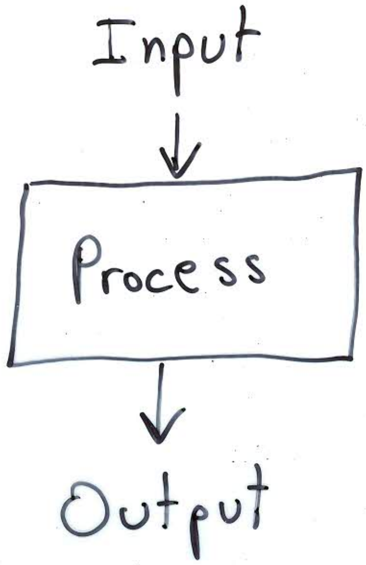
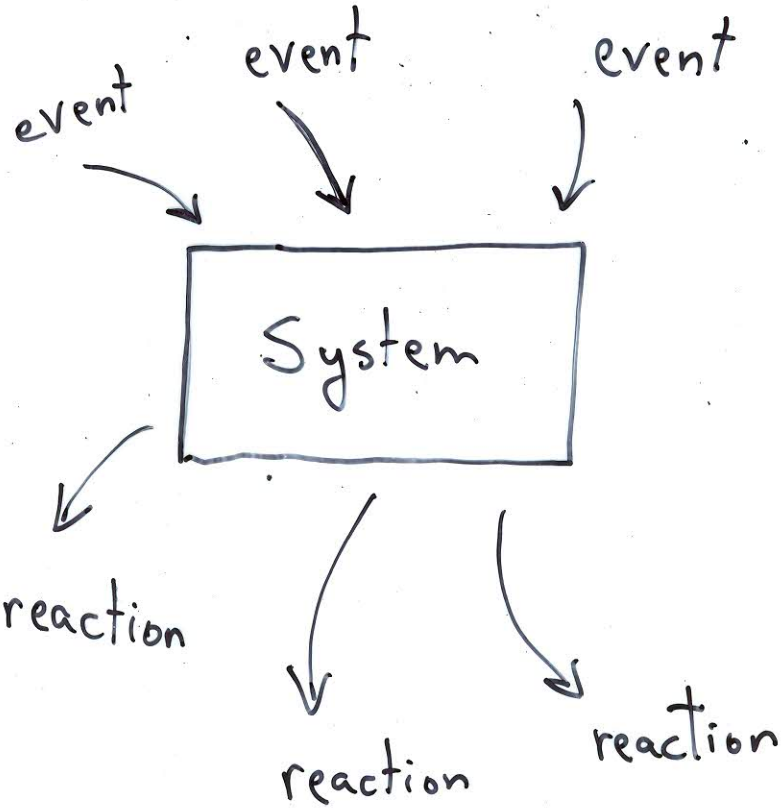
ON LINE ALGORITHMS

Traditional Algorithmic Model:



In "Real Life"



Needed: New Model.

New Measurement function.

Example: Ski Rental Problem.

A pair of skis cost \$100.

Renting a pair for one session: \$10

Should we buy or rent?

— Depends on the situation.

If we buy but never ski —

lost \$100

If we rent and ski k times

($k > 10$) —

lost $\$(k-10)10$

How can we tell if a strategy is good?

Definition: An on-line algorithm A for problem P is α-competitive if for every input sequence s,

$$C_A(s) \leq \alpha\, C_{OPT}(s)$$

where C_A is the cost of algorithm A's strategy, and C_{OPT} is the cost of the optimal strategy.

EXAMPLE: Ski Rental.

Strategy: For the first 10 times - rent the skis.

At 11th time - buy.

Claim: $\forall s \quad C_A(s) \leq 2\, C_{OPT}(s)$

Proof:

