

# **Algorithmen II**

**Peter Sanders, Florian Kurpicz**

**Übungen:**

**Hans-Peter Lehmann, Daniel Seemaier**

Institut für Theoretische Informatik

Web:

[http://algo2.iti.kit.edu/AlgorithmenII\\_WS21.php](http://algo2.iti.kit.edu/AlgorithmenII_WS21.php)

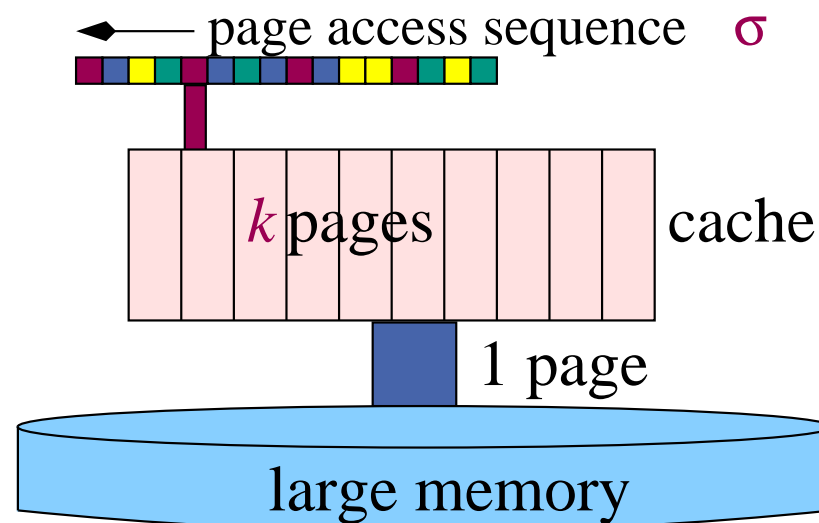
# 13 Onlinealgorithmen [z.T. von Rob van Stee]

- ☐ Information is revealed to the algorithm **in parts**
- ☐ Algorithm needs to process each part **before** receiving the next
- ☐ There is **no information** about the future  
(in particular, no probabilistic assumptions!)
- ☐ How well can an algorithm do  
compared to an algorithm that **knows everything**?
- ☐ Lack of **knowledge** vs. lack of **processing power**



# Examples

- ☐ Renting Skis etc.
- ☐ **Paging** in a virtual memory system
- ☐ **Routing** in communication networks
- ☐ **Scheduling** machines in a factory, where orders arrive over time
- ☐ Google **placing** advertisements



# Competitive analysis

☐ Idea: compare online algorithm ALG to offline algorithm OPT

☐ Worst-case performance measure

☐ Definition:

$$C_{ALG} = \sup_{\sigma} \frac{ALG(\sigma)}{OPT(\sigma)}$$

(we look for the input that results in worst **relative** performance)

☐ Goal:

find ALG with **minimal**  $C_{ALG}$

## A typical online problem: ski rental

- ☐ Renting skis costs 50 euros, buying them costs 300 euros
- ☐ You do not know in advance how often you will go skiing
- ☐ Should you rent skis or buy them?



## A typical online problem: ski rental

- ☐ Renting skis costs 50 euros, buying them costs 300 euros
- ☐ You do not know in advance how often you will go skiing
- ☐ Should you rent skis or buy them?
- ☐ Suggested algorithm: buy skis on the sixth trip
- ☐ Two questions:
  - How good is this algorithm?
  - Can you do better?



## Upper bound for ski rental

- ☐ You plan to buy skis on the sixth trip
- ☐ If you make five trips or less, you pay **optimal** cost (50 euros per trip)
- ☐ If you make at least six trips, you pay 550 euros
- ☐ In this case OPT pays at least 300 euros
- ☐ Conclusion: algorithm is  $\frac{11}{6}$ -competitive:  
it never pays more than  $\frac{11}{6}$  times the optimal cost

## Lower bound for ski rental

- Suppose you buy skis **earlier**, say on trip  $x < 6$ .

You pay  $300 + 50(x - 1)$ , OPT pays only  $50x$

$$\frac{250 + 50x}{50x} = \frac{5}{x} + 1 \geq 2.$$

- Suppose you buy skis **later**, on trip  $y > 6$ .

