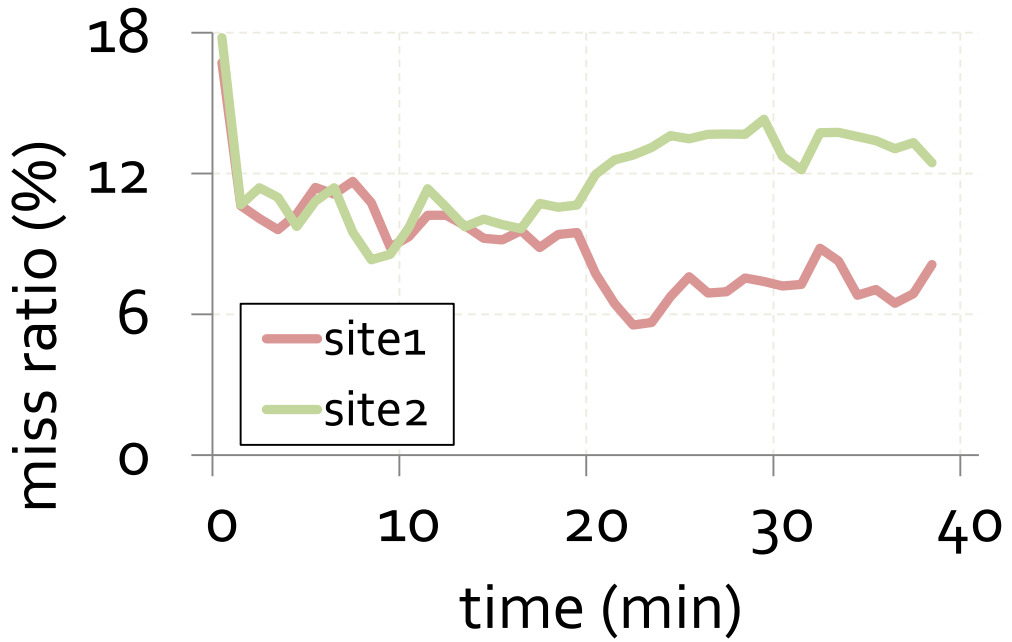




Strategy proofness

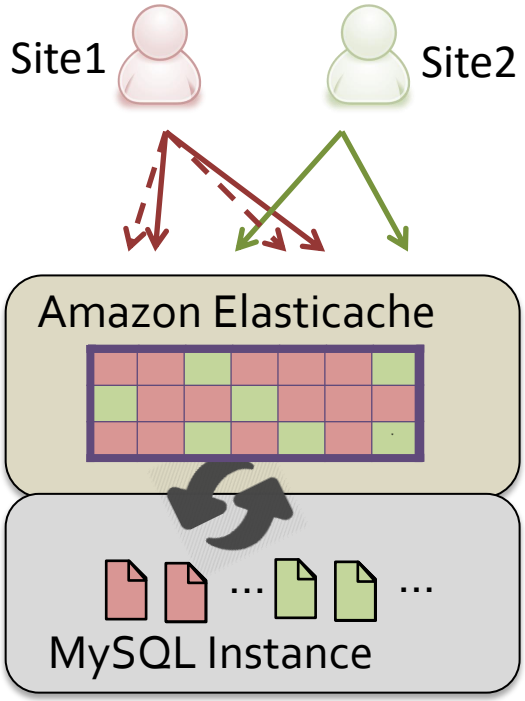
- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice

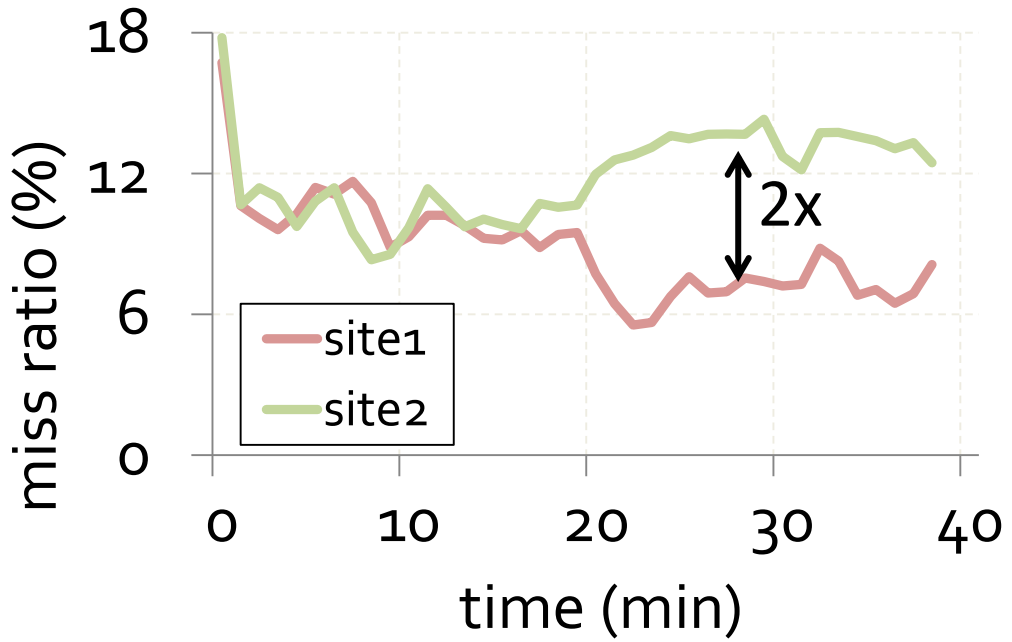




Strategy proofness

- Very easy to cheat, hard to detect
  - e.g., by making spurious accesses
- Can happen in practice





Properties

- Isolation Guarantee
  - No user should be worse off than static allocation
- Strategy-Proofness
  - No user can improve by cheating
- Pareto Efficiency
  - Can't improve a user without hurting others

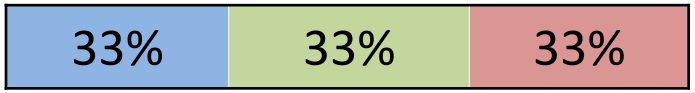
Outline

- What properties do we want?
- Can we extend ***max-min fairness*** to solve the problem?
- How do we solve it? (FairRide)
- How well does FairRide work in practice?

What is *max-min fairness*?

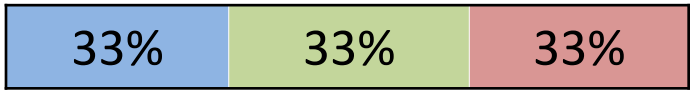
What is *max-min fairness*?

- *Maximize* the the user with *minimum* allocation
  - Solution: allocate each **1/n** (fair share)



What is *max-min fairness*?

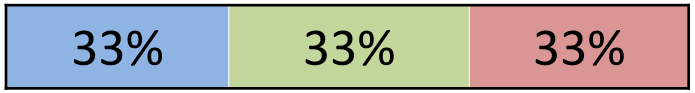
- *Maximize* the the user with *minimum* allocation
  - Solution: allocate each **1/n** (fair share)



  - Handles if some users want less than fair share

What is *max-min fairness*?

- *Maximize* the the user with *minimum* allocation
  - Solution: allocate each **1/n** (fair share)

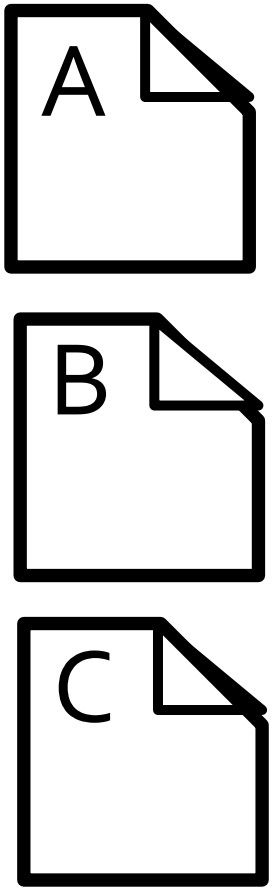


  - Handles if some users want less than fair share

- Widely successful to other resources:
  - OS: round robin, prop sharing, lottery sched…
  - Networking: fair queueing, wfq, wf2q, csfq, drr…
  - Datacenter: DRF, Hadoop fair sched, Quincy…