If we rent ≤ 10 times then

$$C_A(s) = C_{OPT}(s)$$

If we rent > 10 times then

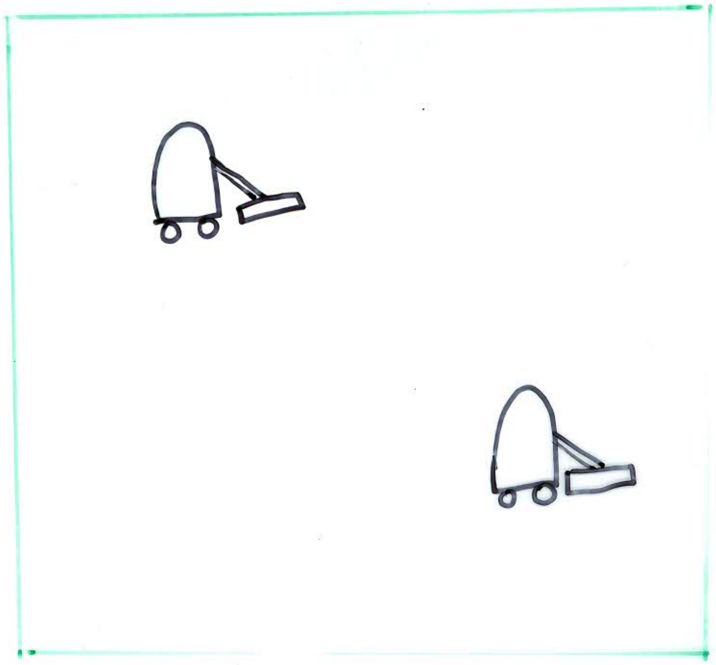
$$C_{OPT}(s) = 100$$

$$C_A(s) = 200$$

In constructing an on-line strategy we assume a malicious adversary in order to test our competitiveness.

EXAMPLE: The 2-Server Problem

Given: A grid, and two servers.

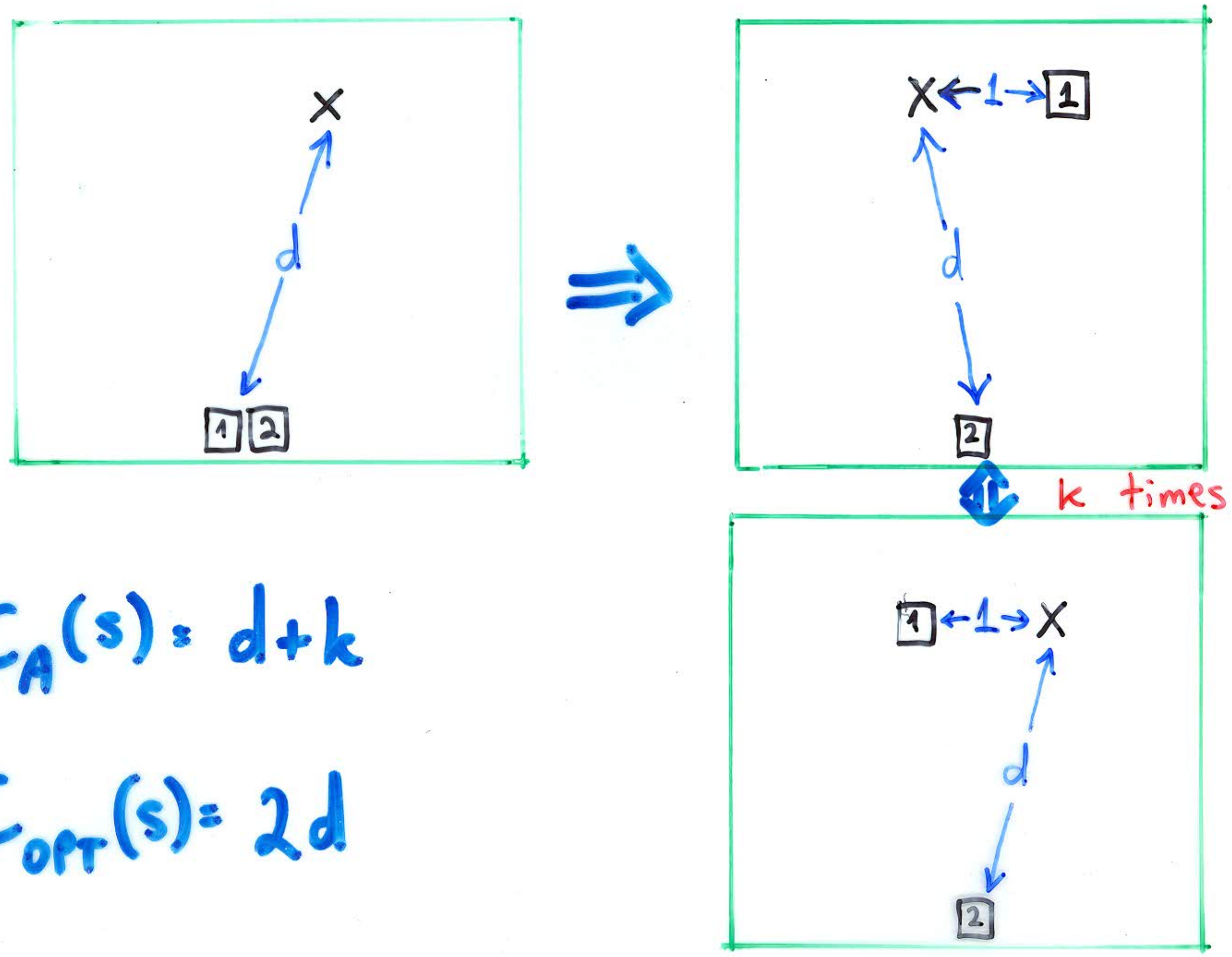


What is a competitive strategy?

Suggestion: Closest Server serves an event.