You pay  $300 + 50(y - 1)$ , OPT pays only 300

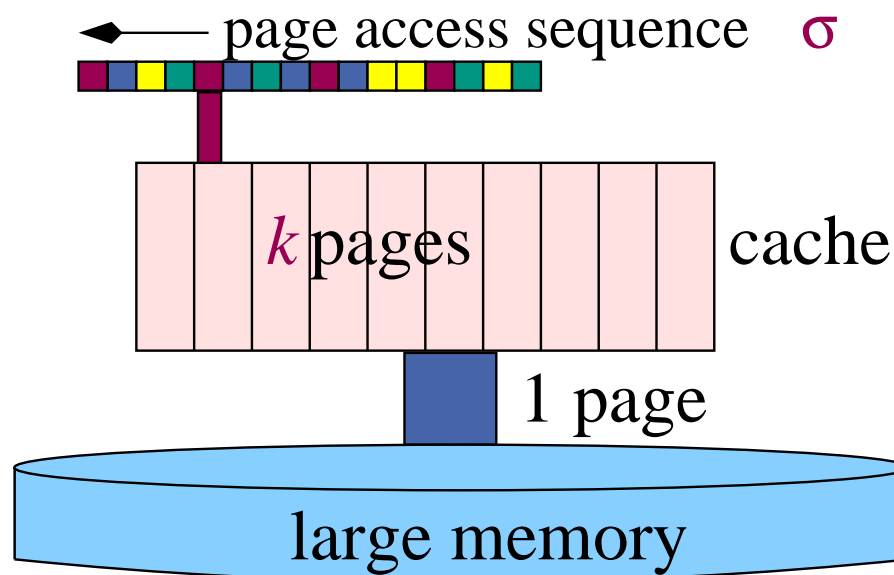
$$\frac{250 + 50y}{300} = \frac{5 + y}{6} \geq 2.$$

- Idea: do not pay the large cost (buy skis) until you would have paid **the same amount** in small costs (rent)



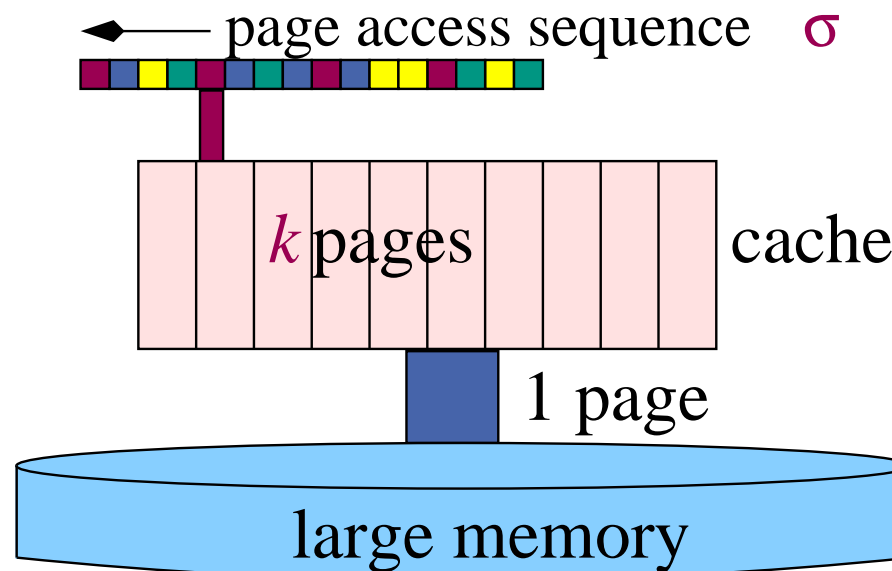
# Paging

- ☐ Computers usually have a small amount of fast memory (cache)
- ☐ This can be used to store data (pages) that are often used
- ☐ Problem when the cache is full and a new page is requested
- ☐ Which page should be thrown out (evicted)?