An example

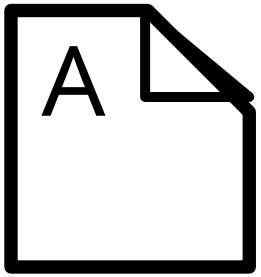
An example

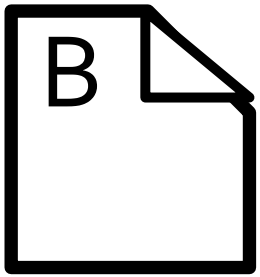


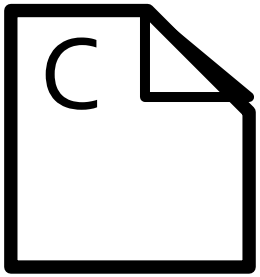
An example

Alice

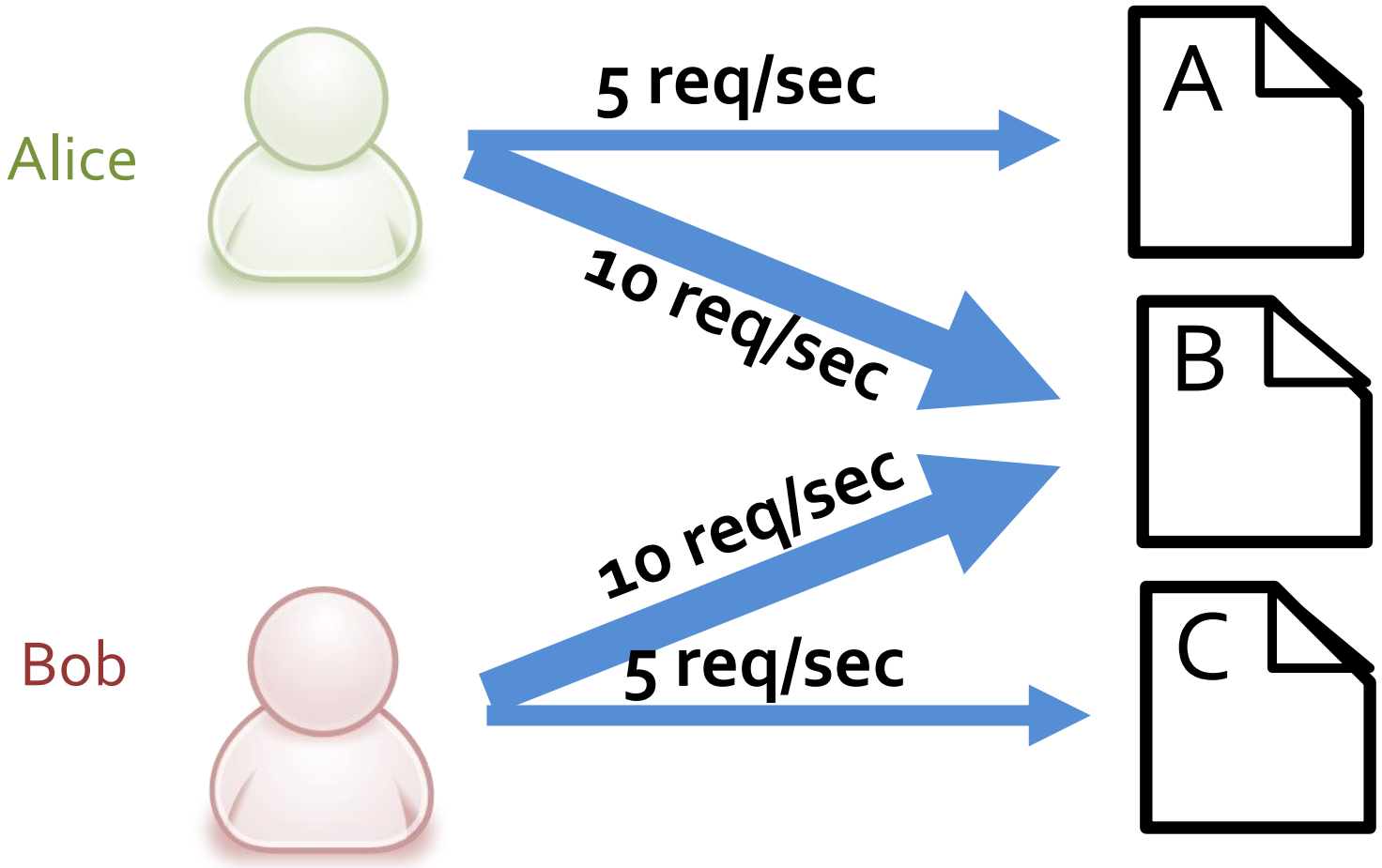
Bob



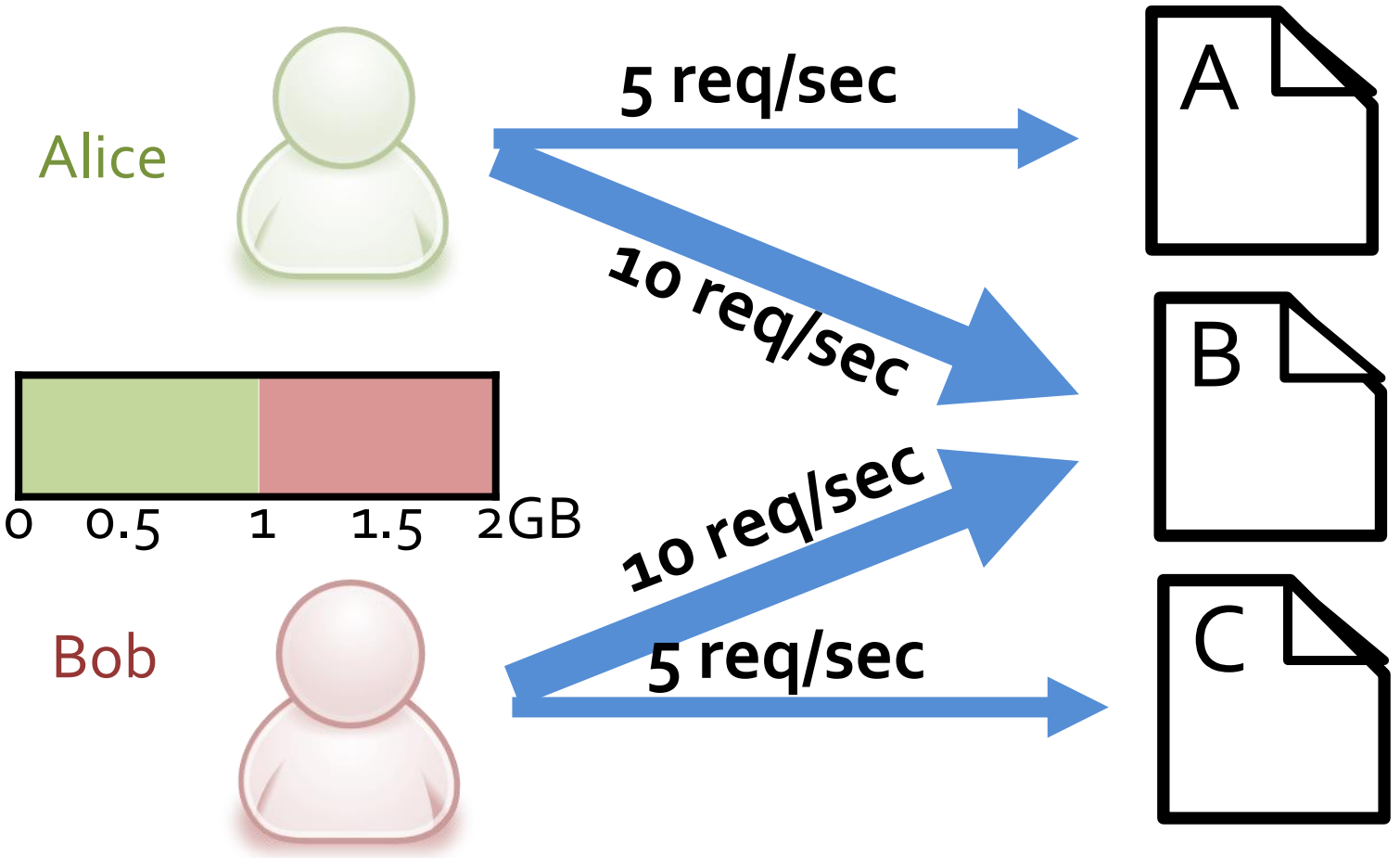




An example

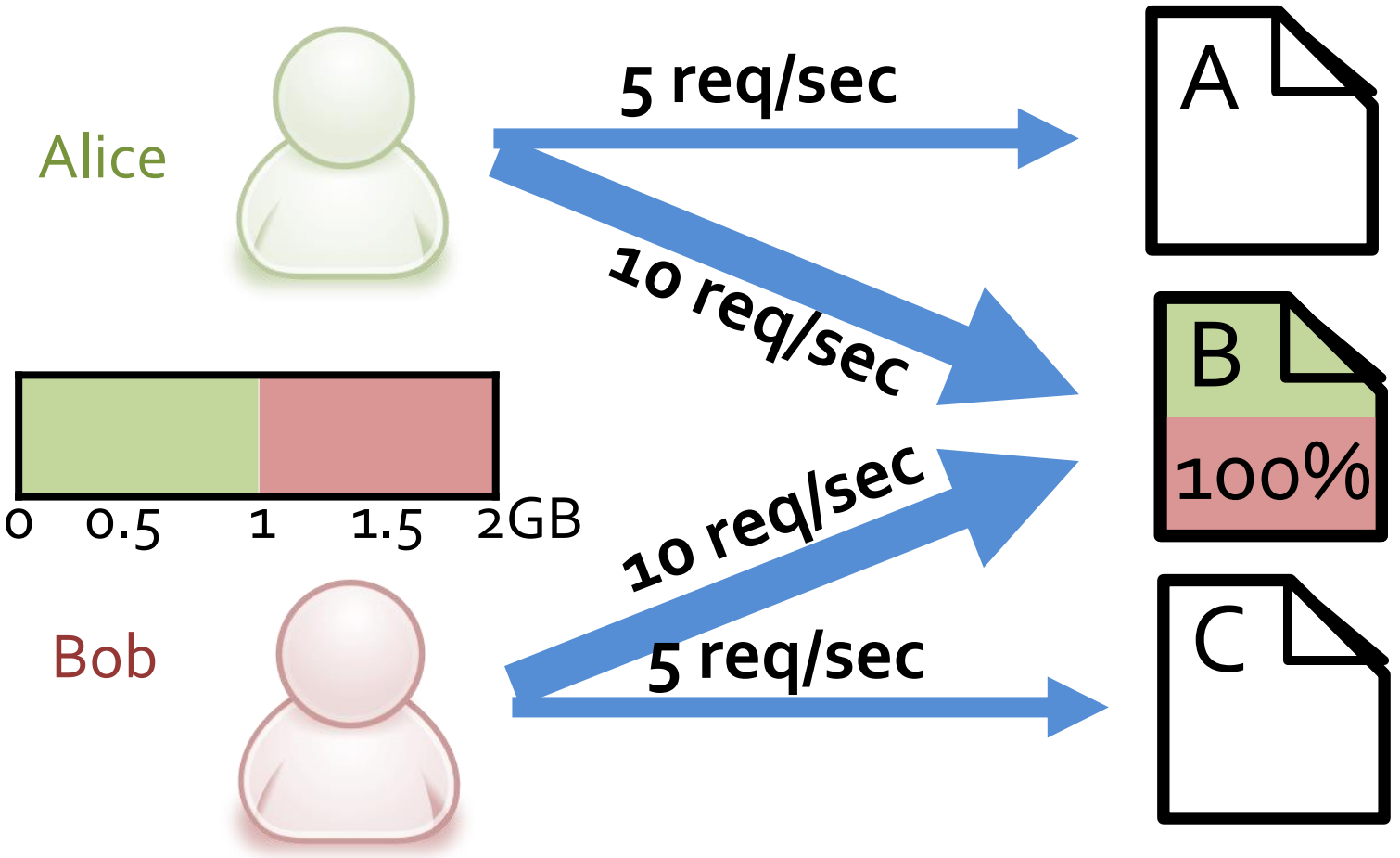


An example



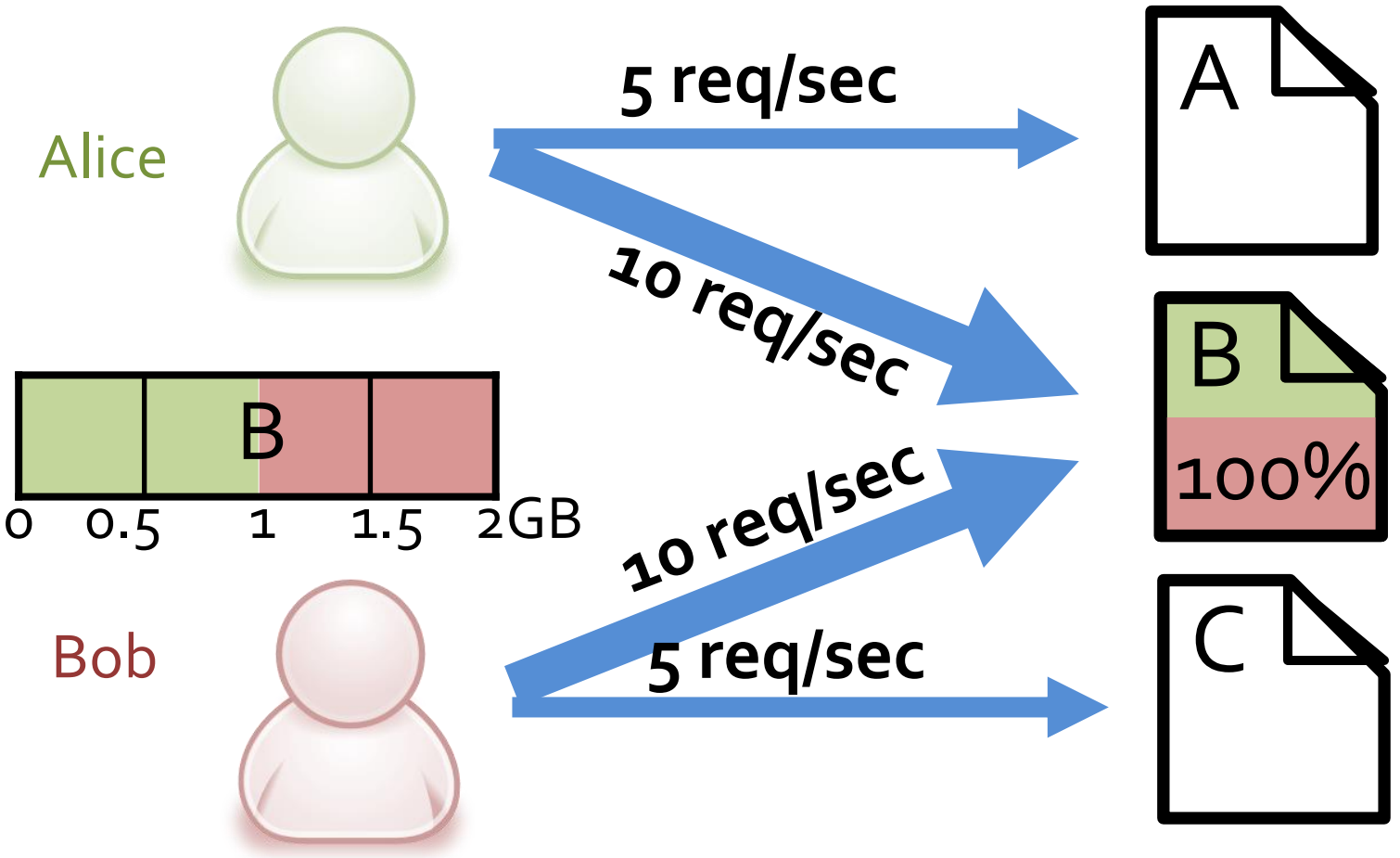
file sizes = 1GB, total cache = 2GB

An example



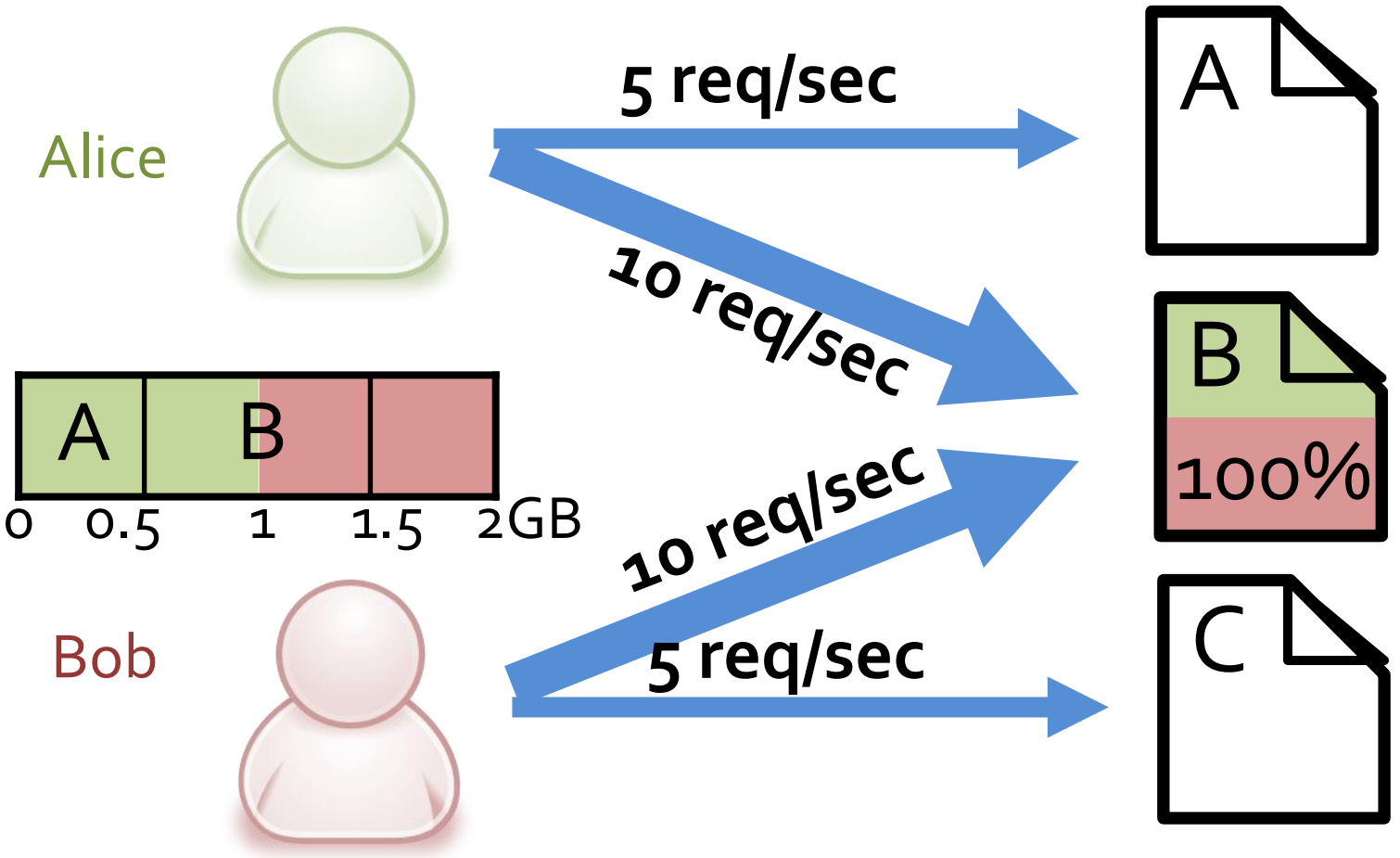
file sizes = 1GB, total cache = 2GB

An example



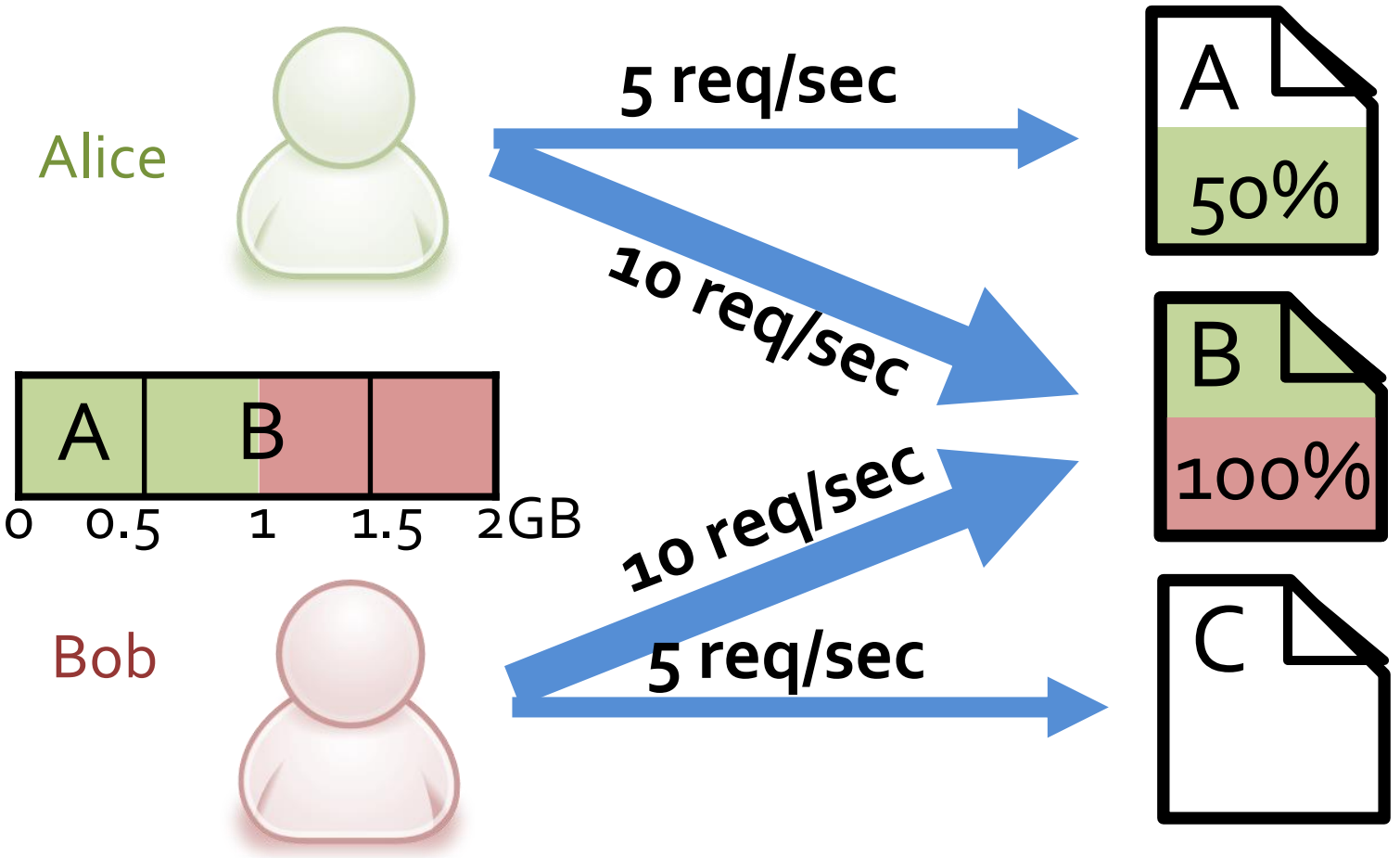
file sizes = 1GB, total cache = 2GB

An example



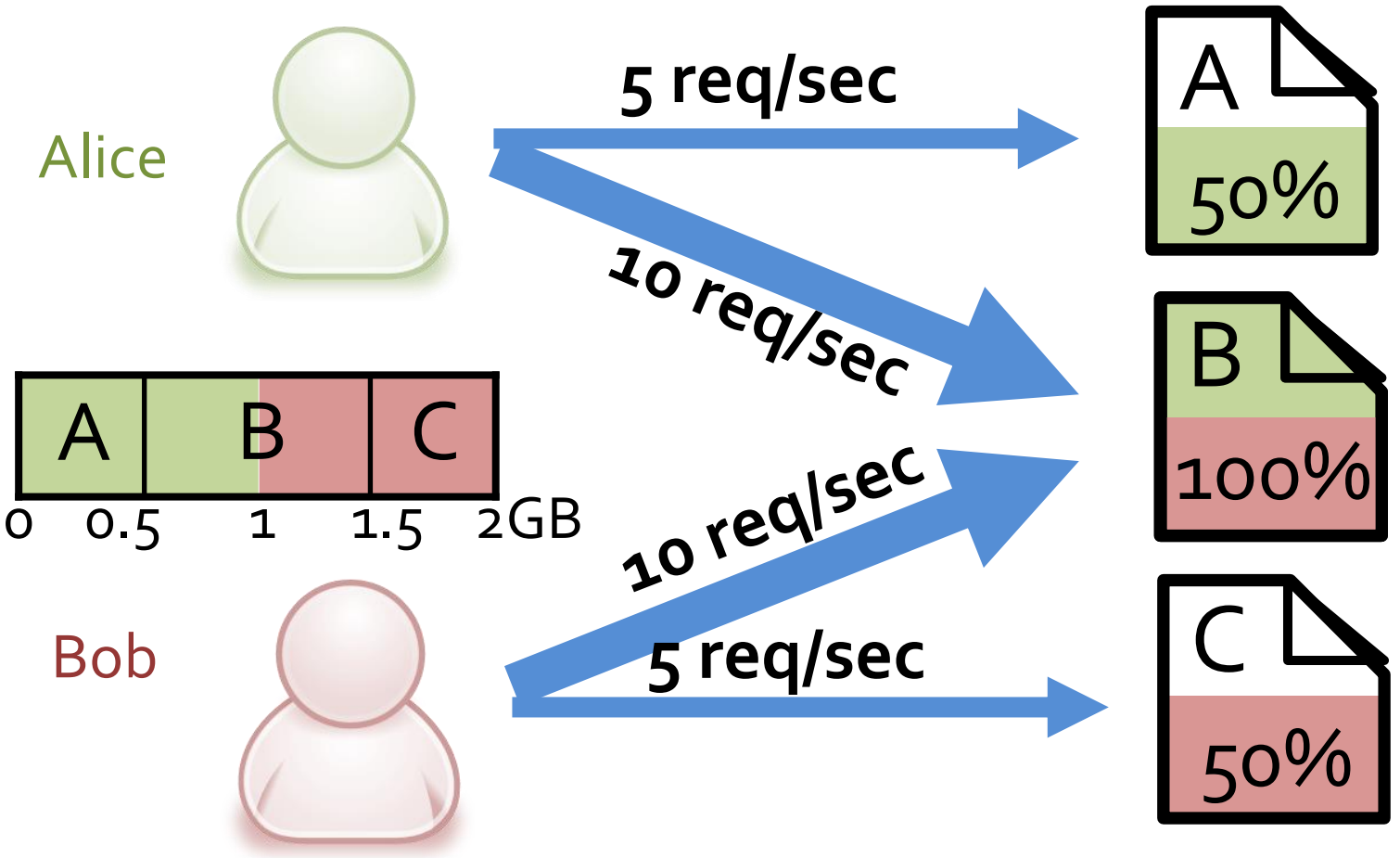
file sizes = 1GB, total cache = 2GB

An example



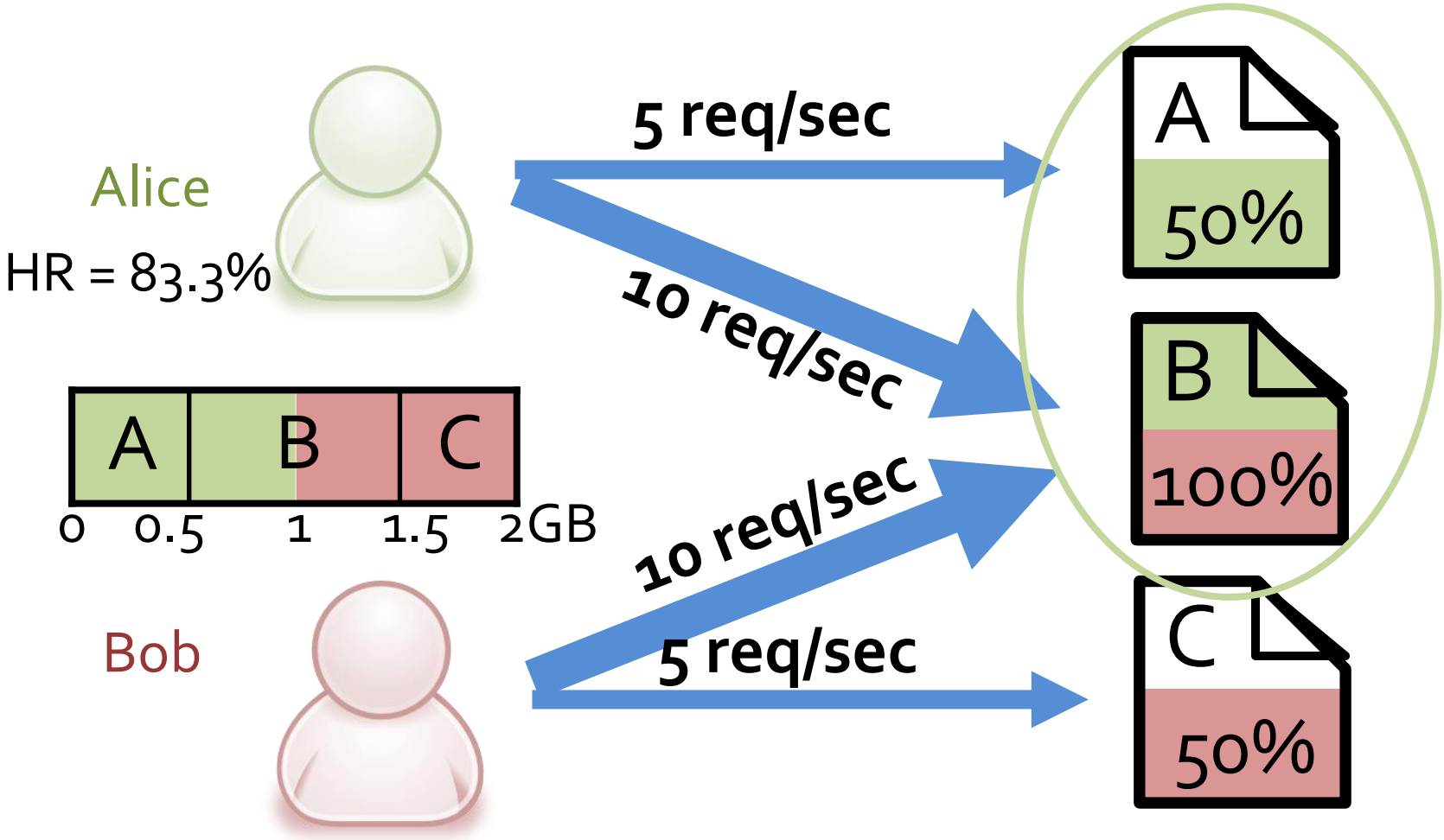
file sizes = 1GB, total cache = 2GB

An example



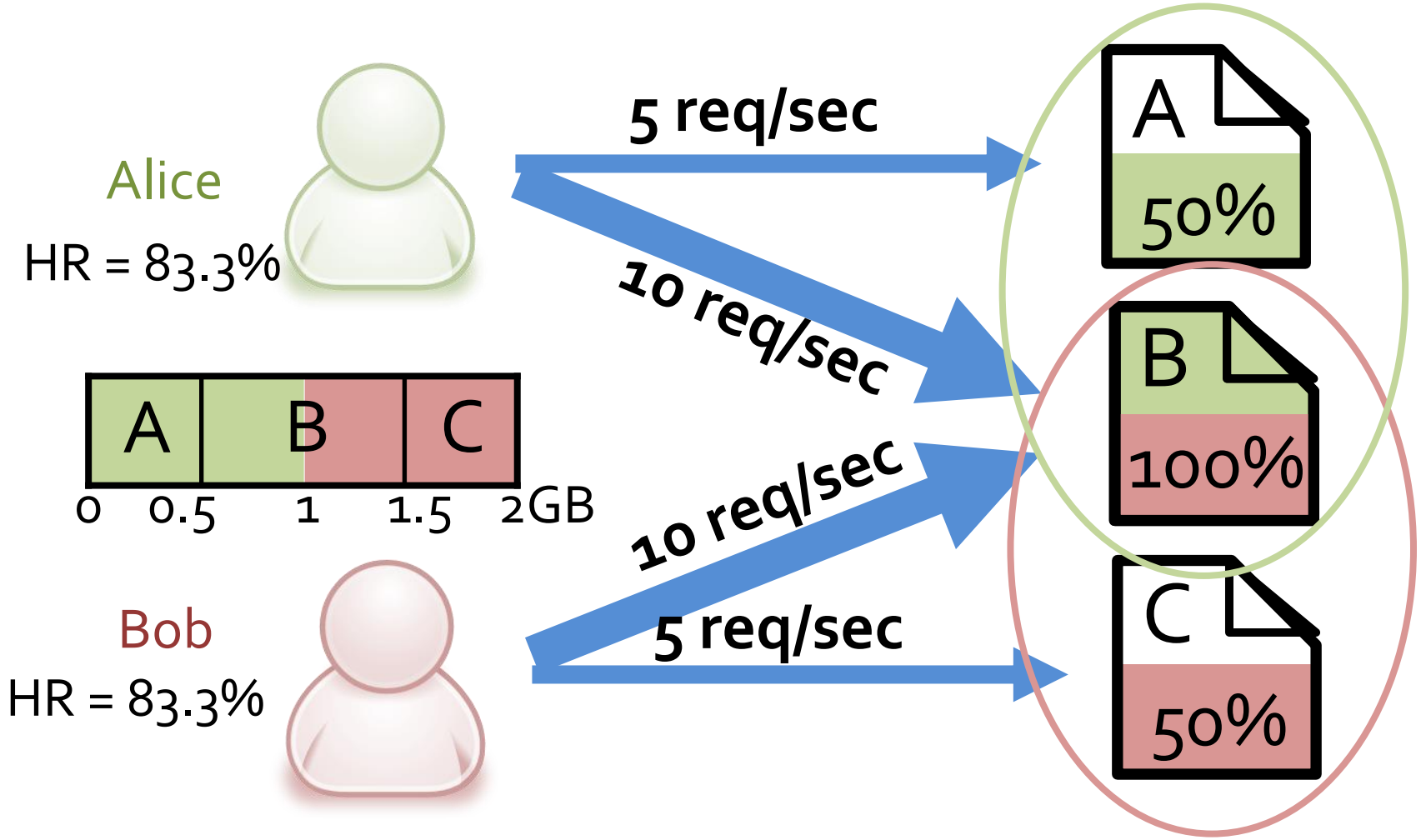
file sizes = 1GB, total cache = 2GB

An example



file sizes = 1GB, total cache = 2GB

An example



file sizes = 1GB, total cache = 2GB

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness			

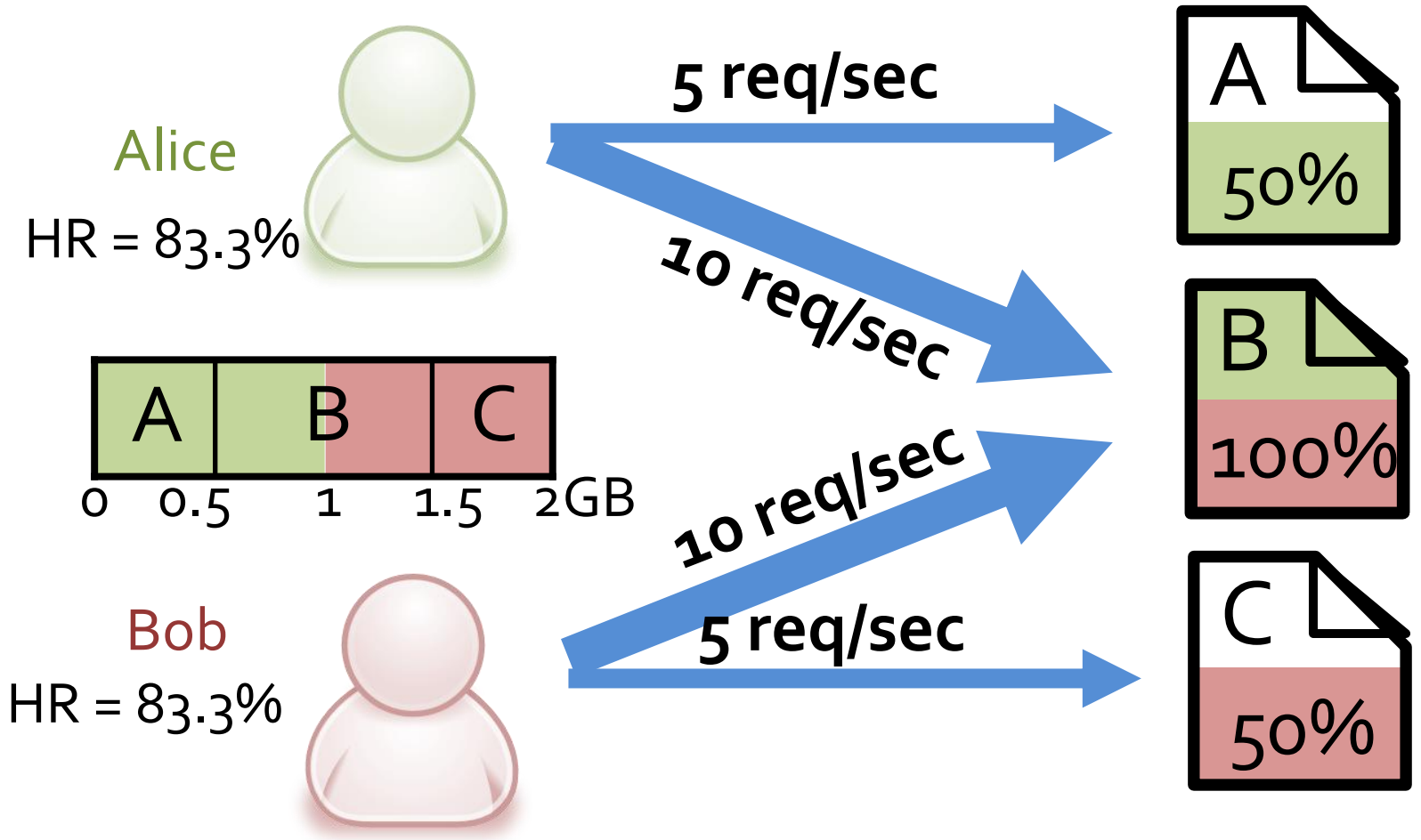
Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓		✓