Problem: Not α-Competitive for any α

Counterexample:



$C_A(s) = d + k$

$C_{OPT}(s) = 2d$

for every fixed α,

if $k > (2\alpha - 1)d$ then

$$d + k > 2\alpha d$$

Are there competitive algorithms for the 2-server problem?

– Yes!

Simple example: HARMONIC strategy.

HARMONIC is a probabilistic strategy.

Its expected performance is competitive, i.e. $\exists \alpha$ such that

$$\sup_S \left\{ E\left[C_A(S) - \alpha C_{OPT}(S) \right] \right\} < \infty$$

In Words: The cost of algorithm A on any sequence S is at most $\alpha \times$ optimum cost + some additive constant independent of S.

HARMONIC STRATEGY

Let s_1, s_2 be the 2 servers

Given a request at location r (r's location different from those of both servers).

Move s_i to r with probability

$$\frac{\frac{1}{s_i r}}{\frac{1}{s_1 r} + \frac{1}{s_2 r}}$$

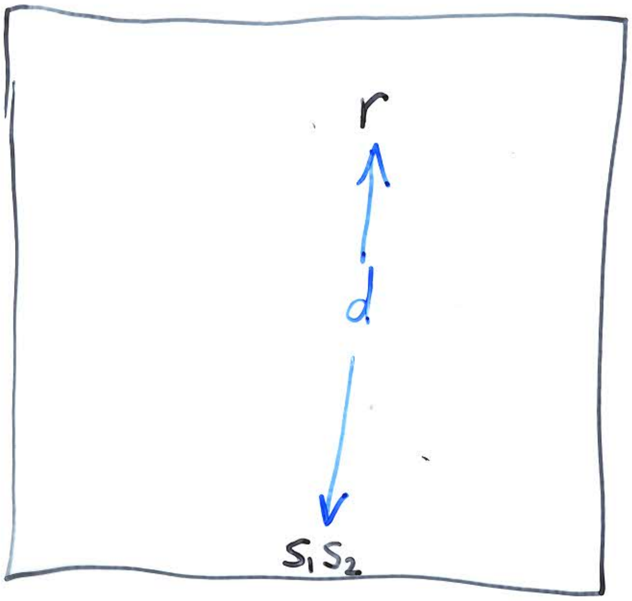
where ab is the distance from a to b.

Notes: 1.

$$\frac{\frac{1}{s_1 r}}{\frac{1}{s_1 r} + \frac{1}{s_2 r}} + \frac{\frac{1}{s_2 r}}{\frac{1}{s_1 r} + \frac{1}{s_2 r}} = 1$$

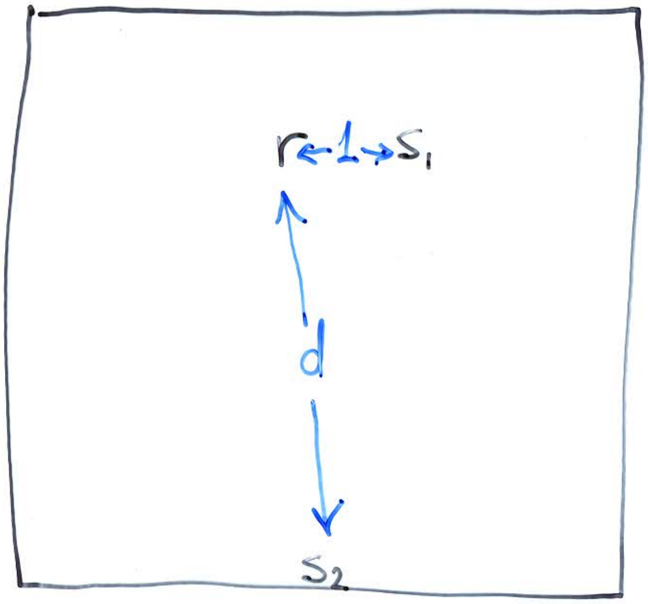
2. Strategy easily generalized to k servers.

What happens in our example?



Each server moves with probability $\frac{1}{2}$

Assume s_1 moved.



s_1 moves with probability

$$\frac{1}{\frac{1}{d}+1} = \frac{d}{d+1}$$

s_2 moves with probability

$$\frac{\frac{1}{d}}{\frac{1}{d}+1} = \frac{1}{d+1}$$

Alternate ...

This means...

After $d+1$ alternations (cost $2d$)

with high probability S_2 will

move (cost d)

and then we are at steady

state (cost 0).

CONCLUDE:

In example,

$$C_{HARMONIC} = 3C_{OPT}$$

with high probability.

Grove (1991):

HARMONIC is c_k-competitive $\forall k$ (for k-server problem)

but c_k is very large and a rapidly growing function of k.

Chrobak & Larmore (1991):

HARMONIC is 3-competitive for two servers.

What about a deterministic strategy?