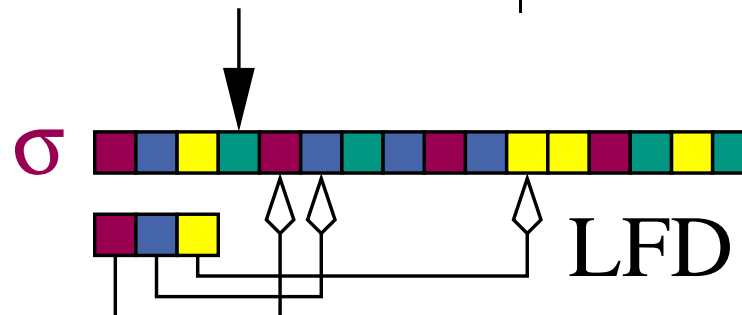
# Definitions

- ☐  $k$  = size of cache (number of pages)
- ☐ We assume that access to the cache is **free**, since accessing main memory costs much more
- ☐ Thus, a cache hit costs 0 and a miss (fault) costs 1
- ☐ The goal is to **minimize the number of page faults**



# Paging algorithms

algorithm		which page to <b>evict</b>
LIFO	Last In First Out	<b>newest</b>
FIFO	First In First Out	<b>oldest</b>
LFU	Least Frequently used	requested <b>least often</b>
LRU	Least Recently Used	requested <b>least recently</b>
FWF	Flush When Full	<b>all</b>
LFD	Longest Forward Distance	(re)requested <b>latest</b> in the <b>future</b>

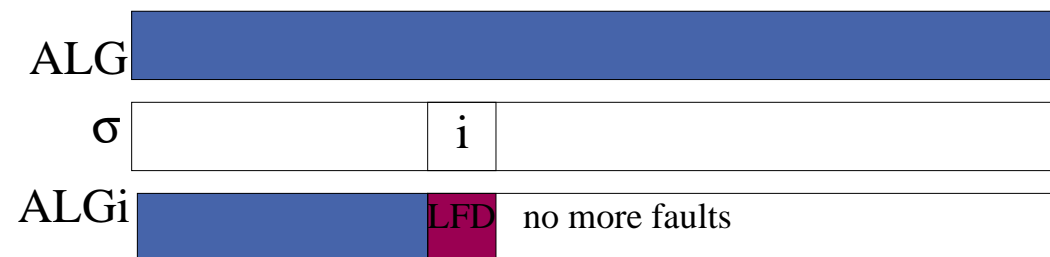


# Longest Forward Distance is optimal

We show: any optimal offline algorithm can be changed to **act like LFD** without increasing the number of page faults.

**Inductive claim:** given an algorithm  $ALG$ , we can create  $ALG_i$  such that

- ☐  $ALG$  and  $ALG_i$  act **identically** on the first  $i - 1$  requests
- ☐ If request  $i$  causes a fault (for **both** algorithms),  
 $ALG_i$  evicts page with **longest forward distance**
- ☐  $ALG_i(\sigma) \leq ALG(\sigma)$

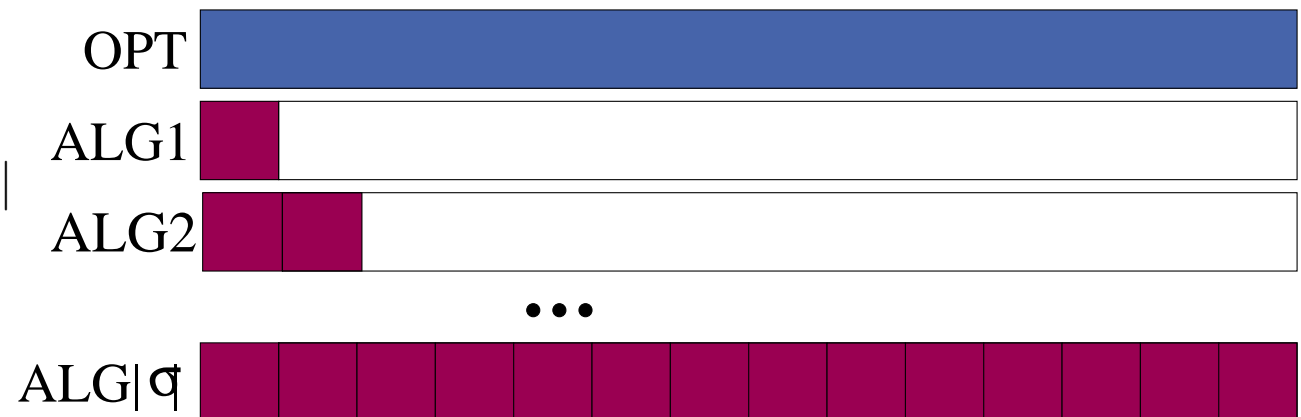


## Using the claim

- ☐ Start with a given request sequence  $\sigma$  and an optimal offline algorithm ALG
- ☐ Use the claim for  $i = 1$  on ALG to get  $\text{ALG}_1$ , which evicts the LFD page on the first request (if needed)
- ☐ Use the claim for  $i = 2$  on  $\text{ALG}_1$  to get  $\text{ALG}_2$

☐ ...