Properties

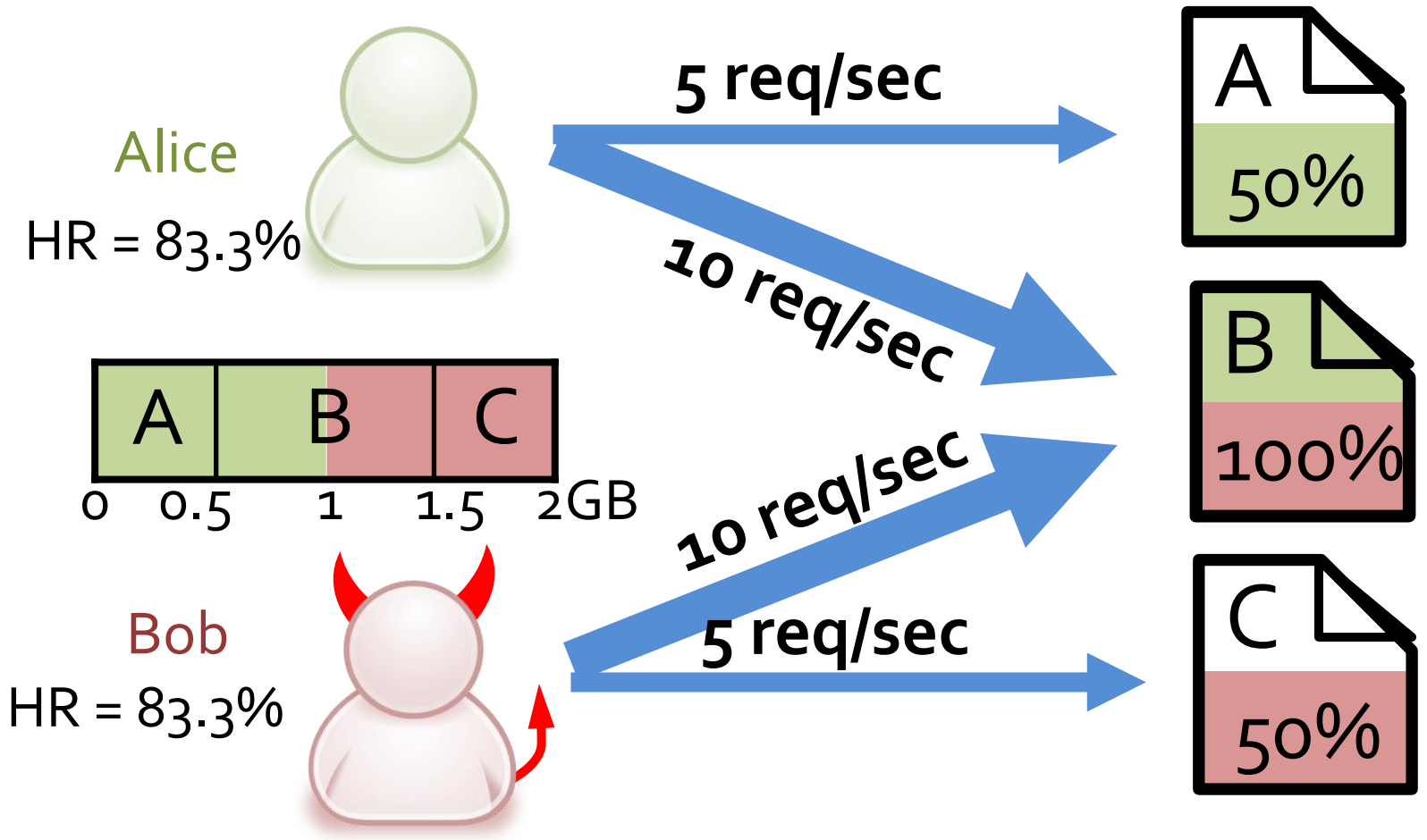
	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	?	✓