Irani & Rubinfeld (1995)

A (simple) 10-competitive algorithm for the 2-server problem.
(balance based)

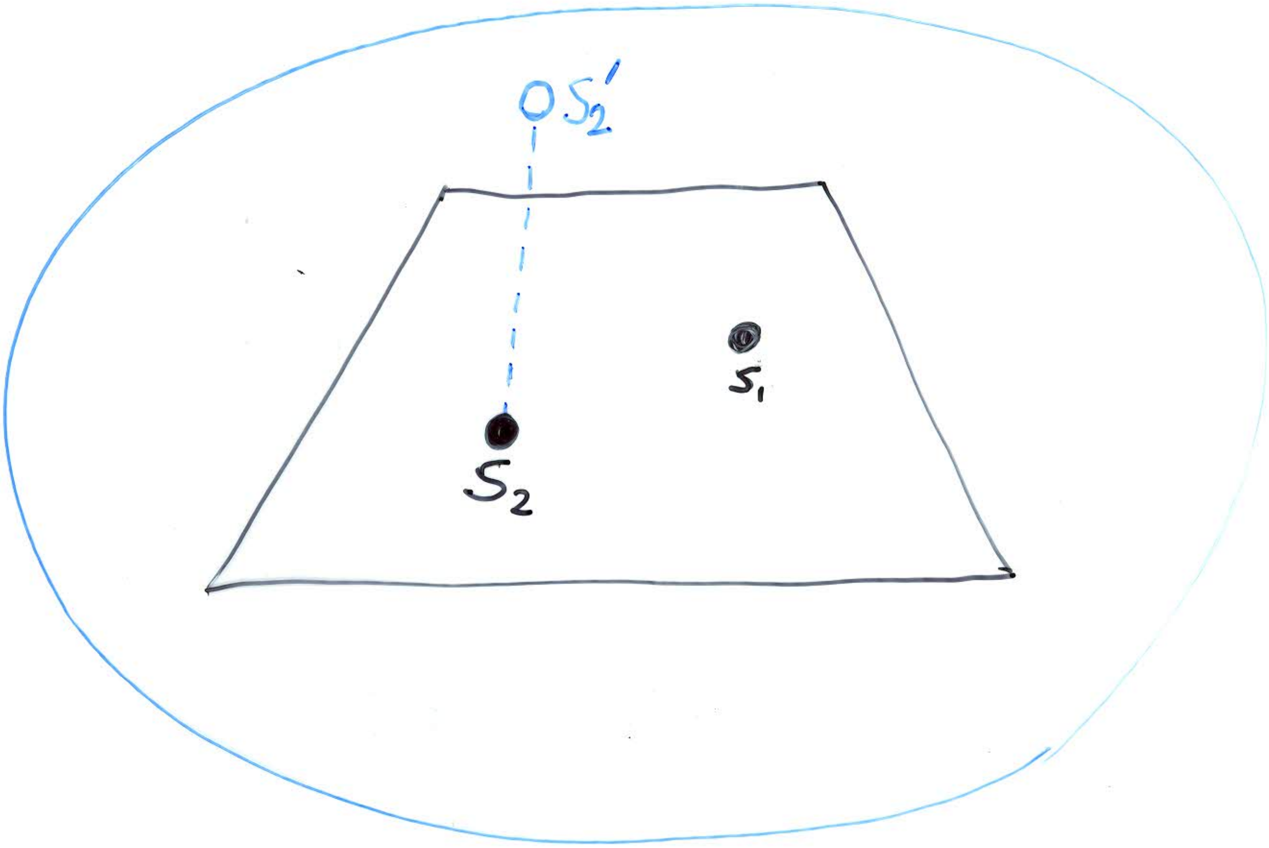
Manasse, McGeoch & Sleator (1988)

A (complicated) 2-competitive algorithm for the 2-server problem.
(residue based)

Chrobak & Larmore (1991)

A (simpler) 2-competitive algorithm for the 2-server problem.
(embedded metric space based)

IDEA: Embed grid in a larger metric space.



The requests and servers are on the grid, but every server also has a "virtual" position in a larger metric space.