- ☐ Final algorithm  $\text{ALG}_{|\sigma|}$  is equal to OPT



# Proof of the claim

not this time

# Comparison of algorithms

- ☐ OPT is not online, since it looks forward
- ☐ Which is the best online algorithm?
- ☐ LIFO is **not** competitive: consider an input sequence

$$p_1, p_2, \dots, p_{k-1}, p_k, p_{k+1}, p_k, p_{k+1}, \dots$$

- ☐ LFU is also **not** competitive: consider

$$p_1^m, p_2^m, \dots, p_{k-1}^m, (p_k, p_{k+1})^{m-1}$$

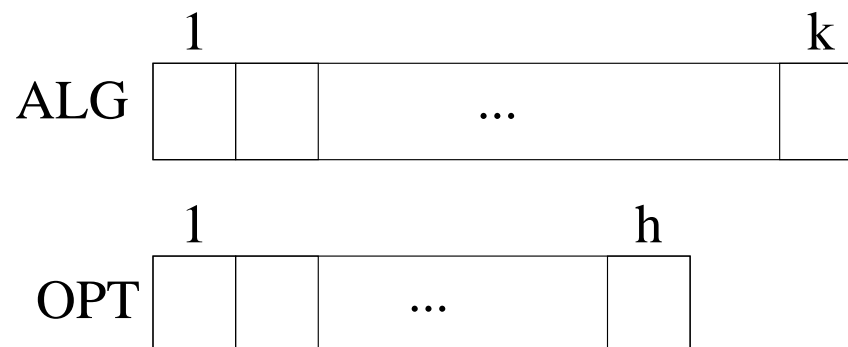
## A general lower bound

- ☐ To illustrate the problem, we show a lower bound for **any** online paging algorithm ALG
- ☐ There are  $k + 1$  pages
- ☐ At all times, ALG has  $k$  pages in its cache
- ☐ There is always one page missing: request this page at each step
- ☐ OPT only faults **once every  $k$  steps**  
 $\Rightarrow$  **lower bound of  $k$**  on the competitive ratio



## Resource augmentation

- ☐ We will compare an online algorithm ALG to an optimal offline algorithm **which has a smaller cache**
- ☐ We hope to get **more realistic** results in this way
- ☐ Size of offline cache =  $h < k$
- ☐ This problem is known as  $(h, k)$ -paging



# Conservative algorithms

- ☐ An algorithm is **conservative** if it has at most  $k$  page faults on any request sequence that contains at most  $k$  distinct pages
- ☐ The request sequence may be **arbitrarily long**
- ☐ LRU and FIFO are conservative
- ☐ LFU and LIFO are **not** conservative (recall that they are not competitive)