An example



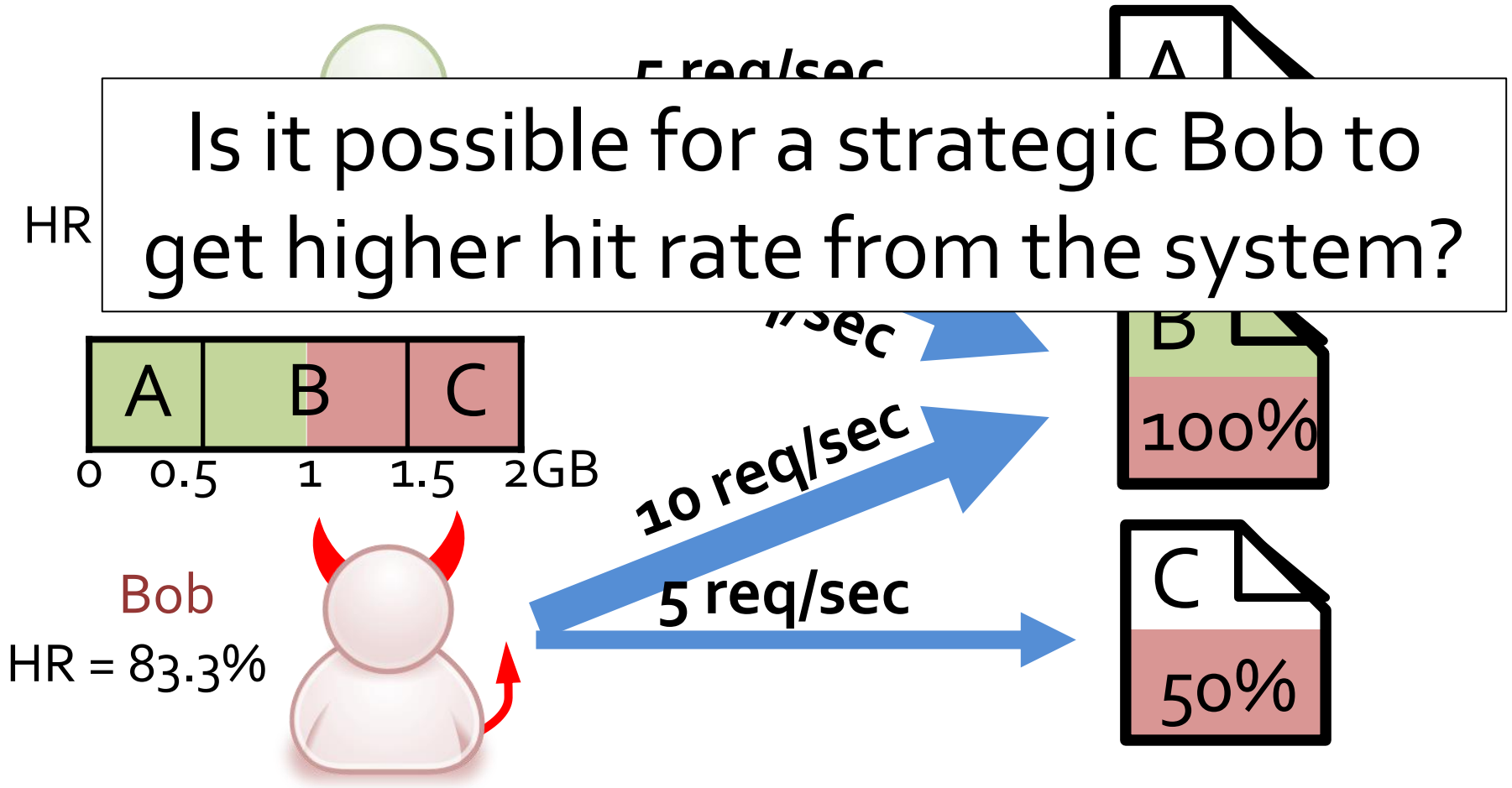
file sizes = 1GB, total cache = 2GB

An example



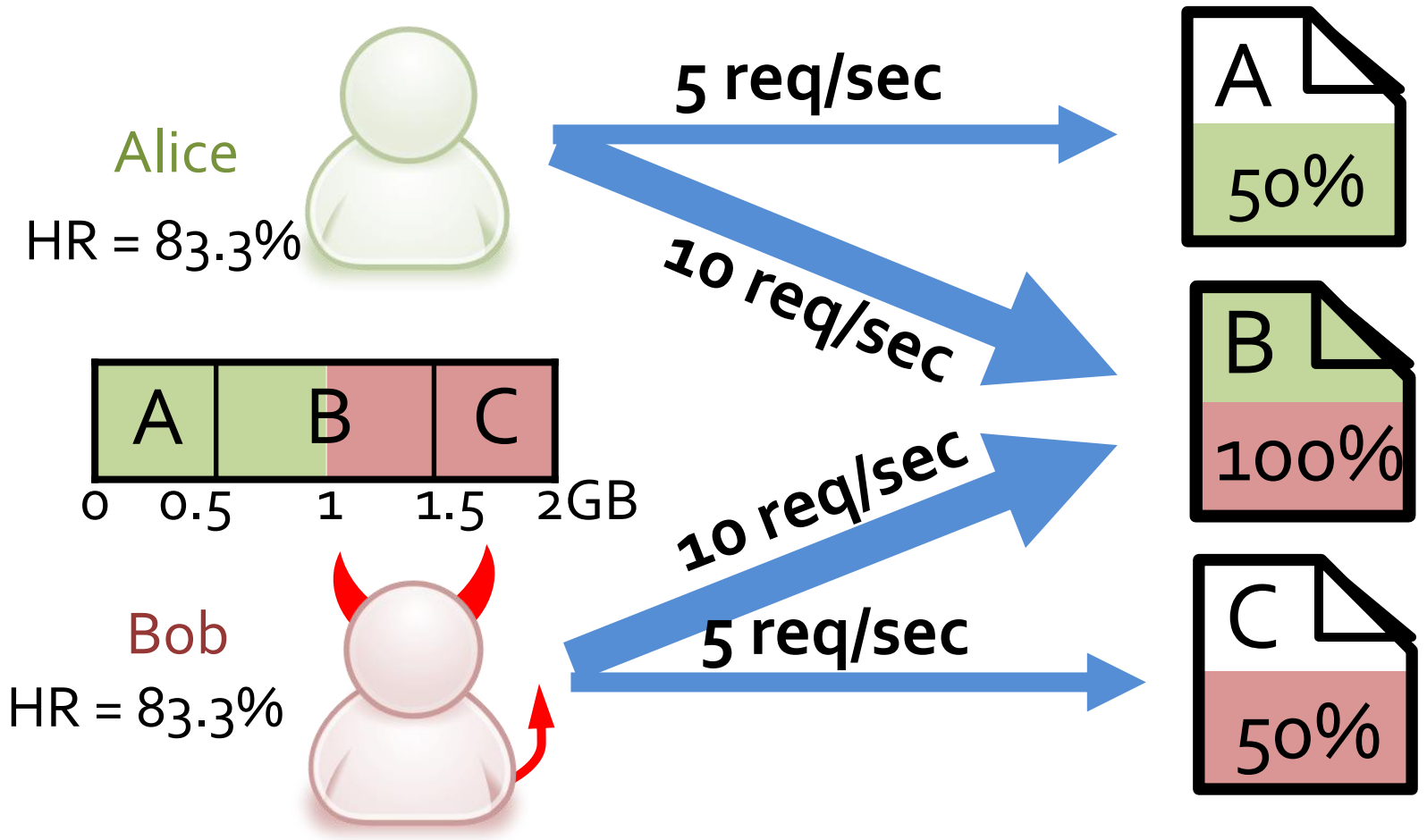
file sizes = 1GB, total cache = 2GB

An example



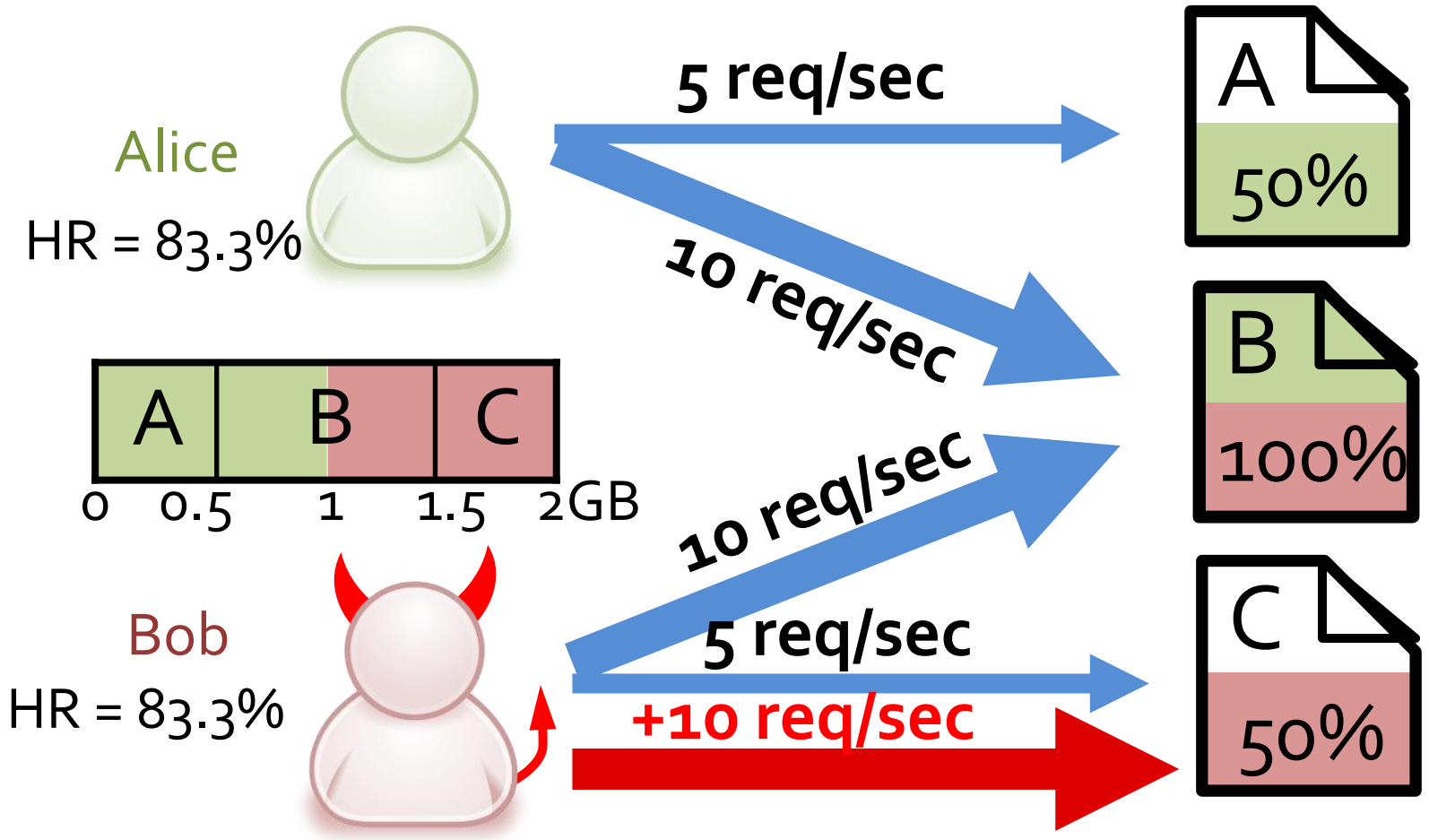
file sizes = 1GB, total cache = 2GB

An example



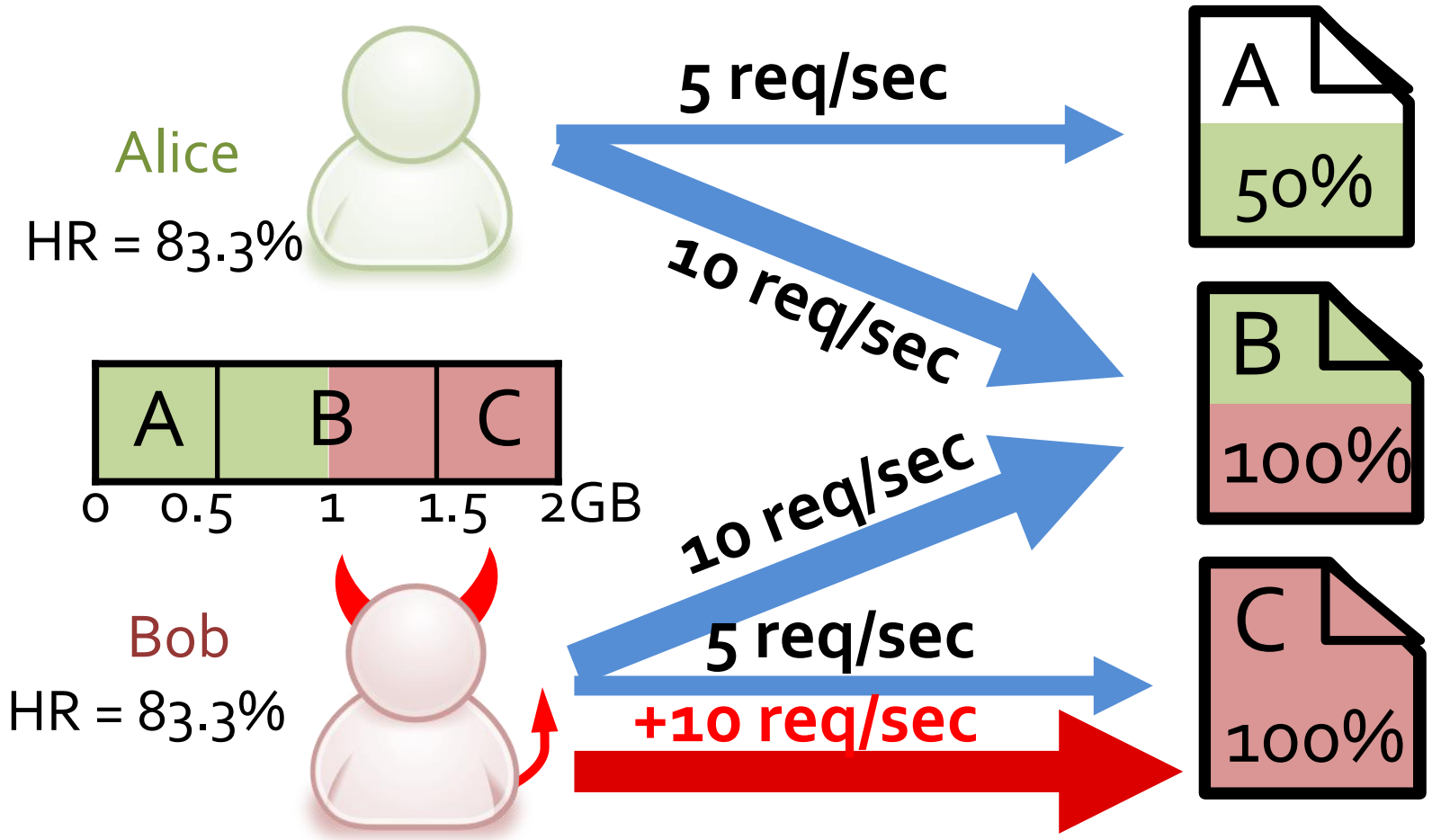
file sizes = 1GB, total cache = 2GB

An example



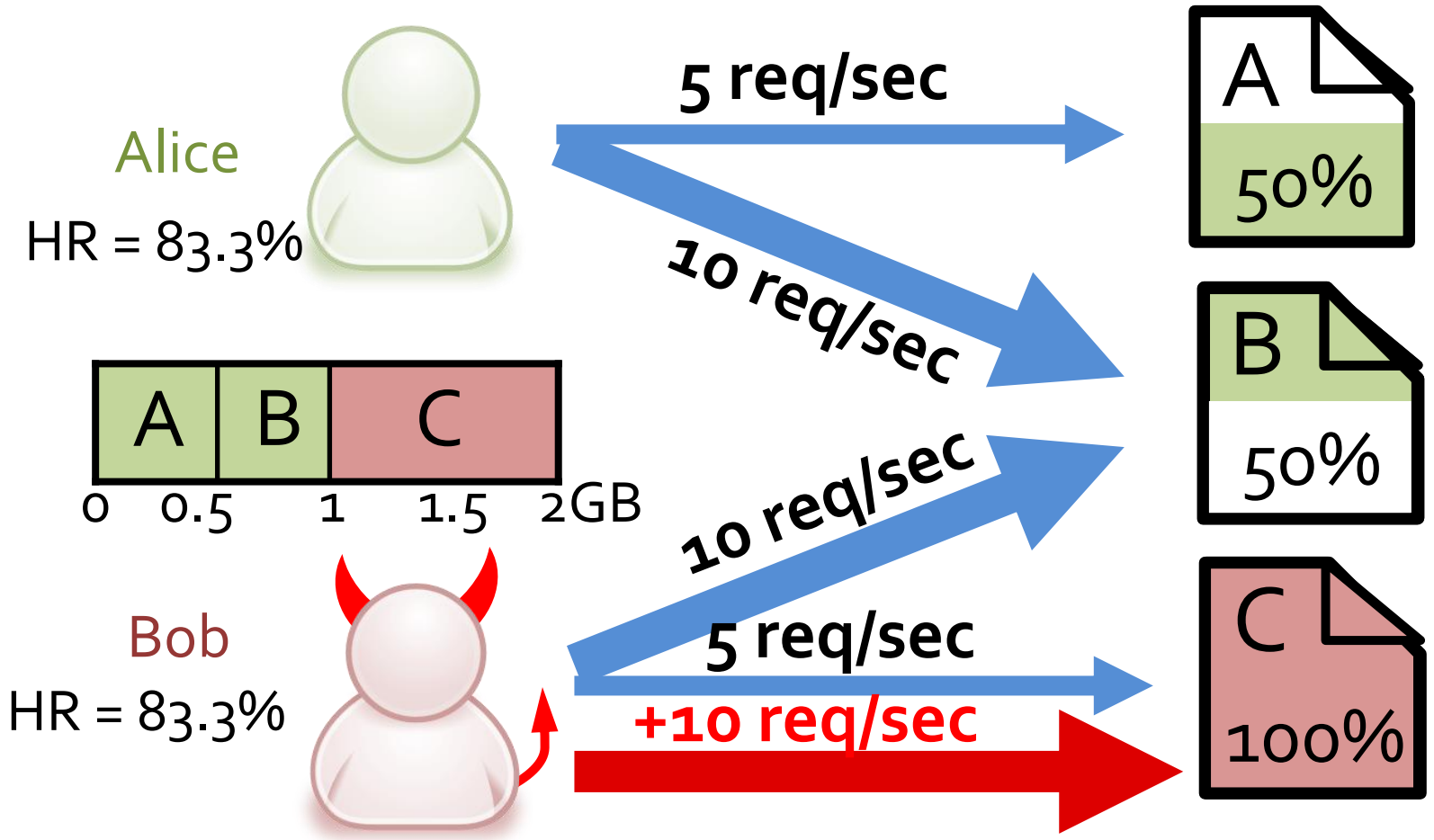
file sizes = 1GB, total cache = 2GB

An example



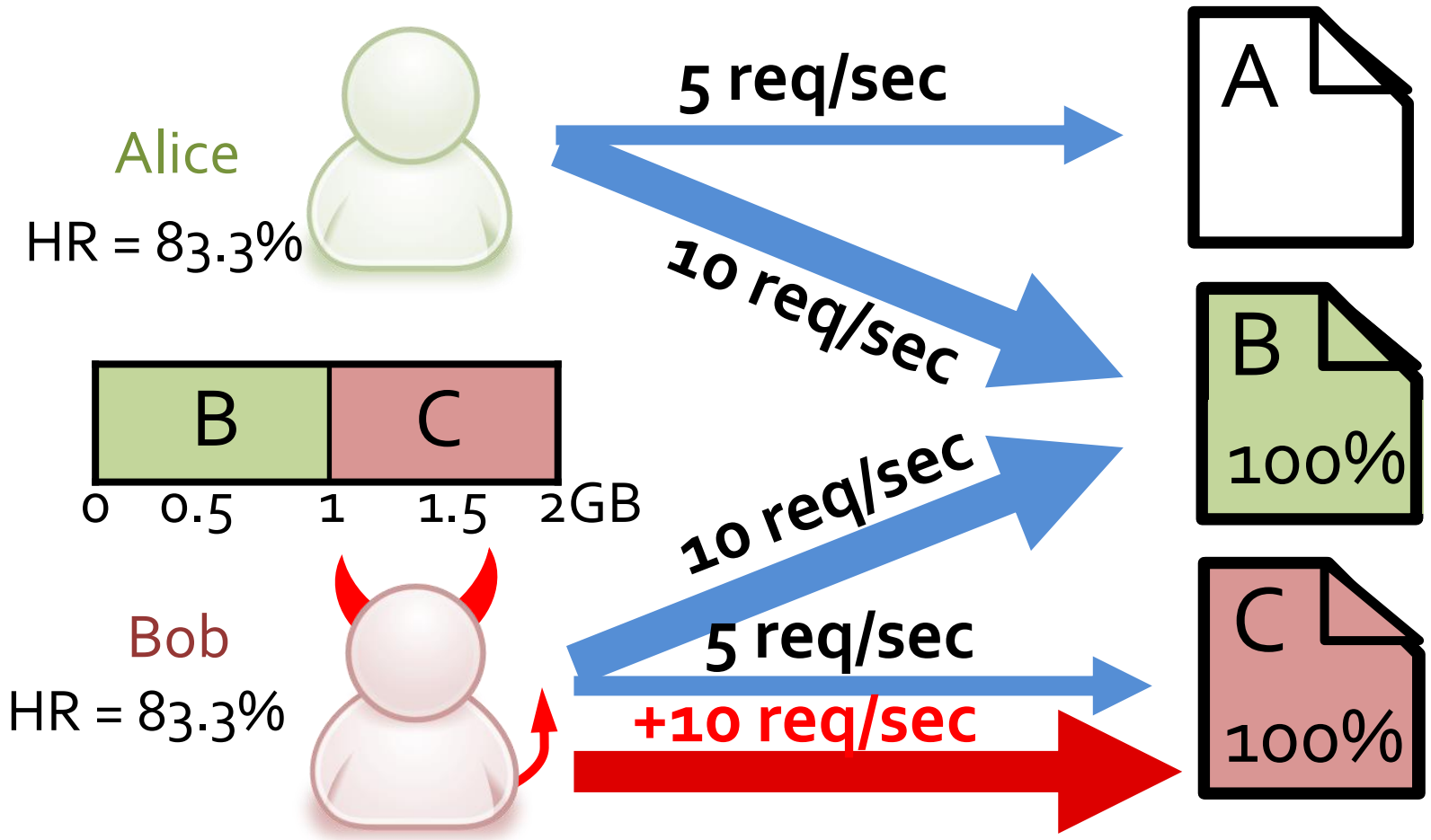
file sizes = 1GB, total cache = 2GB

An example



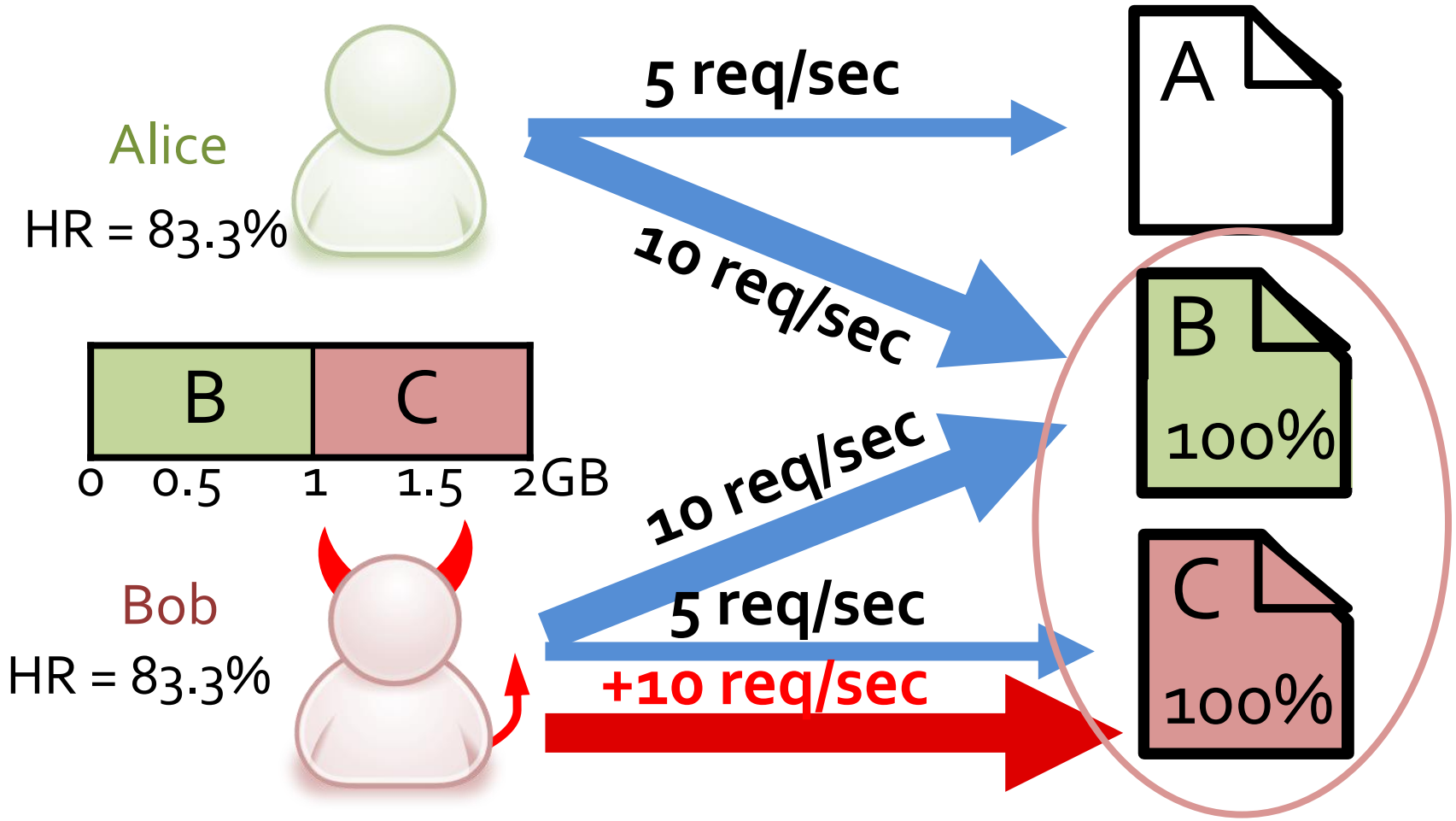
file sizes = 1GB, total cache = 2GB

An example



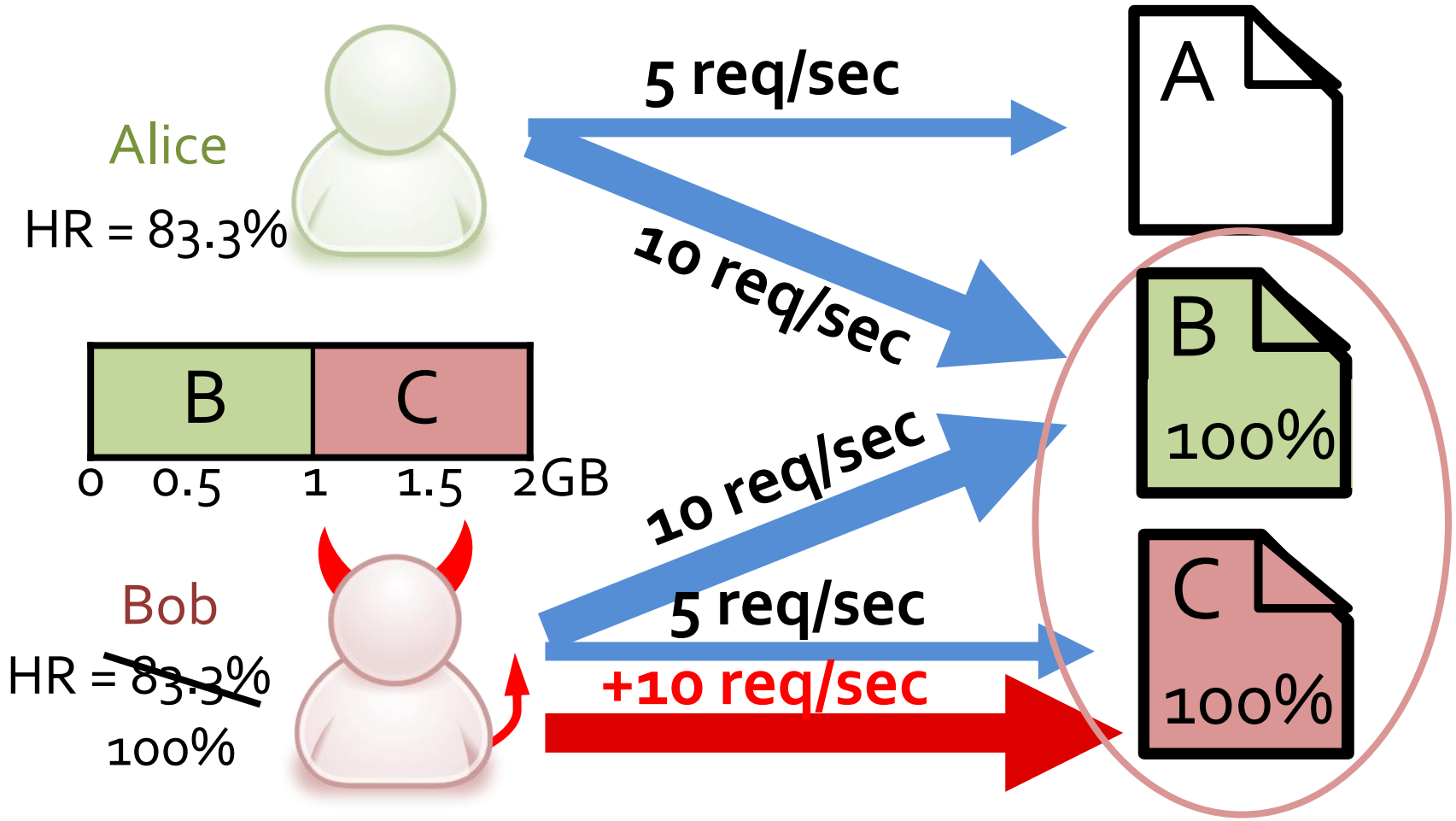
file sizes = 1GB, total cache = 2GB

An example



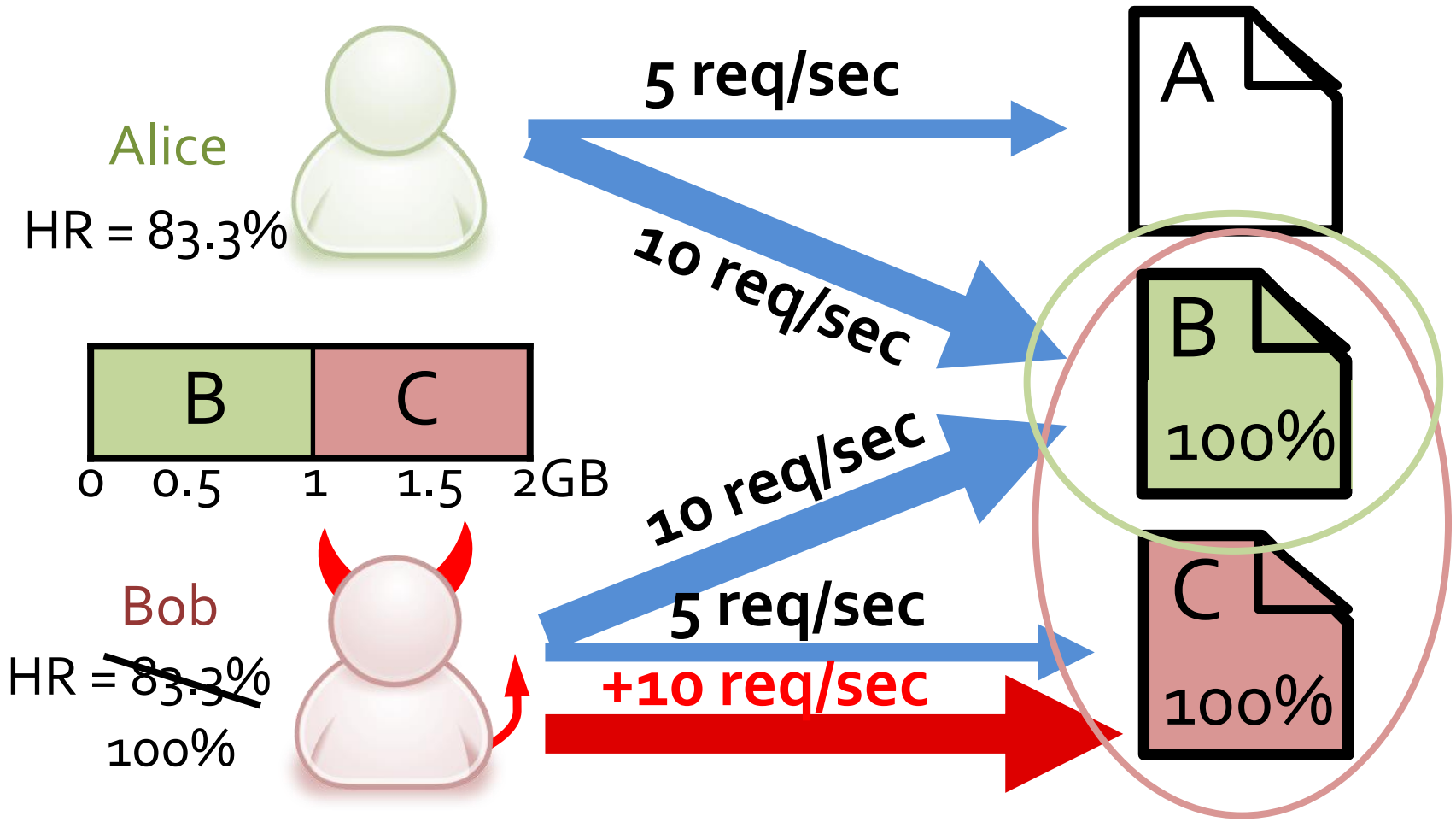
file sizes = 1GB, total cache = 2GB

An example



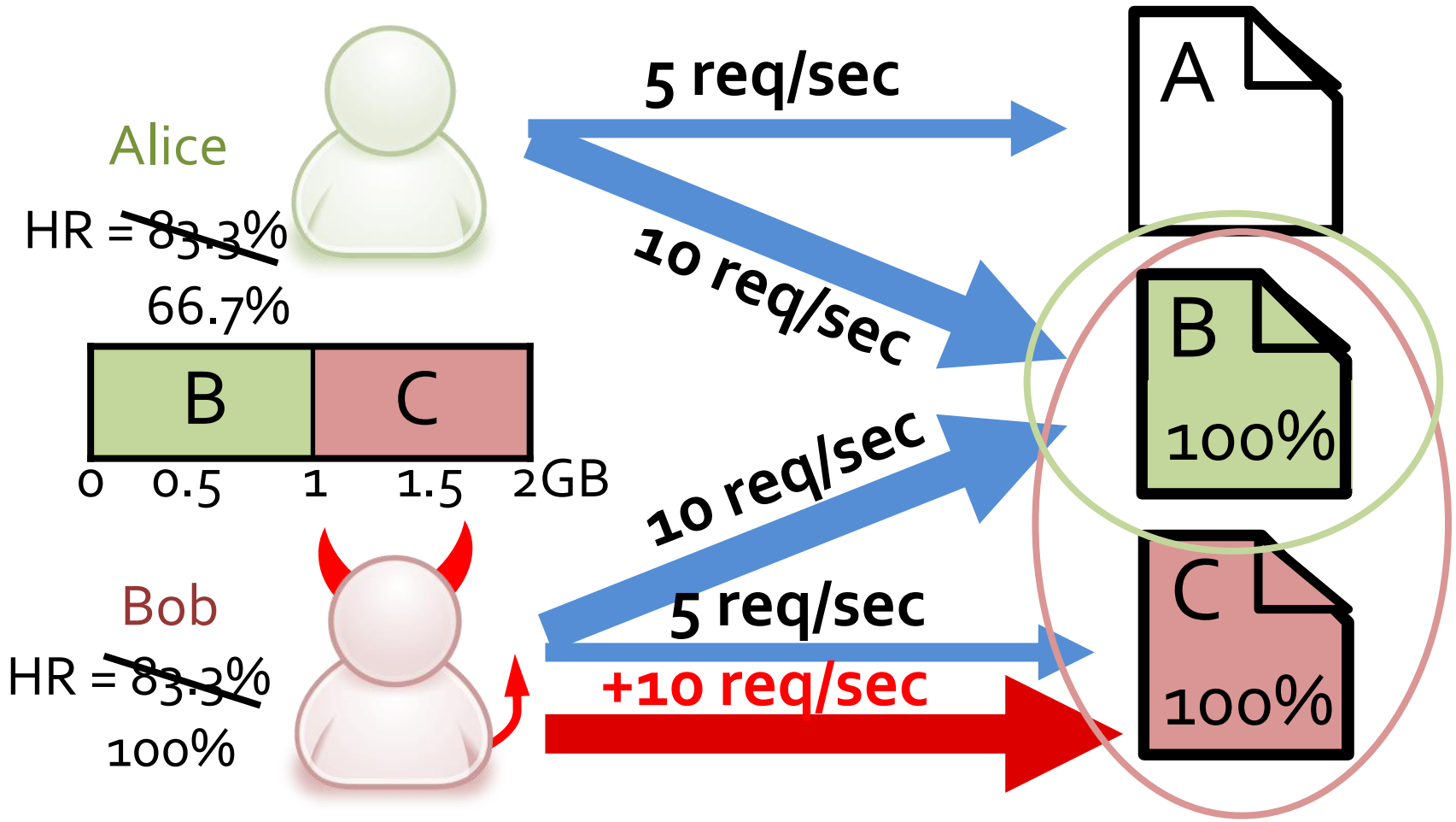
file sizes = 1GB, total cache = 2GB

An example



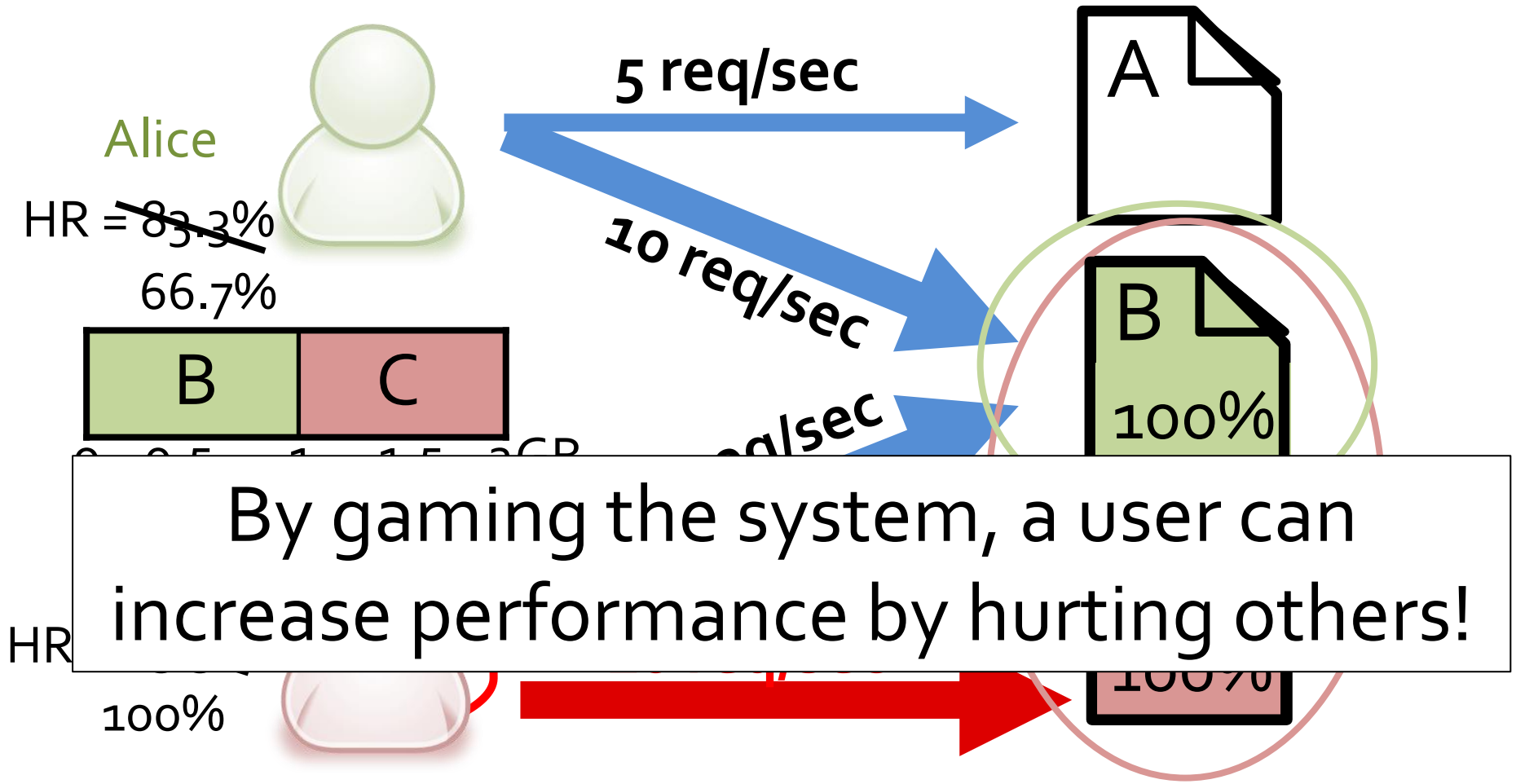
file sizes = 1GB, total cache = 2GB

An example



file sizes = 1GB, total cache = 2GB

An example



By gaming the system, a user can increase performance by hurting others!

file sizes = 1GB, total cache = 2GB

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓		✓

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓
static allocation	✓	✓	✗

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓
static allocation	✓	✓	✗
priority allocation	✗	✓	✓

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓
static allocation	✓	✓	✗
priority allocation	✗	✓	✓
max-min rate	✗	✓	✗
…	…	…	…

Theorem

No allocation policy can satisfy **all three** properties!

Theorem

No allocation policy can satisfy **all three** properties!

- Best we can do: two of three.

What makes cache sharing different?

What makes cache sharing different?

Unlike CPU or network links:

What makes cache sharing different?

Unlike CPU or network links:

- The cost of "switching" is high
  - Cache misses go to slow storage
  - Prevents efficient time multiplexing

What makes cache sharing different?

Unlike CPU or network links:

- The cost of "switching" is high
  - Cache misses go to slow storage
  - Prevents efficient time multiplexing

- Can be shared in space
  - **Shared data** can be accessed non-exclusively
  - A CPU cycle used by only one thread
  - A network link sends one packet at a time

Outline

- What properties do we want?
- Can we extend max-min to solve the problem?
- How do we solve it? (FairRide)
- How well does FairRide work in practice?