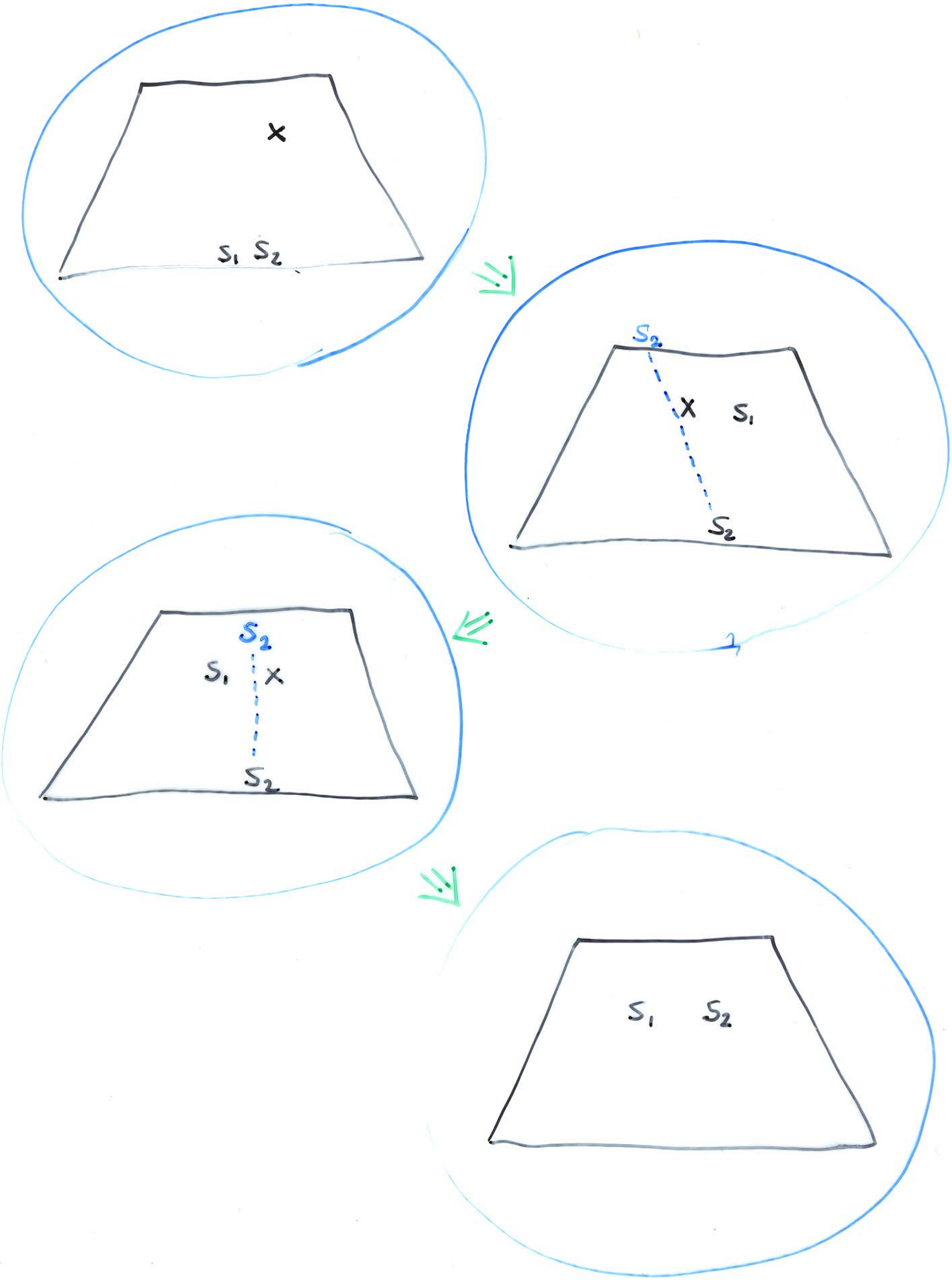
When a server is moved to serve a request, the other server's virtual image moves closer (although real image not moved and no cost incurred).

Eventually, virtual image may be close to a request - and thus serve it, even though physical server may be far.

- Server whose virtual image is closest to request - serves.
- Other server's virtual image is moved closer to request.
- The server who serves a request is physically moved and its virtual position set to coincide with physical.

-OL14-

IN EXAMPLE : (Conceptualy)



The BALANCE Strategy

Minimize the quantity:

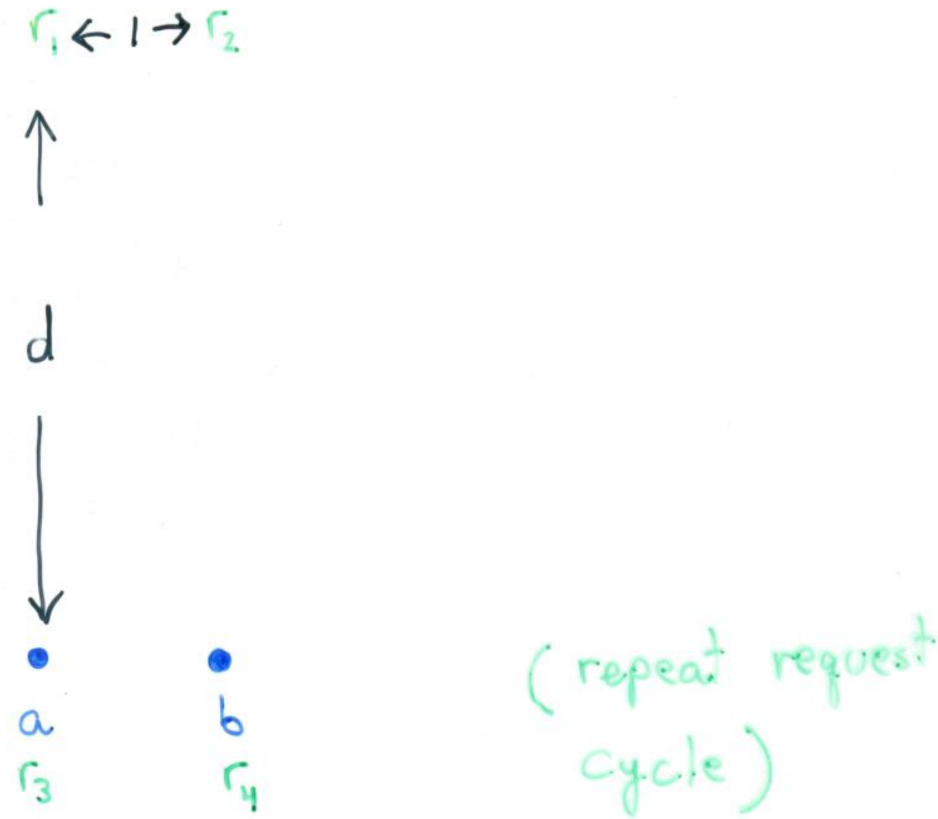
Total cost so far by server + cost of serving next request.

Notes: 1) BALANCE works on previous example.

2) BALANCE is k-competitive for k servers on a (k+1) point grid (MMS '88)

3) BALANCE has unbounded competitive ratio for 2-server problem.

EXAMPLE:



a serves $r_1, r_3, r_5, \ldots$

b serves $r_2, r_4, r_6, \ldots$

Cost for X requests: dX

Optimal cost for X requests: $X + d$.

Since d can be chosen arbitrarily large $\Rightarrow$ no competitive bound.

The Balance2 Strategy

Minimize the quantity:

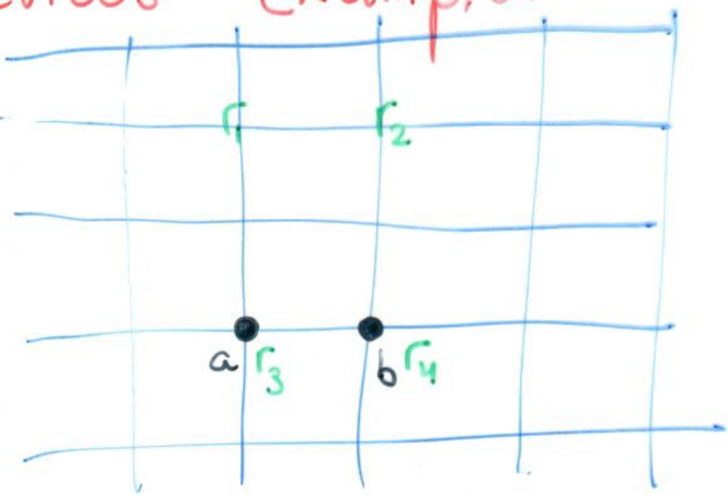
Total cost so far by server +

2× cost of serving next request.

Irani-Rubinfeld (1995):

Balance2 is 10-competitive.

On previous example:

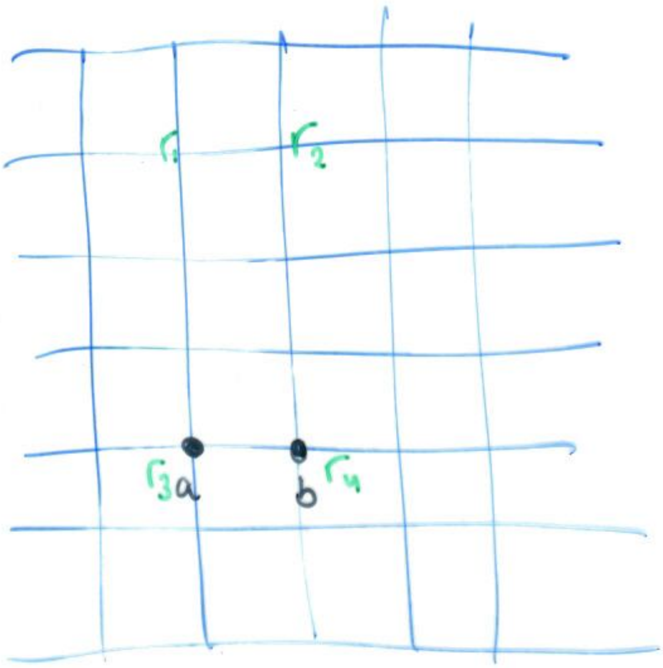


a serves $r_1, r_3, r_5, r_7, \ldots$

b serves $r_2, r_4, r_6, r_8, \ldots$

So cost is $2 \times C_{OPT}$

But once $d \geq 3$ we stabilize at optimal!



a serves r_1

$C^a = 3$

$C^b = 0$

$3+2 < 0+2\cdot 3$

so

a serves r_2

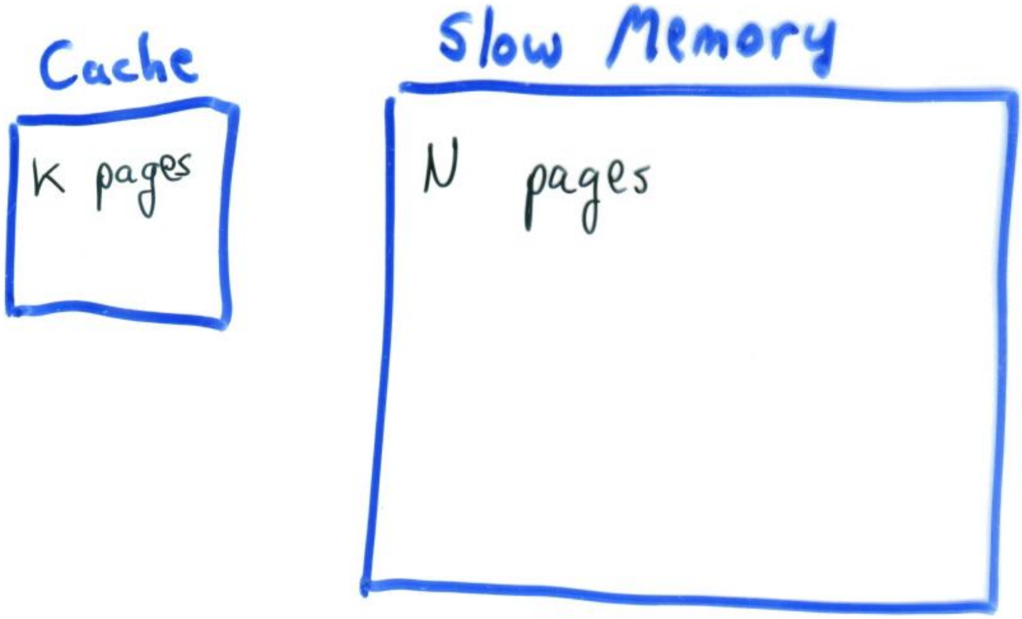
$C^a = 4$

$C^b = 0$

b serves r_3, r_4

stabilize.

THE PAGING PROBLEM



Memory Access:

Address in page P in cache. cost $= 0$.

Address in page P not in cache:

evict page q from cache - **Page fault**

read page P to cache.

cost $= 1$.

Which page q do we evict?

EXAMPLE:

$k = 4$

Page Access Sequence:

1 2 3 4 5 1 2 3 2 1 5 4 2 3 1 5 3 2 4

1, 2, 3, 4, 5

$\boxed{1 \quad 2 \quad 3 \quad 4}$ evict 4

1, 2, 3, 2, 1, 5, 4

$\boxed{1 \quad 2 \quad 3 \quad 5}$ evict 5

2, 3, 1, 5

$\boxed{1 \quad 2 \quad 3 \quad 4}$ evict 1

3, 2, 4

$\boxed{5 \quad 2 \quad 3 \quad 4}$

OFFLINE ALGORITHM:

Evict page whose next access is farthest.

This is optimum.

ONLINE STRATEGIES:

LIFO (Last In First Out)

Evict most recent arrival.

In previous example: LIFO = OPT.

But... $1\ 2\ 3\ 4\ \underbrace{5\ 4\ 5\ 4\ 5\ 4\ \cdots\ 5\ 4}_{n \text{ times}}$

2n faults!

OPT = 1 fault!

COMPETITIVE RATIO: ∞

LEAST RECENTLY USED (LRU):

Evict page whose last access was farthest (in past).

Return to Example:

1 2 3 4 5 1 2 3 2 1 5 4 2 3 1 5 3 2 4

1, 2, 3, 4, 5

$\boxed{1\ 2\ 3\ 4}$ evict 1

1

$\boxed{2\ 3\ 4\ 5}$ evict 2

2

$\boxed{3\ 4\ 5\ 1}$ evict 3

3

$\boxed{4\ 5\ 1\ 2}$ evict 4

COST : 6

2, 1, 5, 4

$\boxed{5\ 1\ 2\ 3}$ evict 5

2, 3, 1, 5

$\boxed{1\ 2\ 3\ 4}$ evict 1

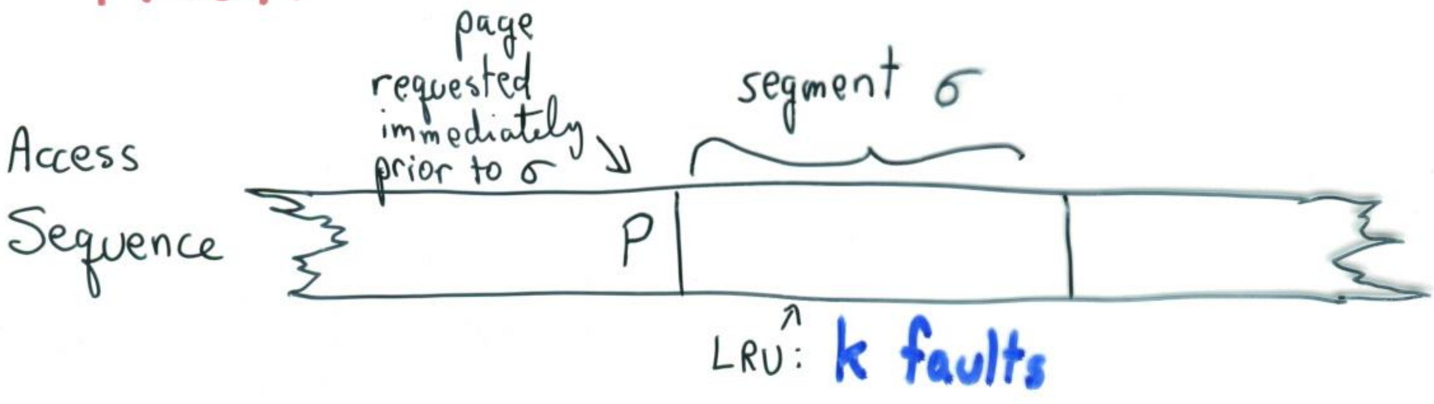
COST OF OPT : 3

3, 2, 4

$\boxed{1\ 2\ 3\ 4}$

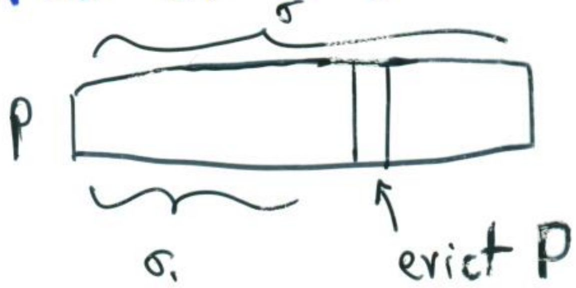
Theorem: LRU is k-competitive.

Proof:



Cases:

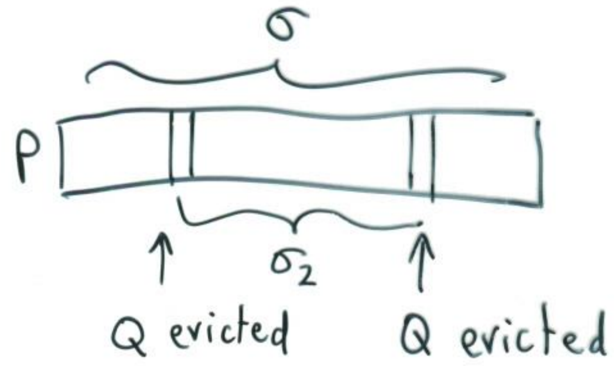
1. p evicted in σ.



least recently used page evicted ⇒
in σ_1, k-1 different pages were requested.
p was a different request (k).
Now new request is k+1 st.

Even OPT needs to fault at least once in k+1 different page requests!

2.



As in case 1, for Q to be evicted twice $\Rightarrow$

$k+1$ different pages accessed in σ_2 $\Rightarrow$

OPT faults at least once.

3.

k faults in σ - none P, none twice $\Rightarrow$

k **different** pages were faulted, none P!

With P, $k+1$ different pages were accessed $\Rightarrow$

OPT faults at least once.

Conclude: For every k faults of LRU, at least one fault of OPT.

$$LRU \leq k \cdot OPT$$

UPPER BOUND:

1 2 3 . . . k k+1 $\underbrace{\text{k+1 1 2 3 . . . k}}_{n}$ k k+1 1 . . .

LRU faults: 1 2 3 4 . . . k-1 k k+1 . . .

OPT faults: k k-1

For sequence of length n:

LRU faults $n-k$ times.

OPT faults $\frac{n-k}{k}$ times.

$$LRU = k \; OPT$$

CAN A BETTER STRATEGY BE FOUND?

Theorem: No deterministic algorithm can have a better than k competitive ratio in worst case.

Proof:

Consider case were request sequence has only k+1 different pages.

Online Algorithm:

Malicious adversary always requests page that was evicted last time.

fault every request.

Offline Algorithm:

Page furthest in future is evicted ⇒ **fault after at least k-requests.**