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓
static allocation	✓	✓	✗
priority allocation	✗	✓	✓
max-min rate	✗	✓	✗

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓
static allocation	✓	✓	✗
priority allocation	✗	✓	✓
max-min rate	✗	✓	✗
FairRide			

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓
static allocation	✓	✓	✗
priority allocation	✗	✓	✓
max-min rate	✗	✓	✗
FairRide	✓	✓	

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓
static allocation	✓	✓	✗
priority allocation	✗	✓	✓
max-min rate	✗	✓	✗
FairRide	✓	✓	Near-optimal

FairRide

FairRide

- Starts with max-min fairness
  - Allocate 1/n to each user
  - Split "cost" of shared files equally among shared users

FairRide

- Starts with max-min fairness
  - Allocate 1/n to each user
  - Split "cost" of shared files equally among shared users

- Only difference:

  blocking users who don't "pay" from accessing

FairRide

- Starts with max-min fairness
  - Allocate 1/n to each user
  - Split "cost" of shared files equally among shared users

- Only difference:

  blocking users who don't "pay" from accessing

- Probabilistic blocking: with some probability

FairRide

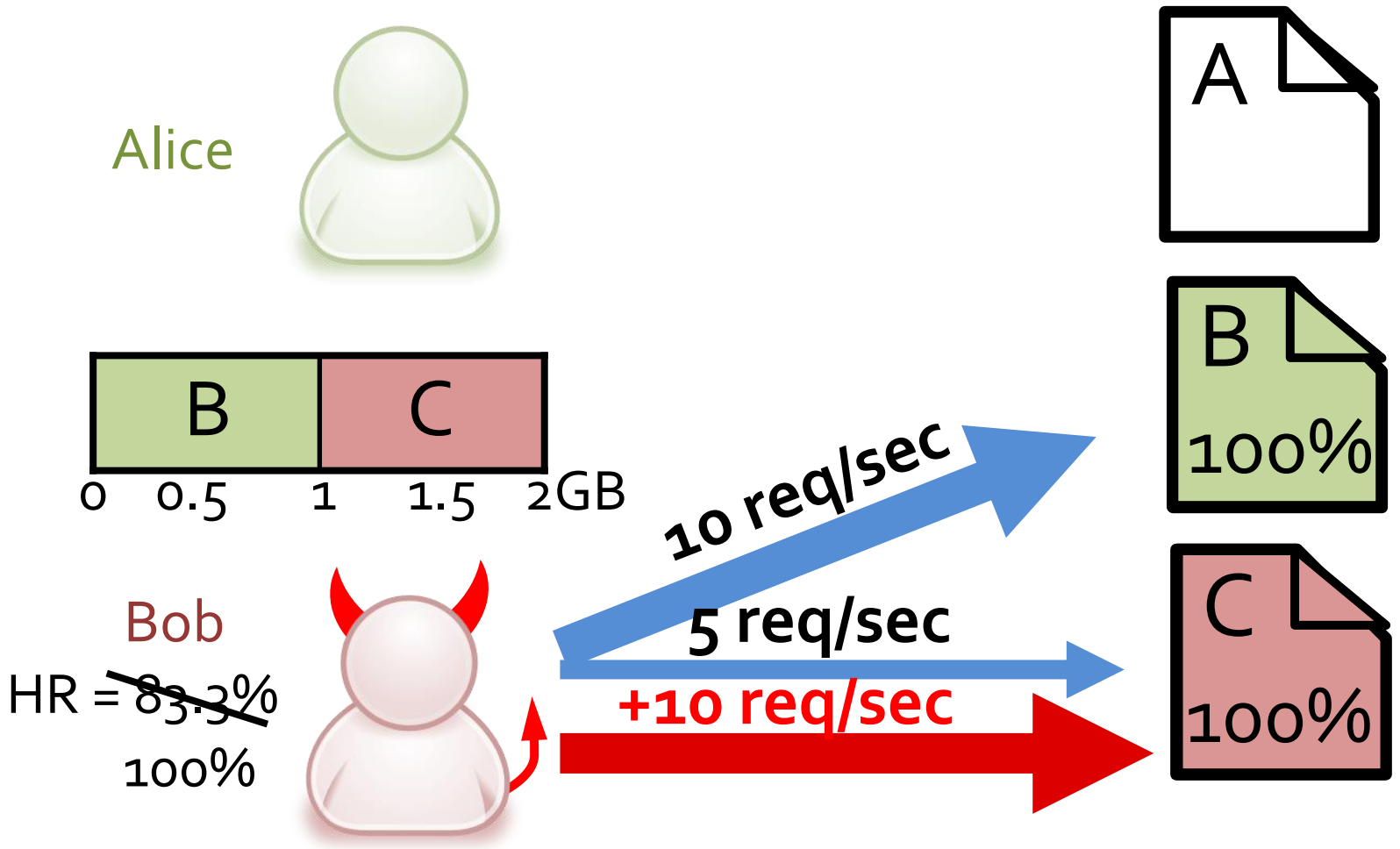
- Starts with max-min fairness
  - Allocate 1/n to each user
  - Split "cost" of shared files equally among shared users

- Only difference:

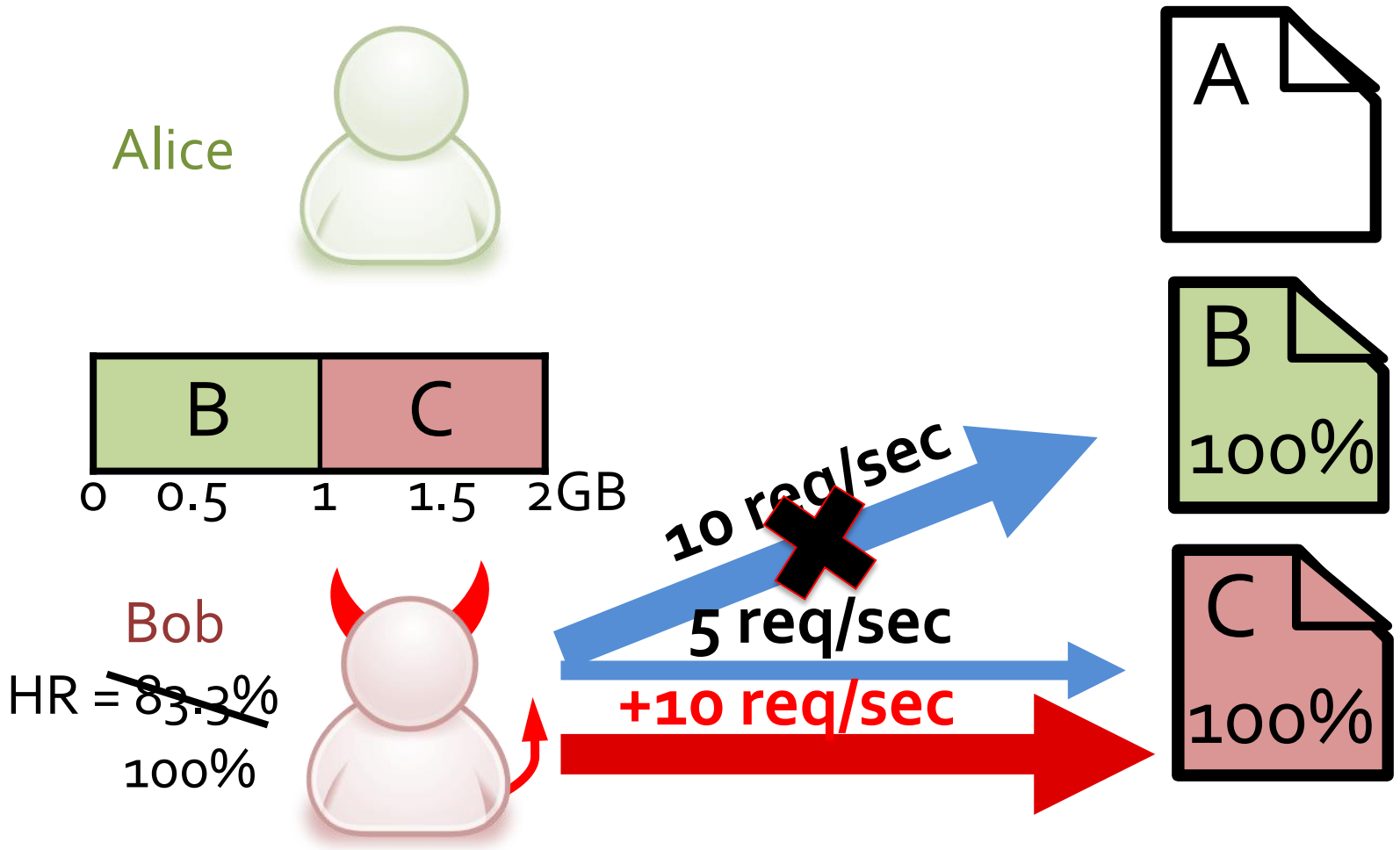
  blocking users who don't "pay" from accessing

- Probabilistic blocking: with some probability
  - Implemented with delaying

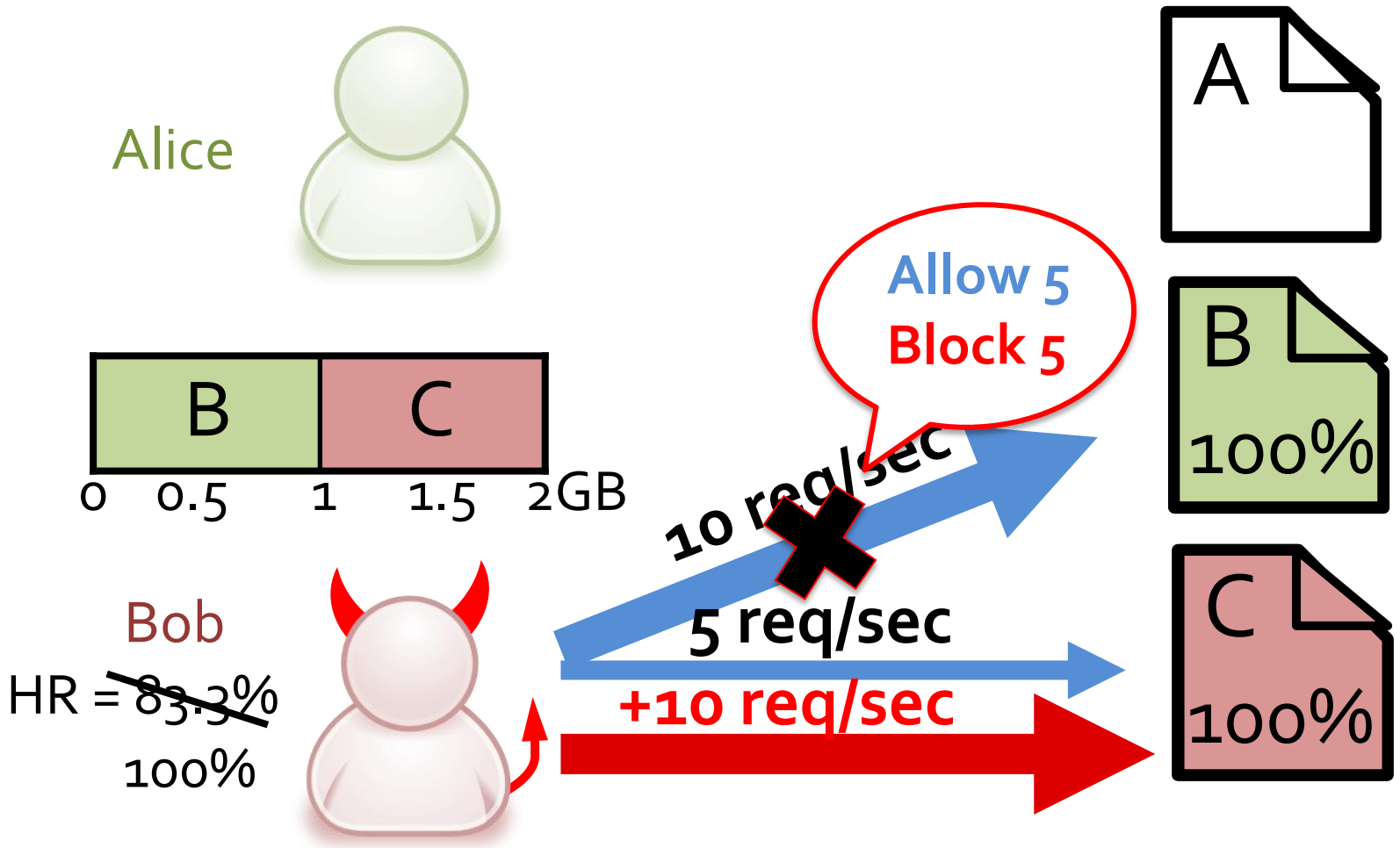
FairRide: Blocking



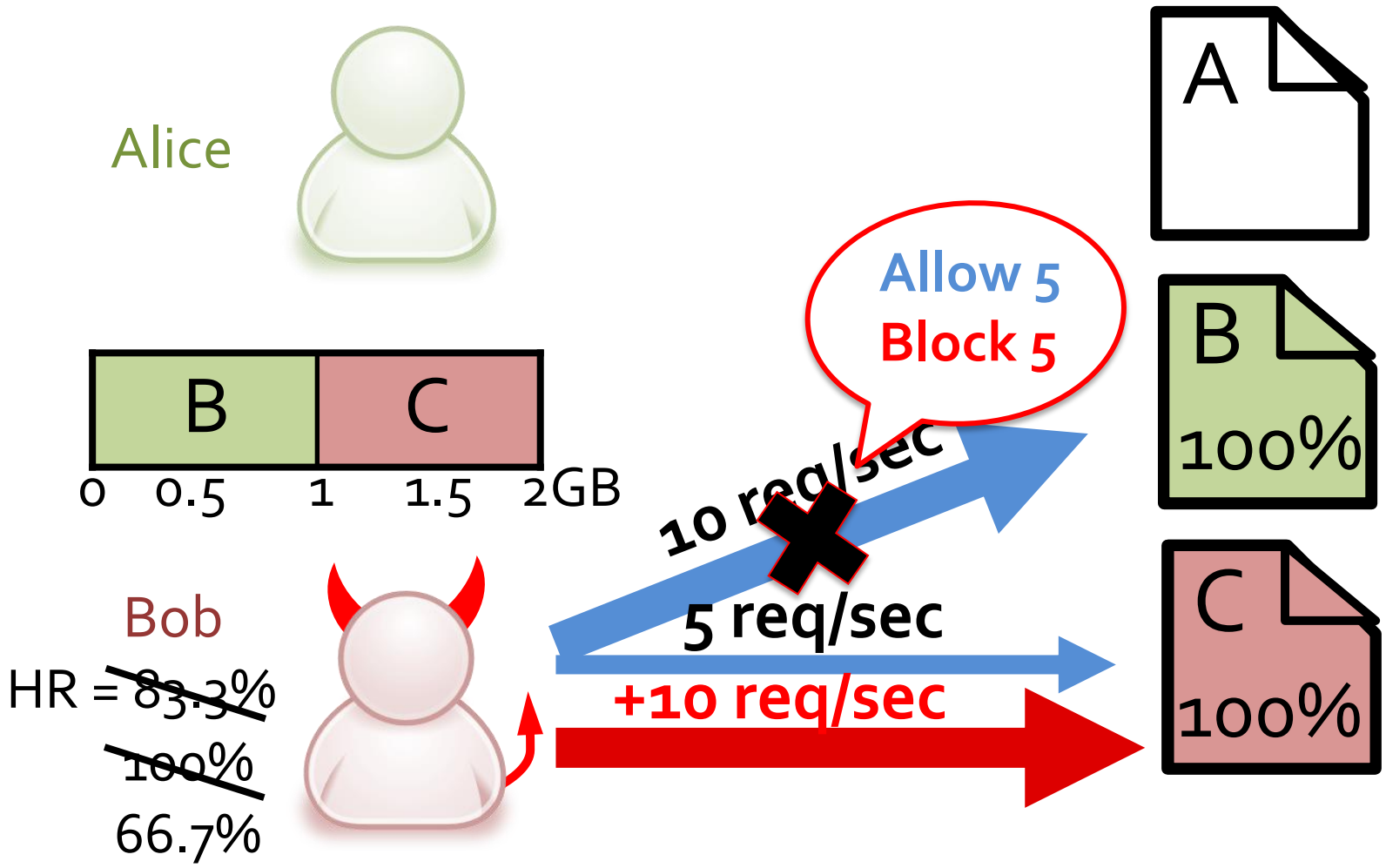
FairRide: Blocking



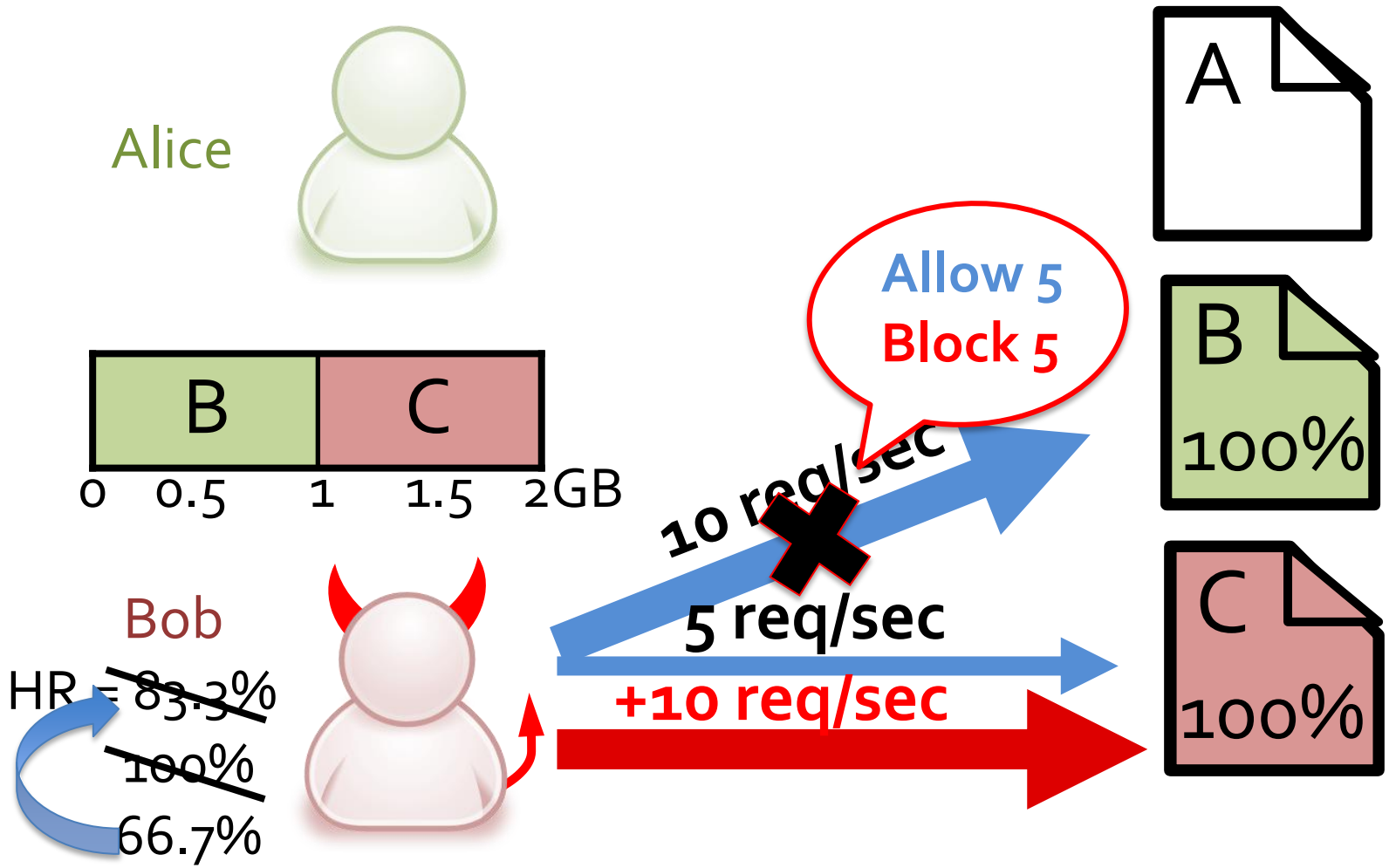
FairRide: Blocking



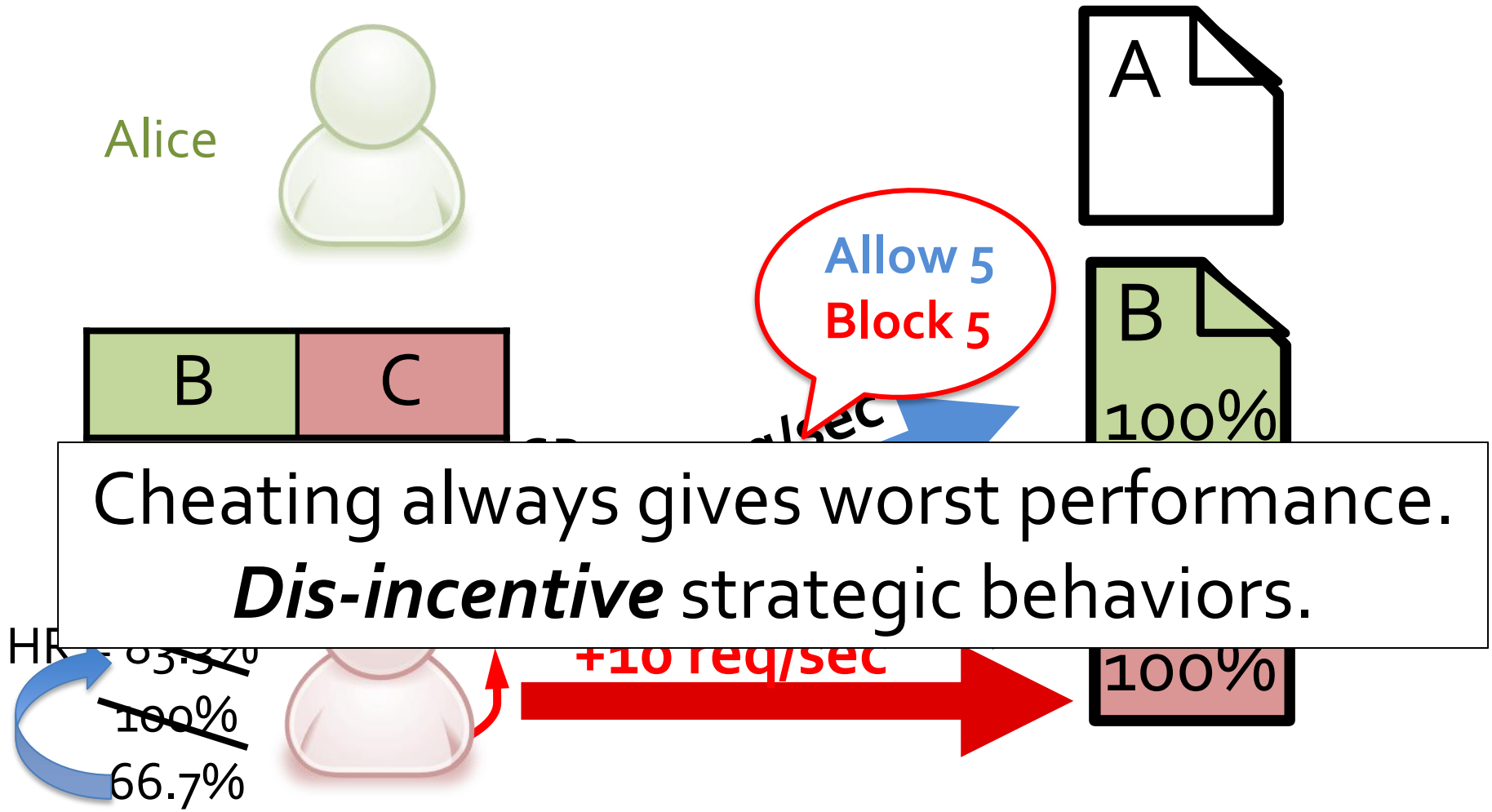
FairRide: Blocking



FairRide: Blocking



FairRide: Blocking



Cheating always gives worst performance.

Dis-incentive strategic behaviors.

Probabilistic blocking

Probabilistic blocking

- FairRide blocks a user with ***p(nj) = 1/(nj+1)*** probability
  - *nj* is number of other users caching file *j*
  - e.g., p(1)=50%, p(4)=20%

Probabilistic blocking

- FairRide blocks a user with ***p(nj) = 1/(nj+1)*** probability
  - *nj* is number of other users caching file *j*
  - e.g., p(1)=50%, p(4)=20%

- The best you can do in a general case

Probabilistic blocking

- FairRide blocks a user with ***p(nj) = 1/(nj+1)*** probability
  - *nj* is number of other users caching file *j*
  - e.g., p(1)=50%, p(4)=20%

- The best you can do in a general case
  - **Less blocking does not prevent cheating**

Properties

	Isolation Guarantee	Strategy Proofness	Pareto Efficiency
max-min fairness	✓	✗	✓
static allocation	✓	✓	✗
priority allocation	✗	✓	✓
max-min rate	✗	✓	✗
FairRide	✓	✓	Near-optimal

Strategy-proofness vs. Pareto-efficiency

Strategy-proofness vs. Pareto-efficiency

- More efficient when user cheats
  - Minimal impact on efficiency when no user cheats

Strategy-proofness vs. Pareto-efficiency

- More efficient when user cheats
  - Minimal impact on efficiency when no user cheats
- Cost of cheating vs. cost of blocking/delaying

Strategy-proofness vs. Pareto-efficiency

- More efficient when user cheats
  - Minimal impact on efficiency when no user cheats
- Cost of cheating vs. cost of blocking/delaying
  - The latter is small *insurance* for the former

Strategy-proofness vs. Pareto-efficiency

- More efficient when user cheats
  - Minimal impact on efficiency when no user cheats
- Cost of cheating vs. cost of blocking/delaying
  - The latter is small *insurance* for the former
- Strategy-proofness makes the system stable

Outline

- What properties do we want?
- Can we extend max-min to solve the problem?
- How do we solve it? (FairRide)
- How well does FairRide work in practice?

Evaluation

- Implemented in Alluxio (formerly Tachyon)

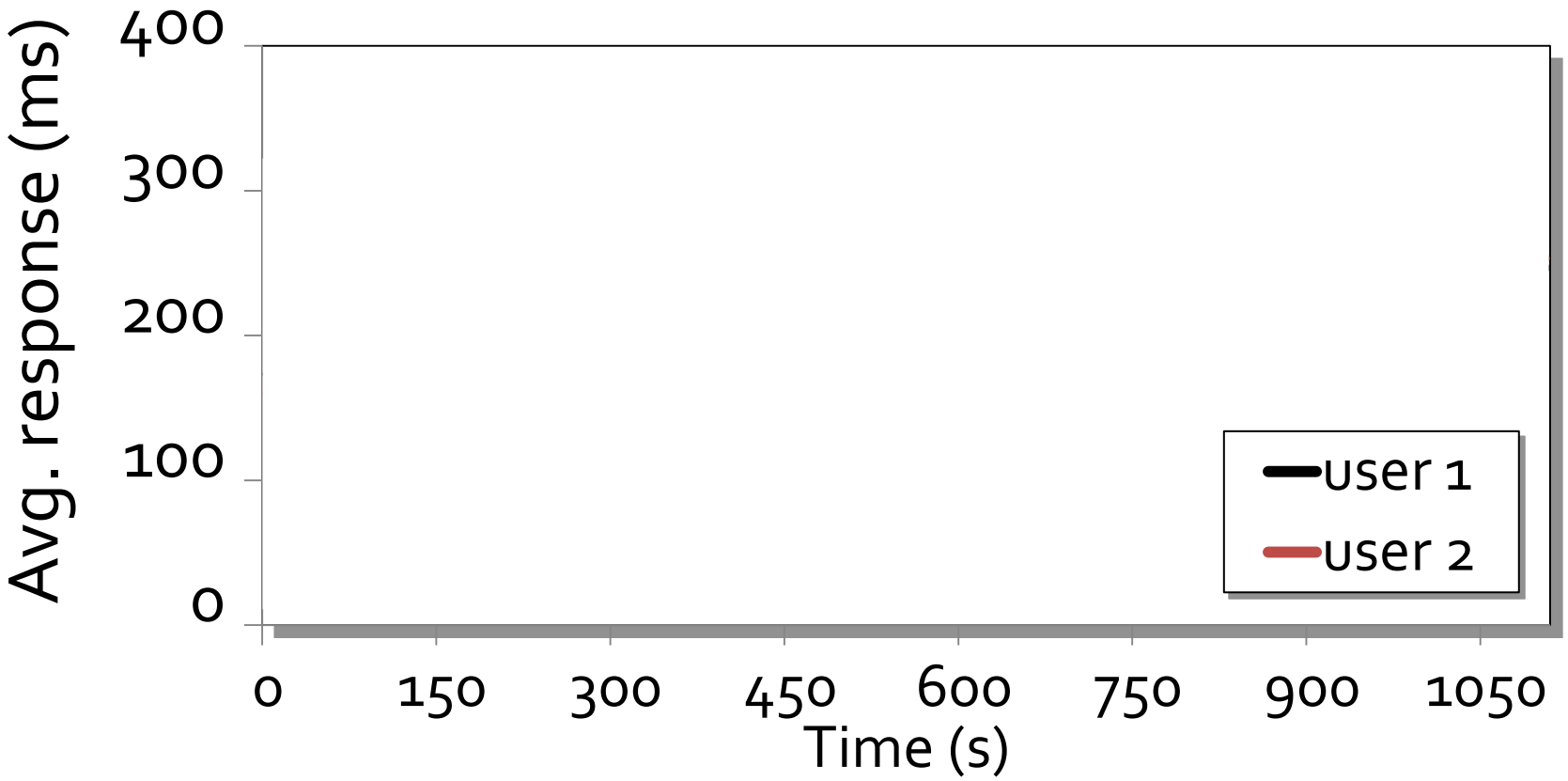
- FairRide: delay a request as if blocked
- Compared with max-min fairness.

- Benchmarked with TPC-H, YCSB, Facebook workloads.

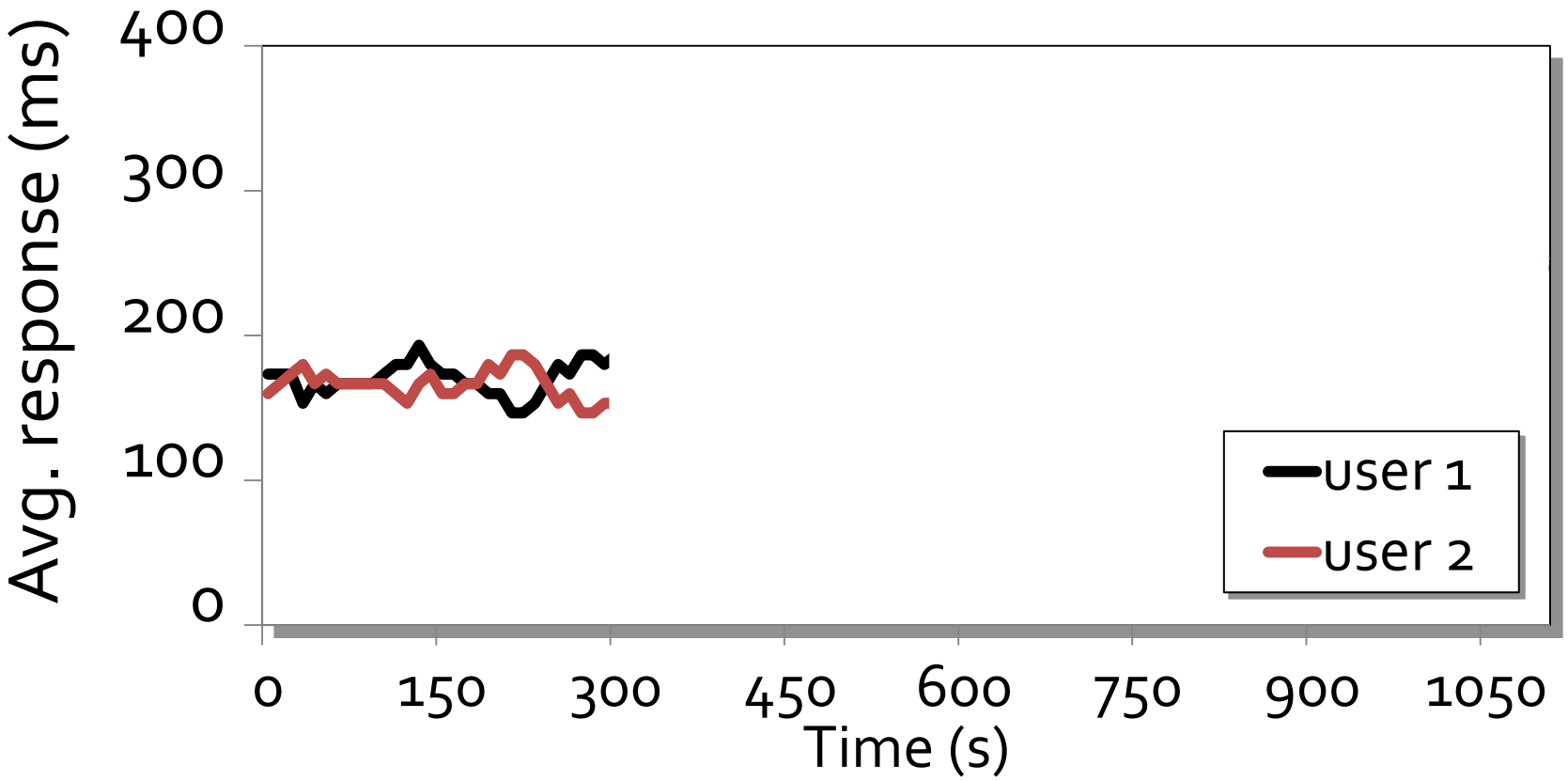
Evaluation

- Does FairRide prevent cheating?
- What is the cost of FairRide?
- How does it perform end-to-end?

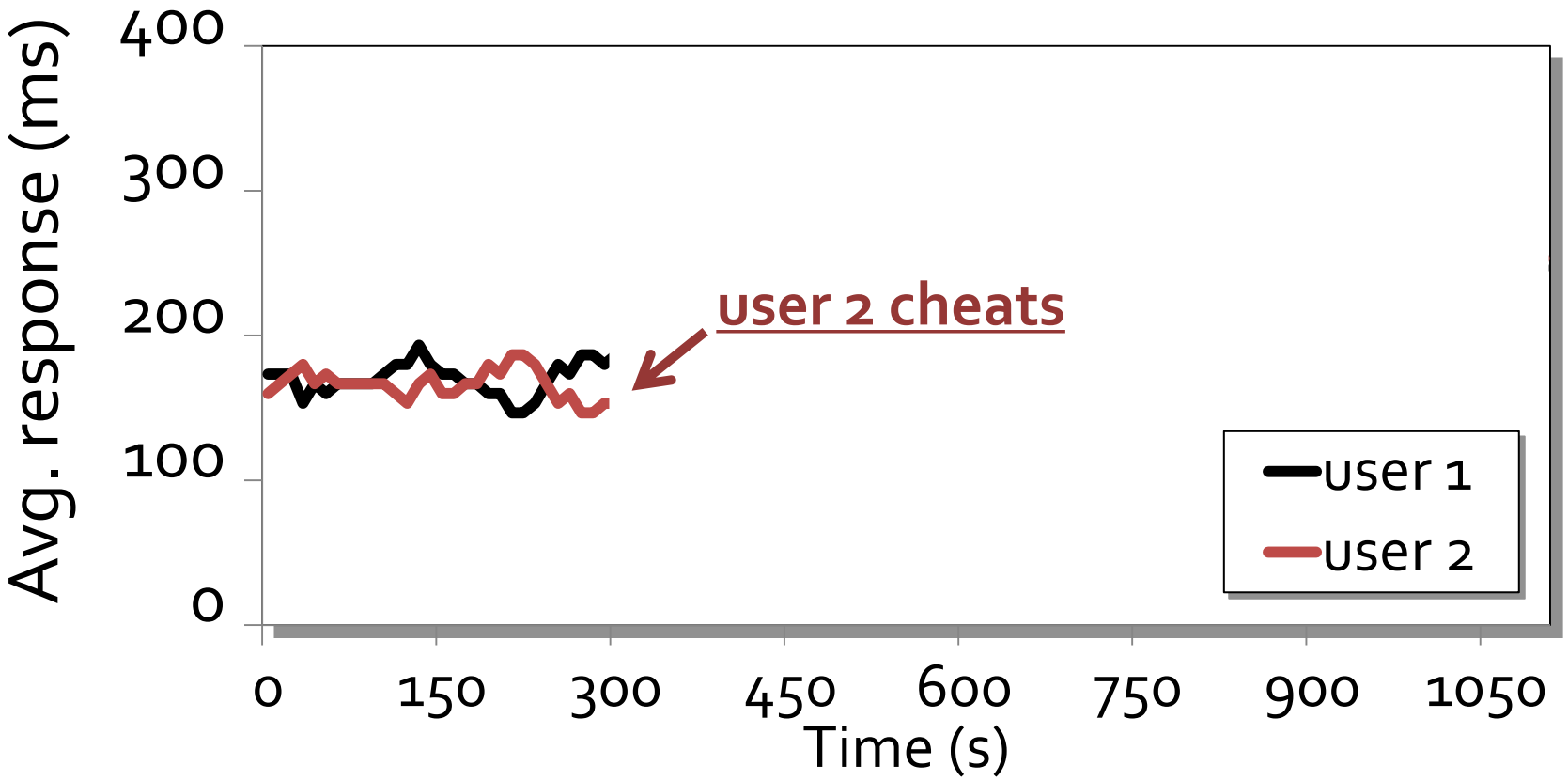
Cheating under Max-min fairness



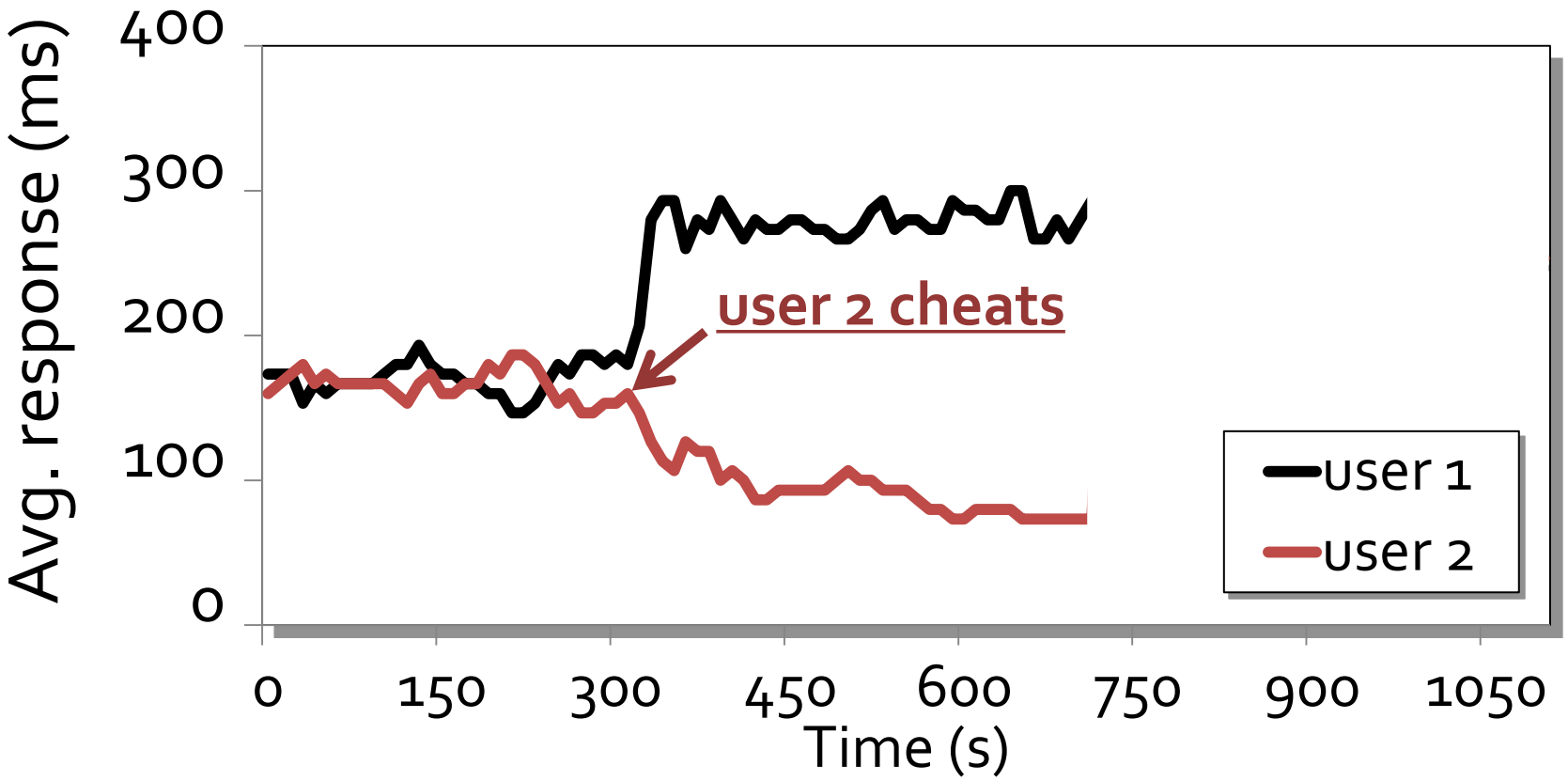
Cheating under Max-min fairness



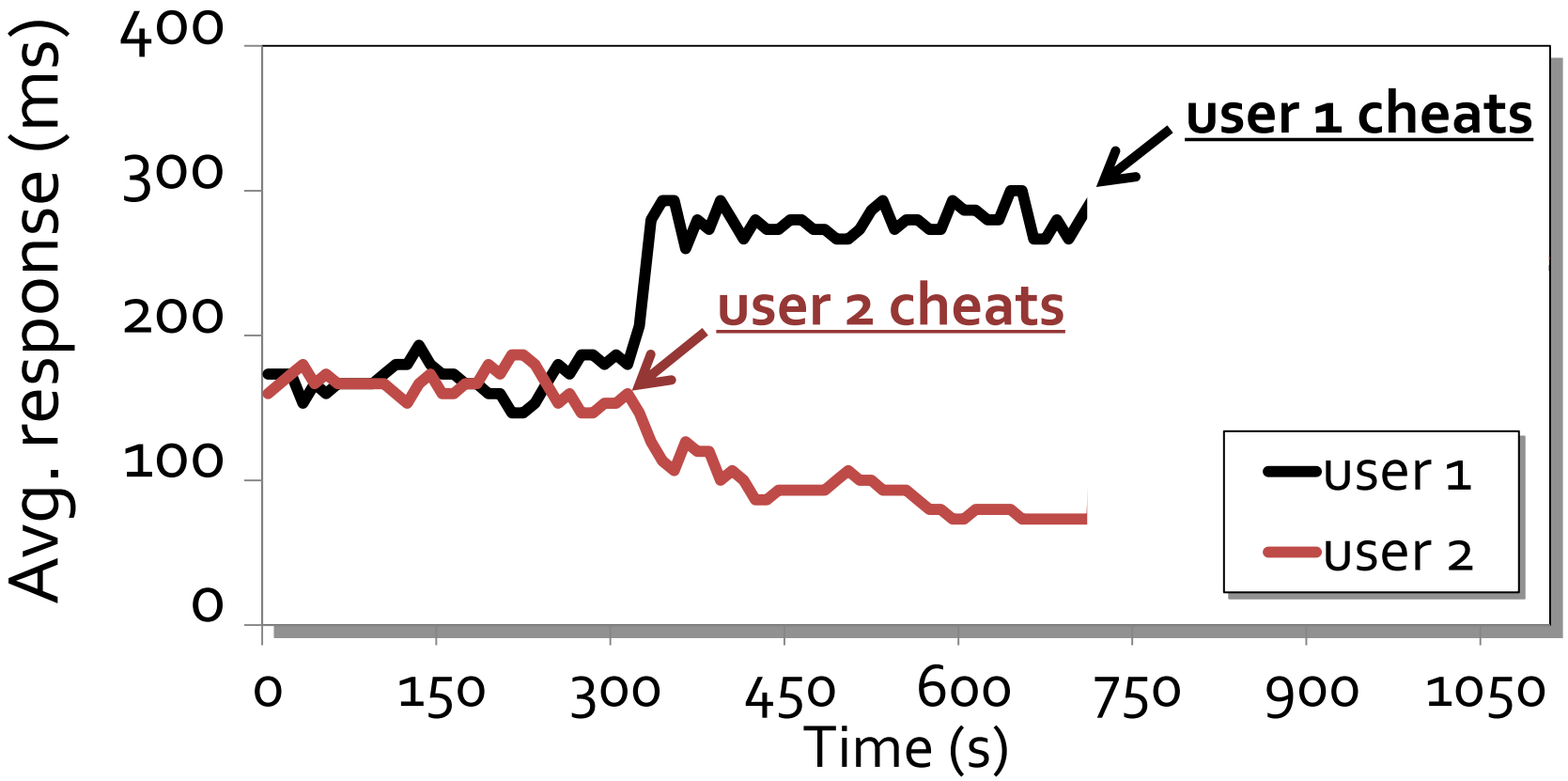
Cheating under Max-min fairness



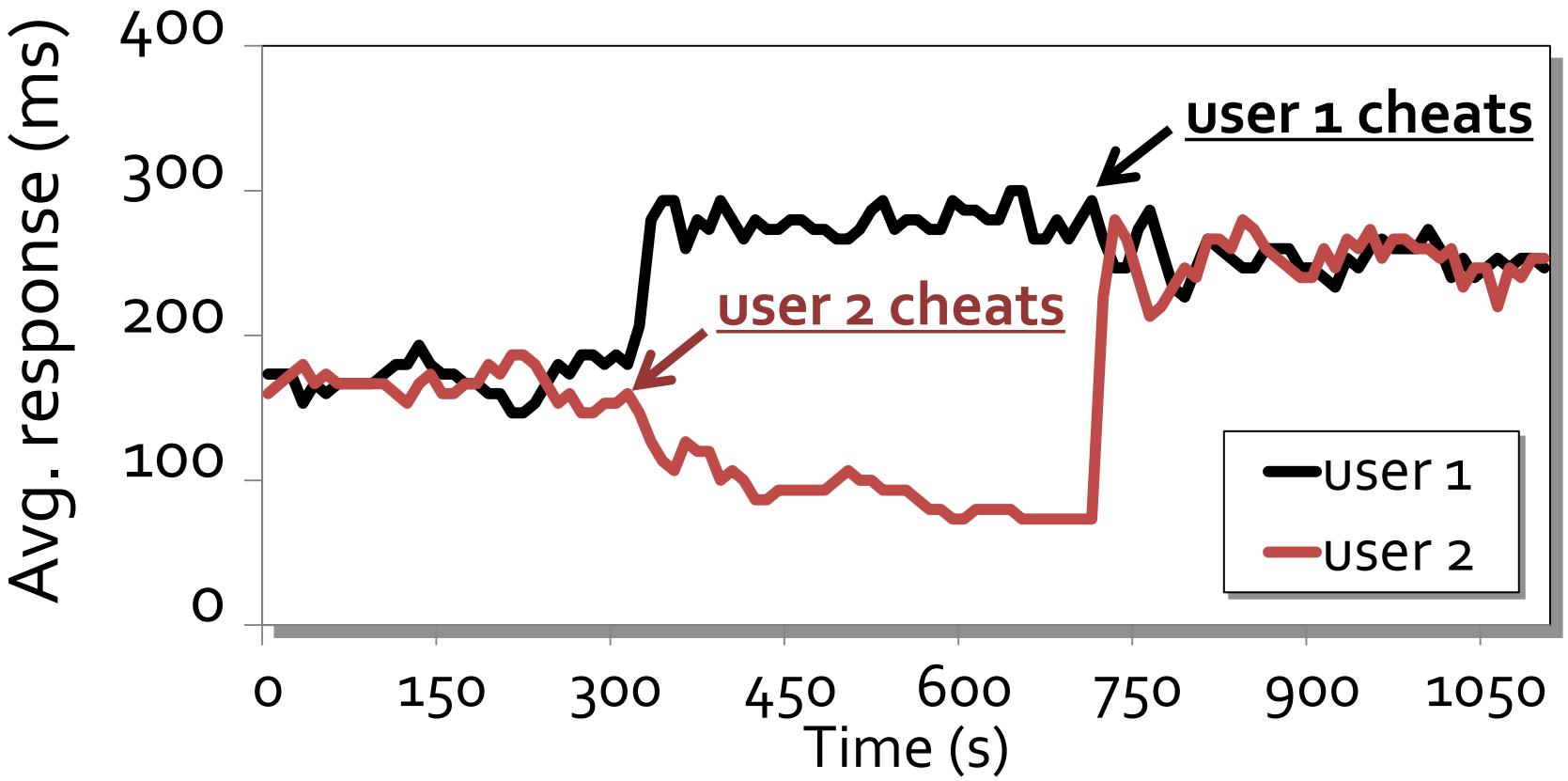
Cheating under Max-min fairness



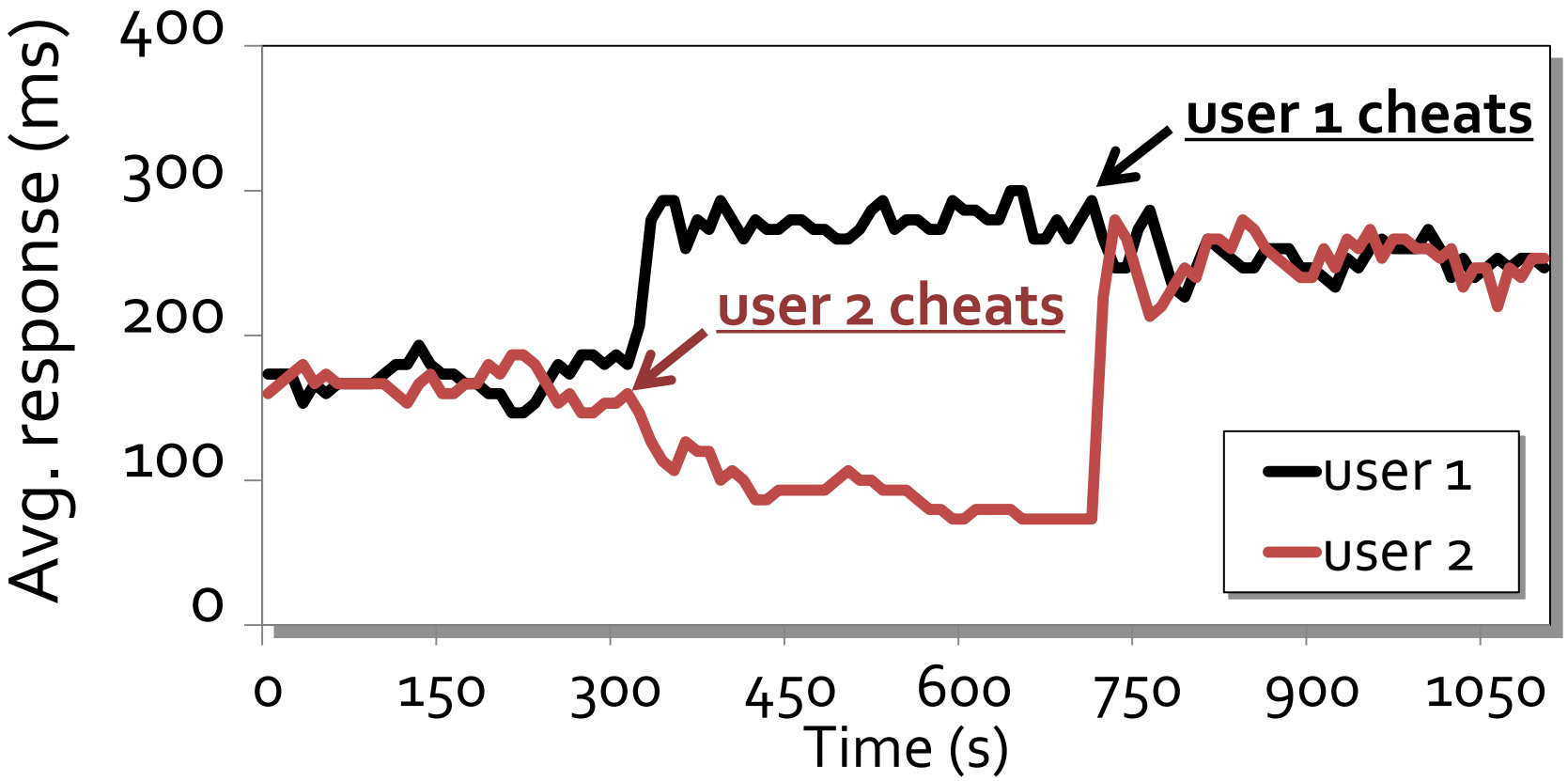
Cheating under Max-min fairness



Cheating under Max-min fairness

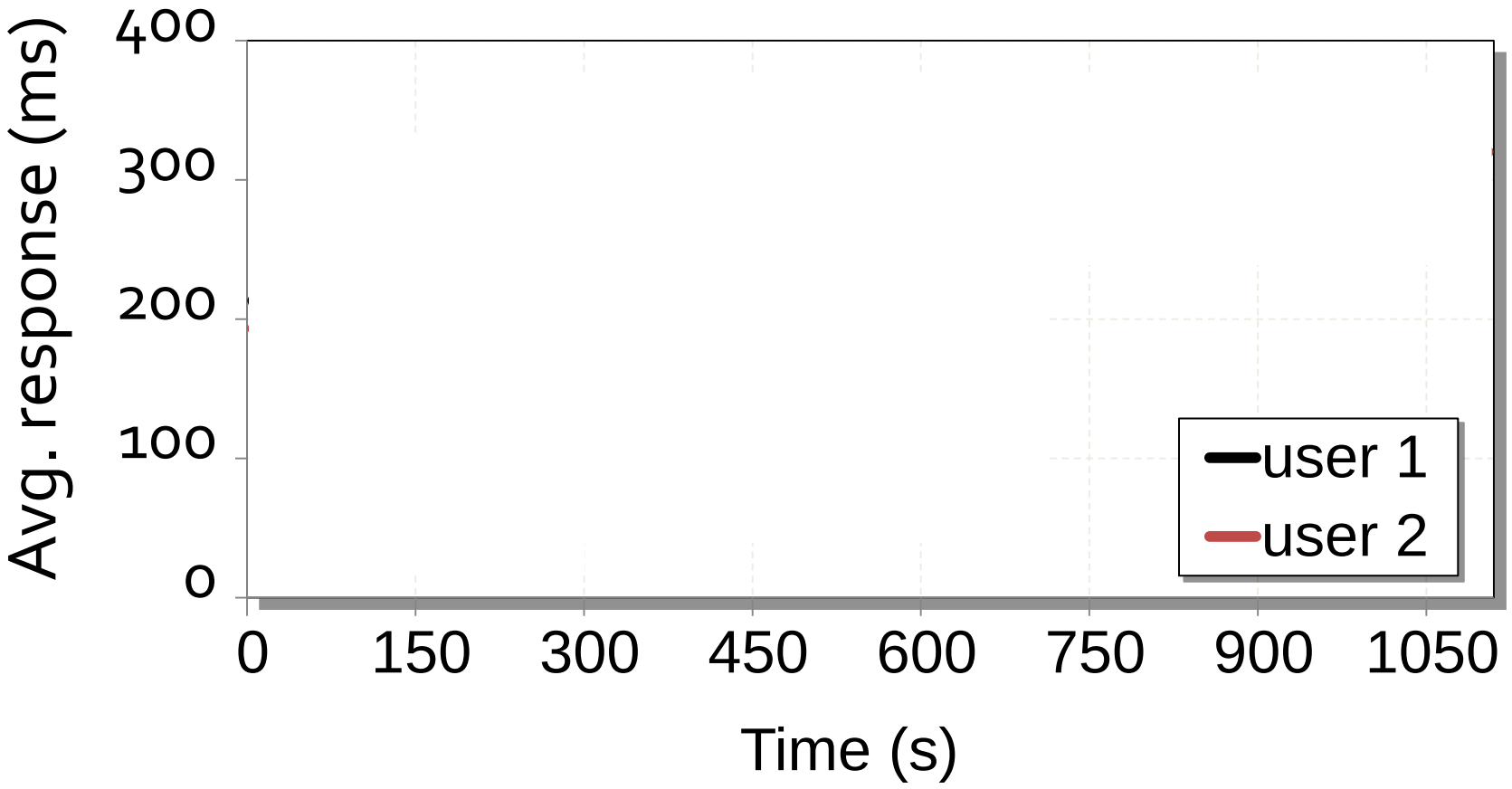


Cheating under Max-min fairness

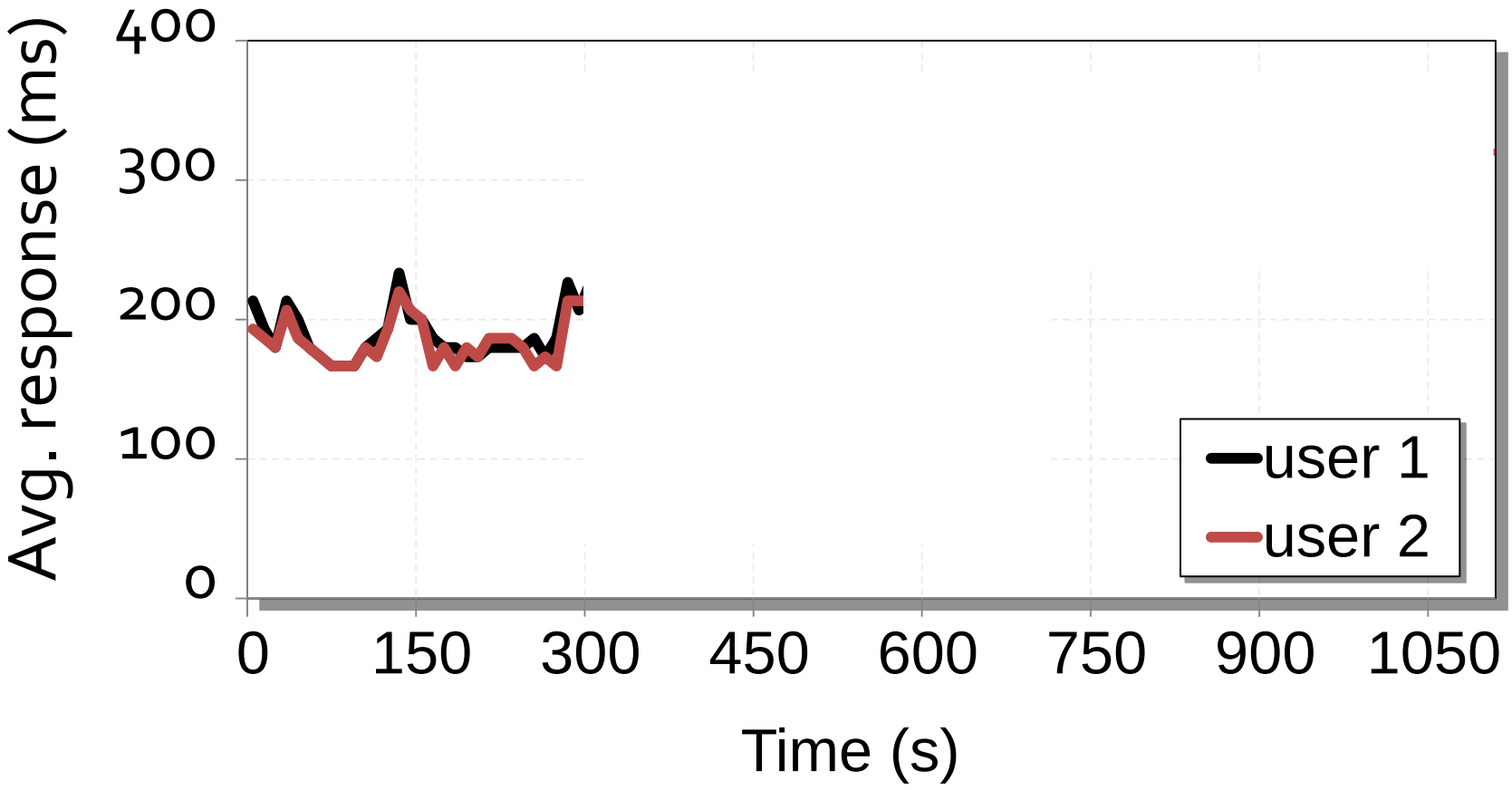


Cheating can greatly hurt user performance.

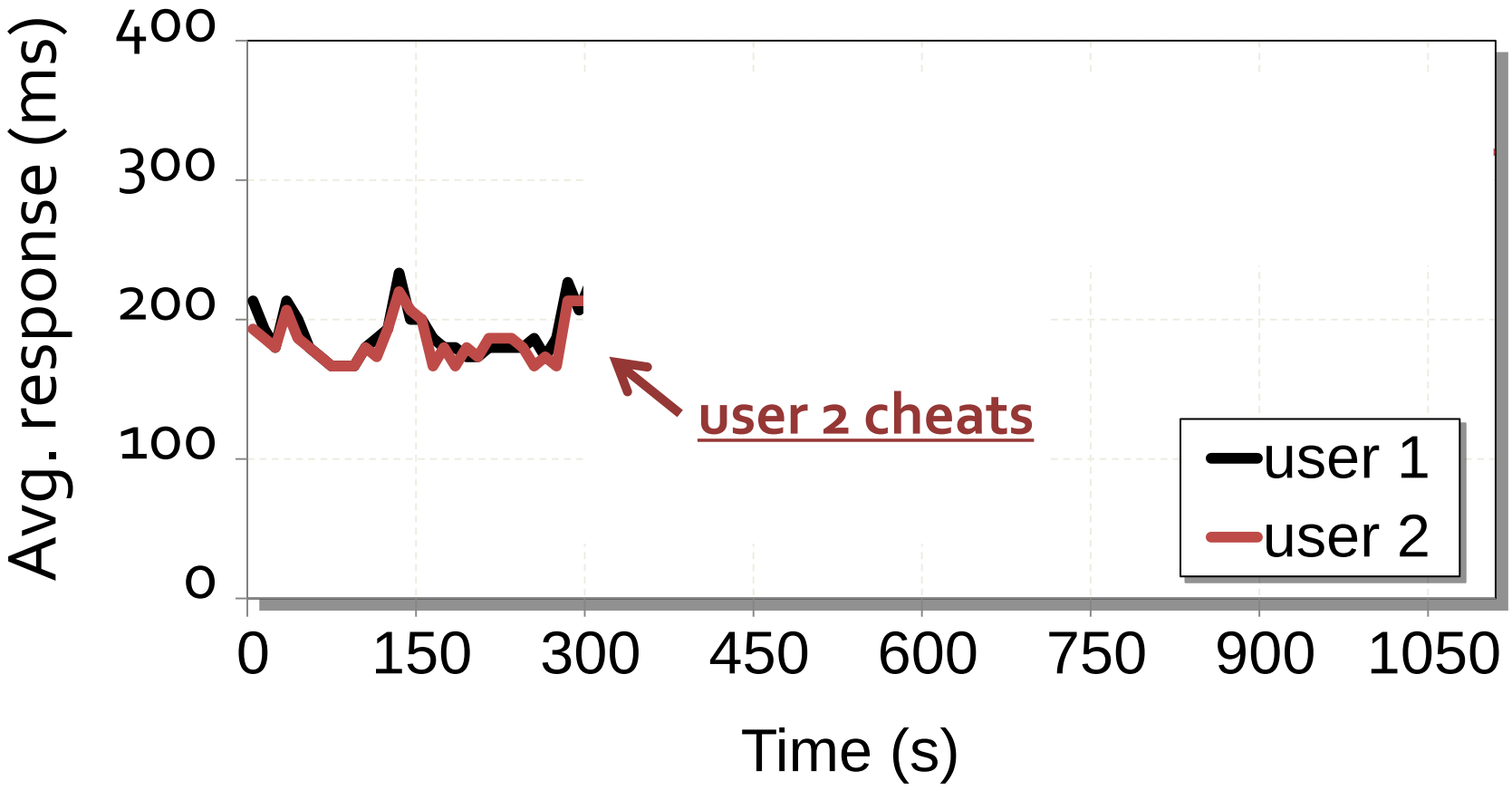
Cheating under FairRide



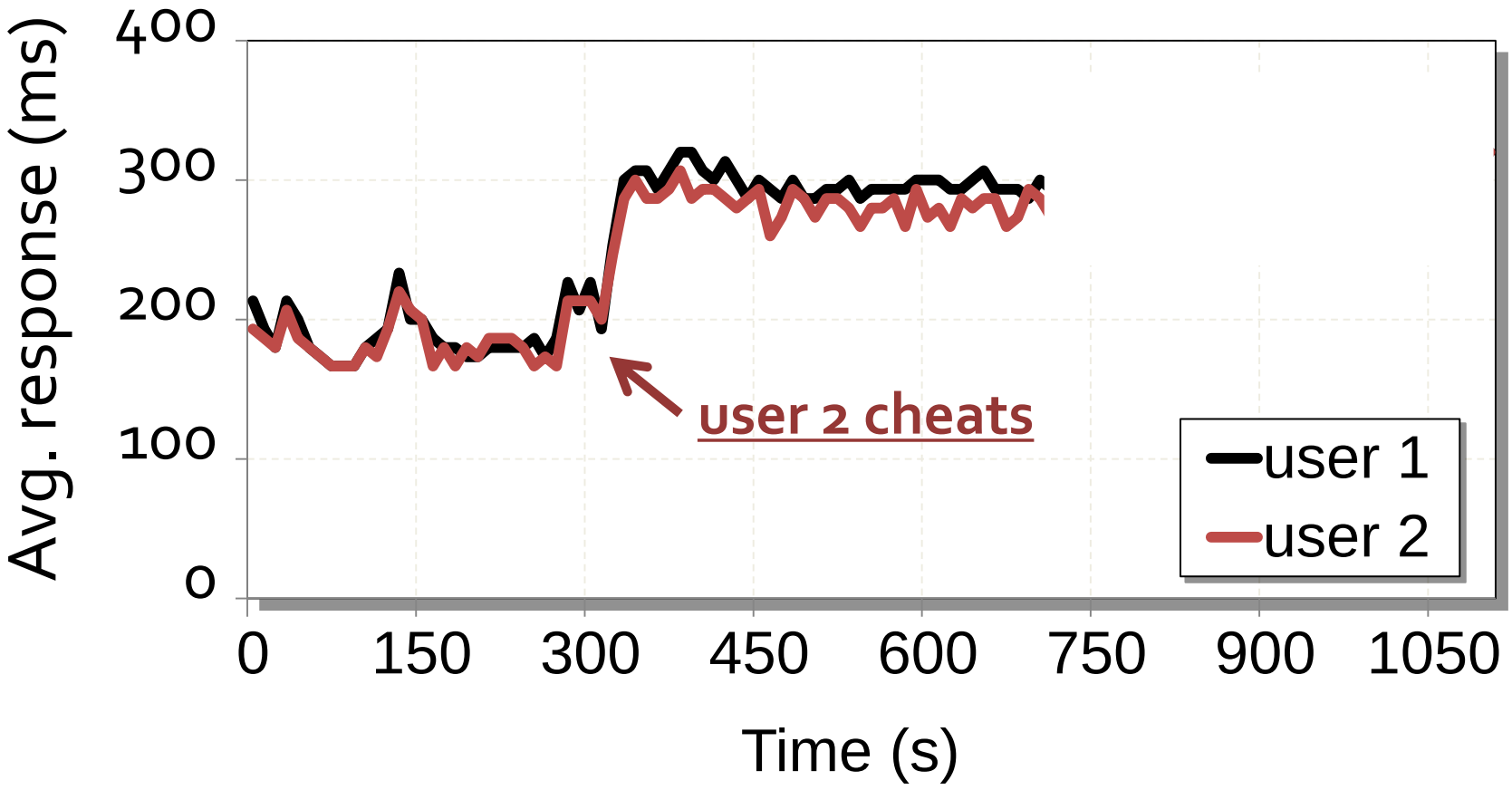
Cheating under FairRide



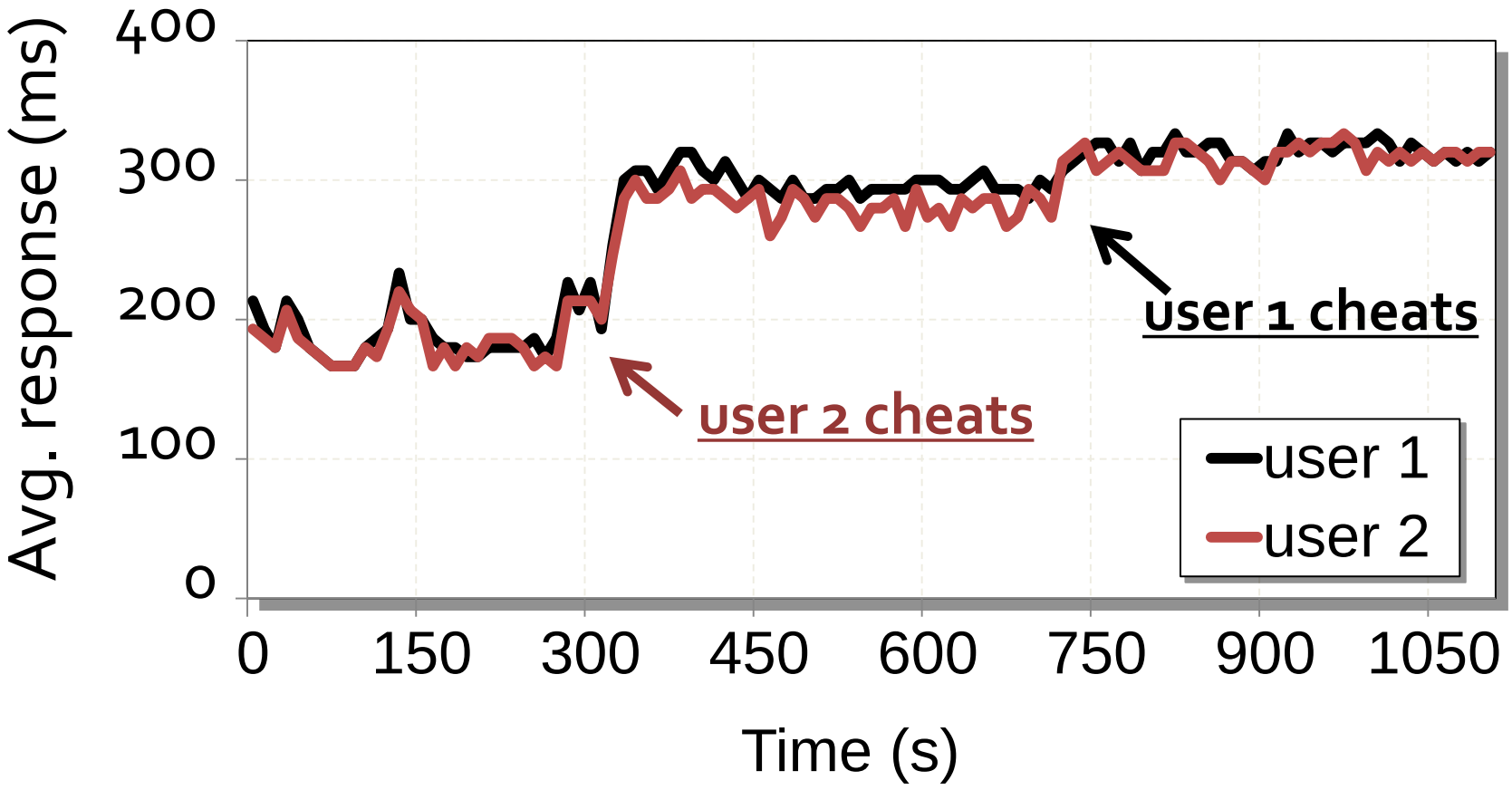
Cheating under FairRide



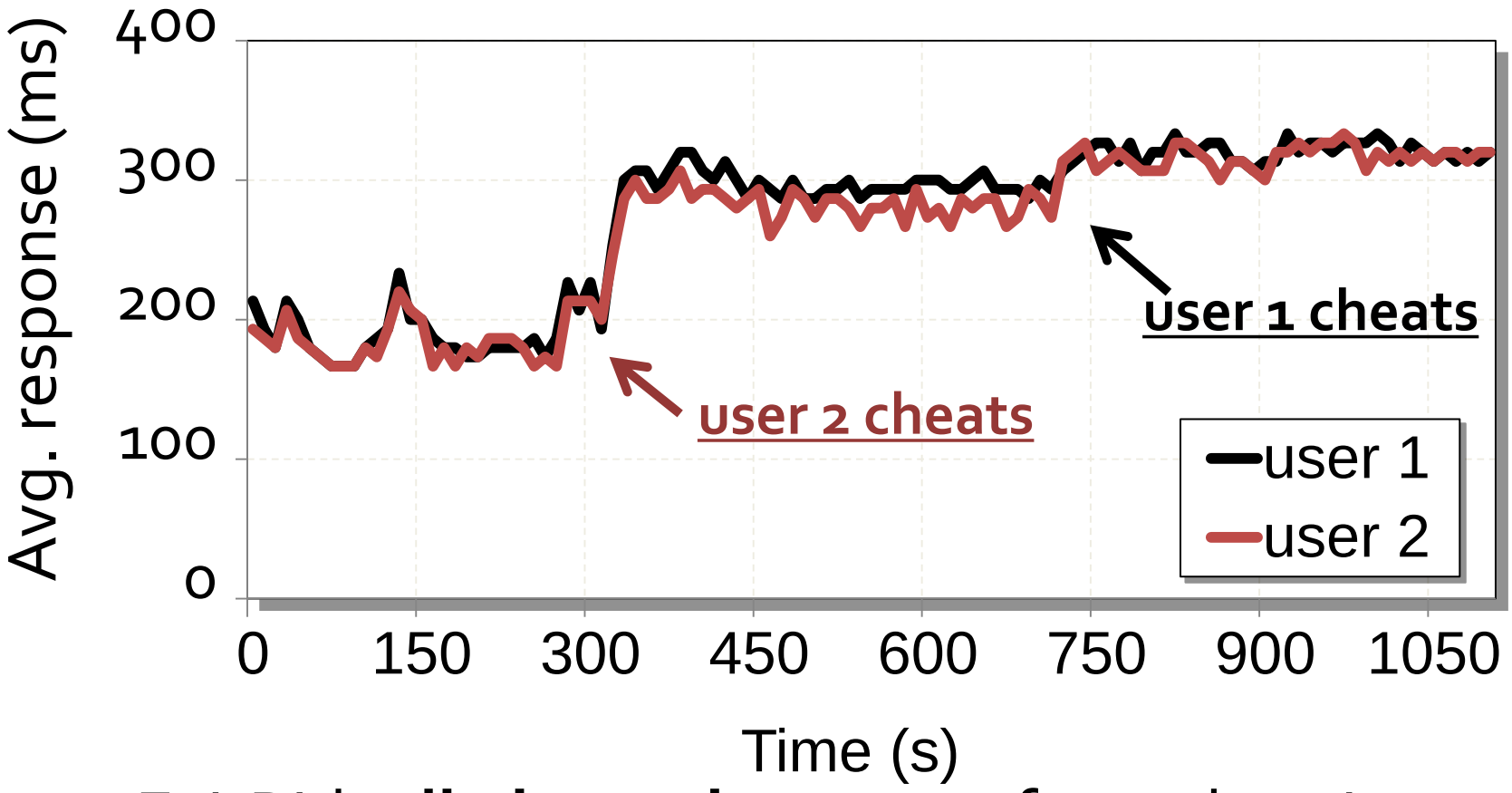
Cheating under FairRide



Cheating under FairRide

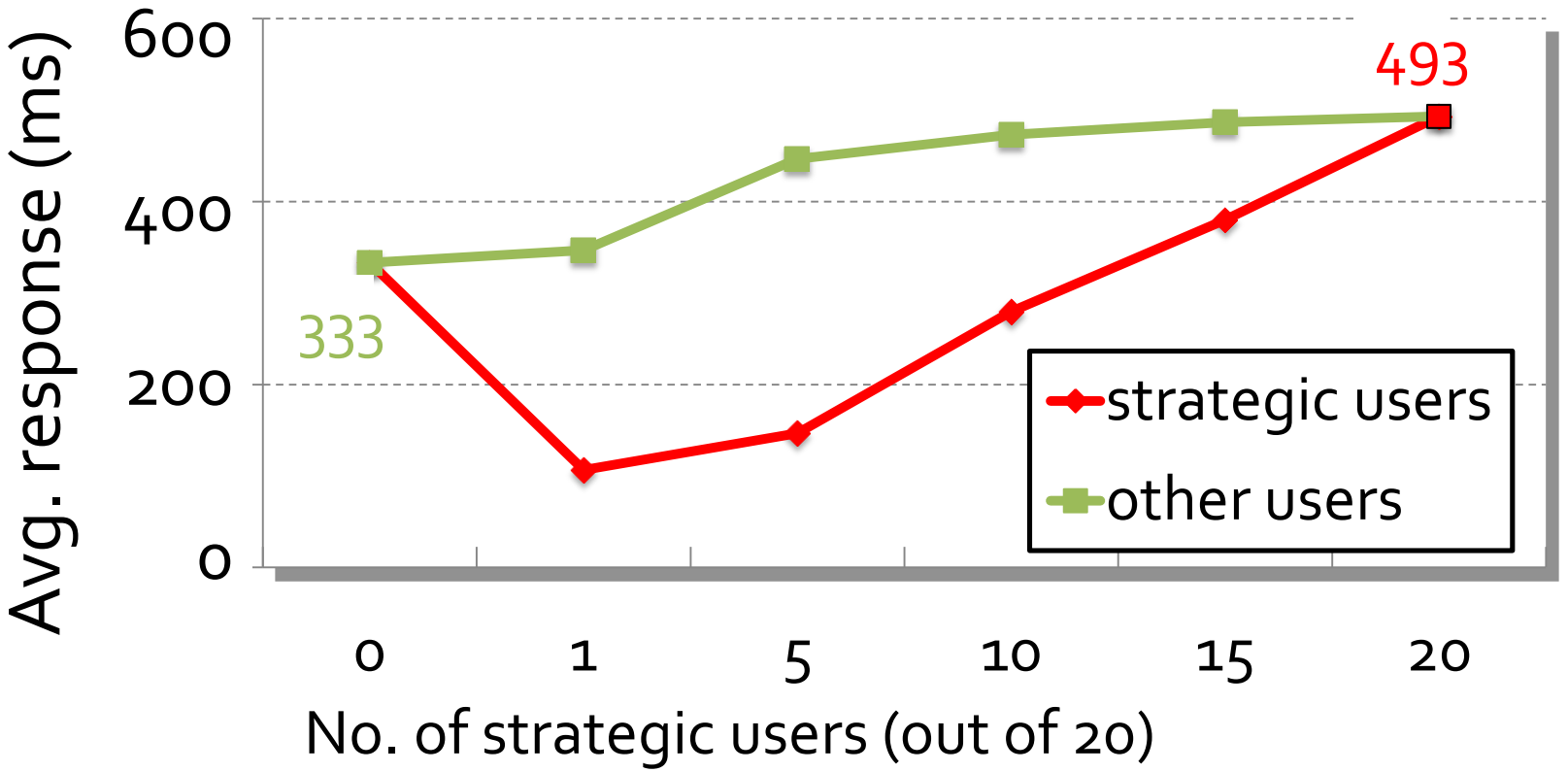


Cheating under FairRide

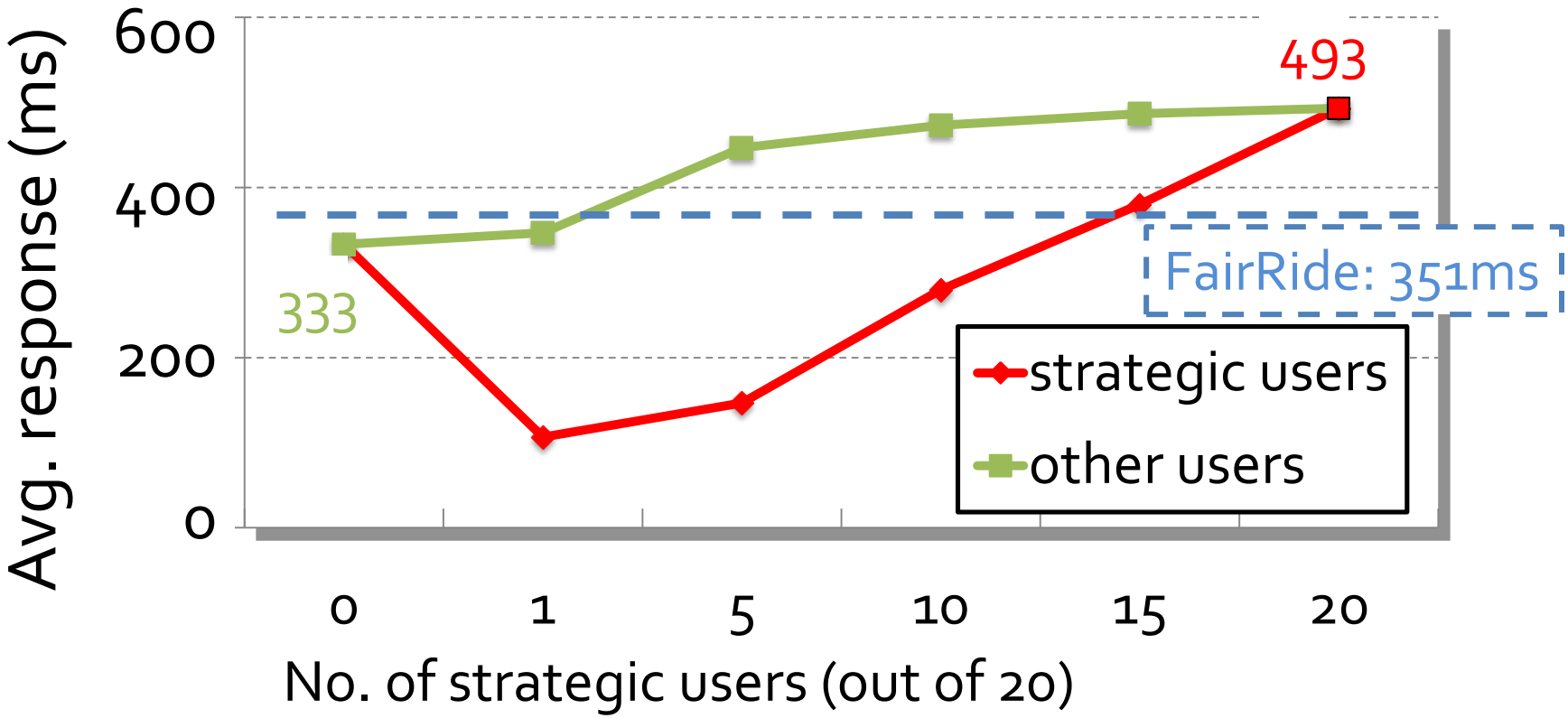


FairRide **dis-incentives** users from cheating.

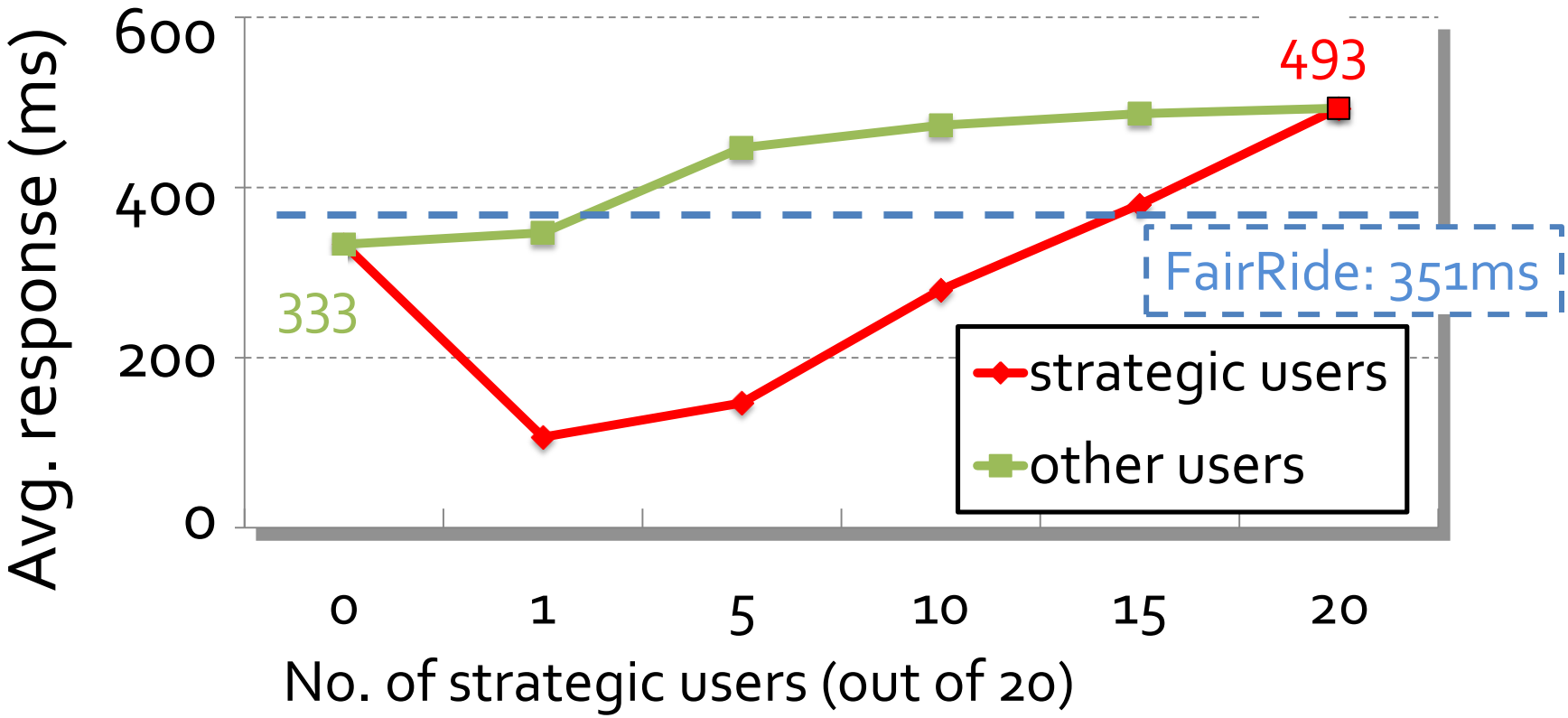
Many users


Avg. response (ms)
600
400
200
0
493
333
strategic users
other users
0
1
5
10
15
20
No. of strategic users (out of 20)

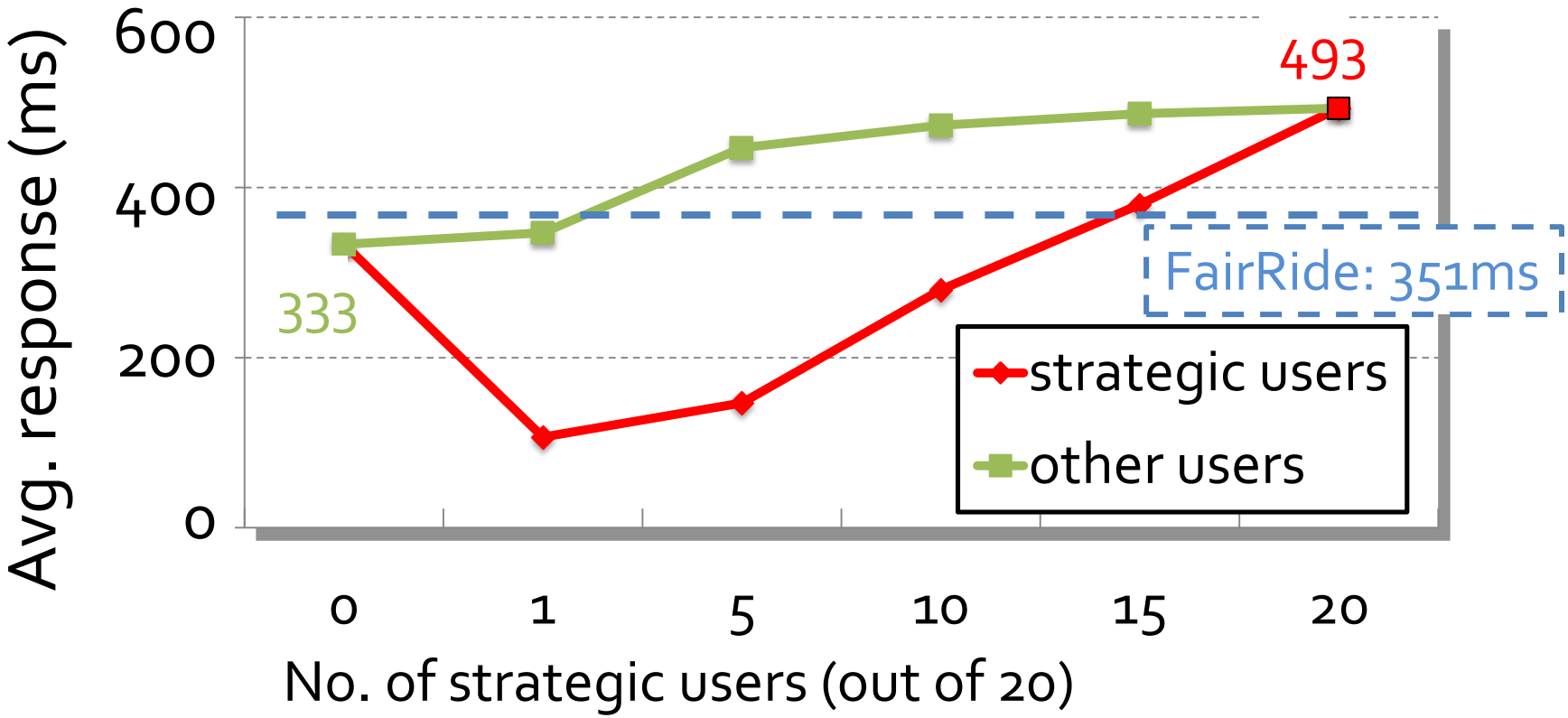
Many users


Avg. response (ms)
600
400
200
0
493
333
FairRide: 351ms
strategic users
other users
0
1
5
10
15
20
No. of strategic users (out of 20)

Many users

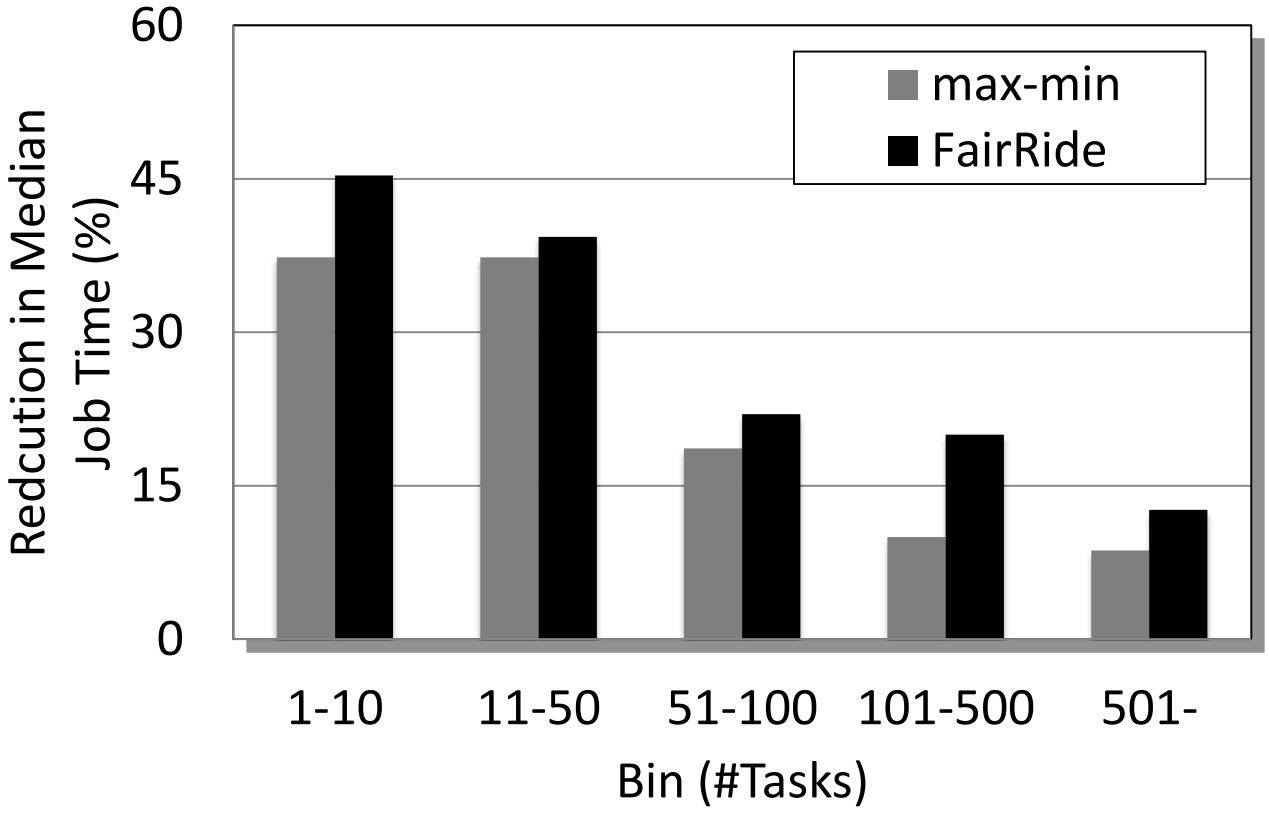

Avg. response (ms)
600
400
200
0
493
333
FairRide: 351ms
strategic users
other users
0
1
5
10
15
20
No. of strategic users (out of 20)

Many users

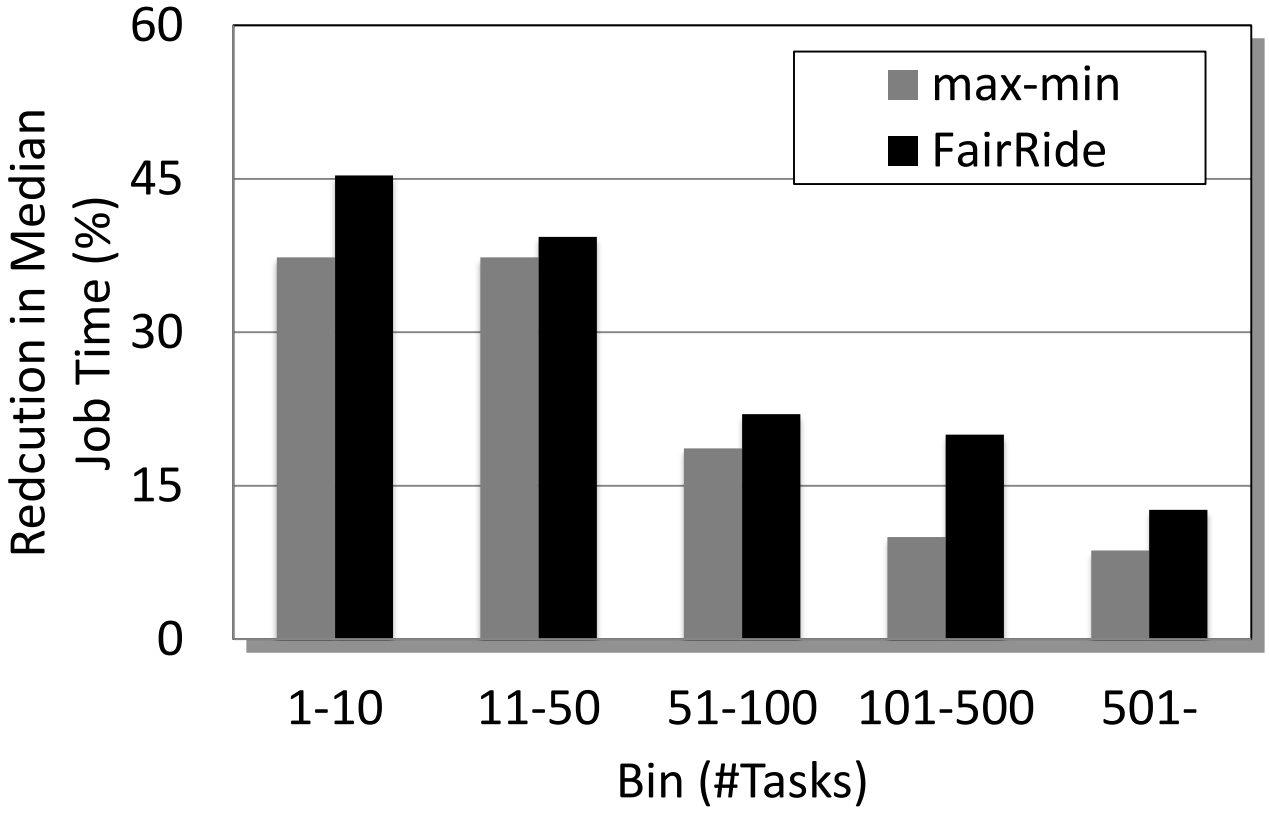


FairRide has minimal loss.

Facebook experiments



Facebook experiments



FairRide outperforms max-min fairness by 29%

Conclusion

- No policy can satisfy all desirable properties:
  - Isolation guarantee
  - Strategy proofness
  - Pareto efficiency

Conclusion

- No policy can satisfy all desirable properties:
  - Isolation guarantee
  - Strategy proofness
  - Pareto efficiency
- FairRide:isolation guarantee and strategy-proofness through ***probabilistic blocking***.
  - Outperforms static allocation and other sharing policies when users cheat.
  - Achieves this with least overhead