# Competitive ratio

**Theorem:** Any conservative algorithm is  $\frac{k}{k-h+1}$ -competitive

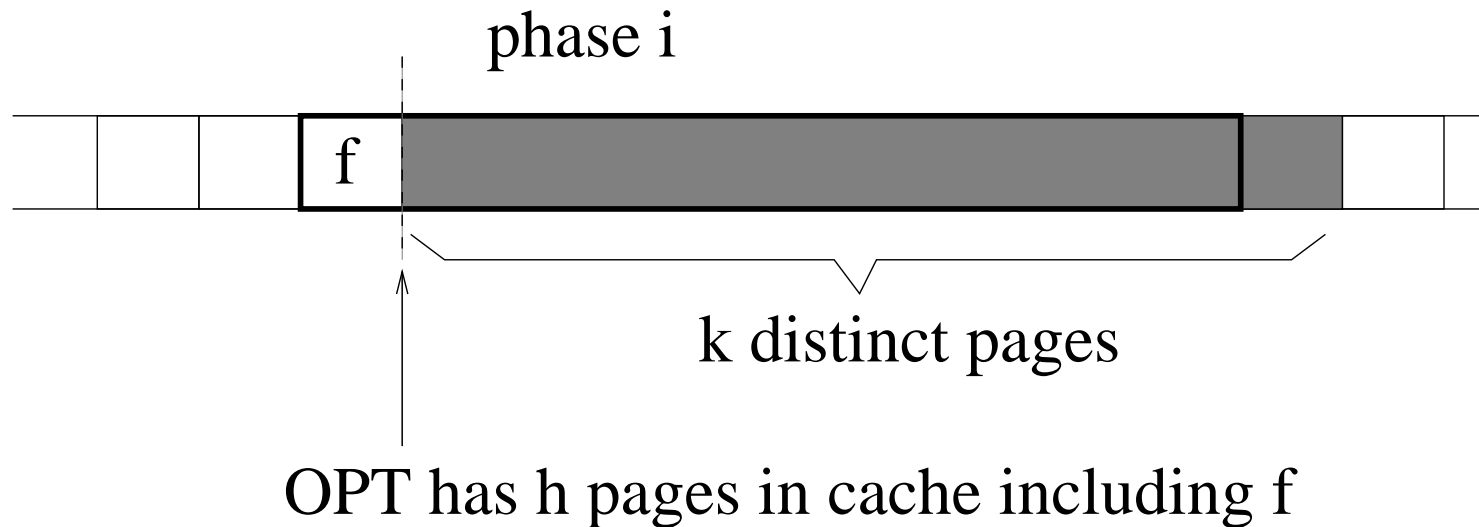
**Proof:** divide request sequence  $\sigma$  into **phases**.

- ☐ Phase 0 is the empty sequence
- ☐ Phase  $i > 0$  is the maximal sequence following phase  $i - 1$  that contains at most  $k$  distinct pages

Phase partitioning **does not depend on algorithm**. A conservative algorithm has at most  $k$  faults per phase.

## Counting the faults of OPT

Consider some phase  $i > 0$ , denote its first request by  $f$



Thus OPT has at least  $k - (h - 1) = k - h + 1$  faults on the grey requests

## Conclusion

- In each phase, a conservative algorithm has  $k$  faults
- To each phase except the last one, we can assign (charge)  $k - h + 1$  faults of OPT

- Thus

$$\text{ALG}(\sigma) \leq \frac{k}{k - h + 1} \cdot \text{OPT}(\sigma) + r$$

where  $r \leq k$  is the number of page faults of ALG in the last phase

- This proves the theorem

# Notes

- ☐ For  $h = k/2$ , we find that conservative algorithms are 2-competitive
- ☐ The previous lower bound construction does not work for  $h < k$
- ☐ In practice, the “competitive ratio” of LRU is a small constant
- ☐ Resource augmentation can give better (more realistic) results than pure competitive analysis

## New results (Panagiotou & Souza, STOC 2006)

- ☐ Restrict the adversary to get more “natural” input sequences
- ☐ **Locality of reference**: most consecutive requests to pages have short distance
- ☐ **Typical memory access patterns**: consecutive requests have either short or long distance compared to the cache size

## Randomized algorithms

- ☐ Another way to avoid the lower bound of  $k$  for paging is to use a **randomized** algorithm
- ☐ Such an algorithm is allowed to use random bits in its decision making
- ☐ Crucial is **what the adversary knows** about these random bits



## Three types of adversaries

- ☐ **Oblivious**: knows only the probability distribution that ALG uses, determines input in advance
- ☐ **Adaptive online**: knows random choices made so far, bases input on these choices
- ☐ **Adaptive offline**: knows random choices in advance (!)

Randomization **does not help** against adaptive offline adversary

We focus on the **oblivious** adversary

# Marking Algorithm

- ☐ **marks** pages which are requested
- ☐ **never evicts a marked page**
- ☐ When **all** pages are marked **and there is a fault**, unmark everything  
(but mark the page which caused the fault)  
(new **phase**)

# Marking Algorithms

Only difference is eviction strategy

- ☐ LRU
- ☐ FWF
- ☐ RMARK: Evict an unmarked page choosen **uniformly at random**

# Competitive ratio of RMARK

**Theorem:** RMARK is  $2H_k$ -competitive

where

$$H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} \leq \ln k + 1$$

is the  $k$ -th harmonic number

## Analysis of RMARK

Consider a phase with  $m$  new pages  
(that are not cached in the beginning of the phase)

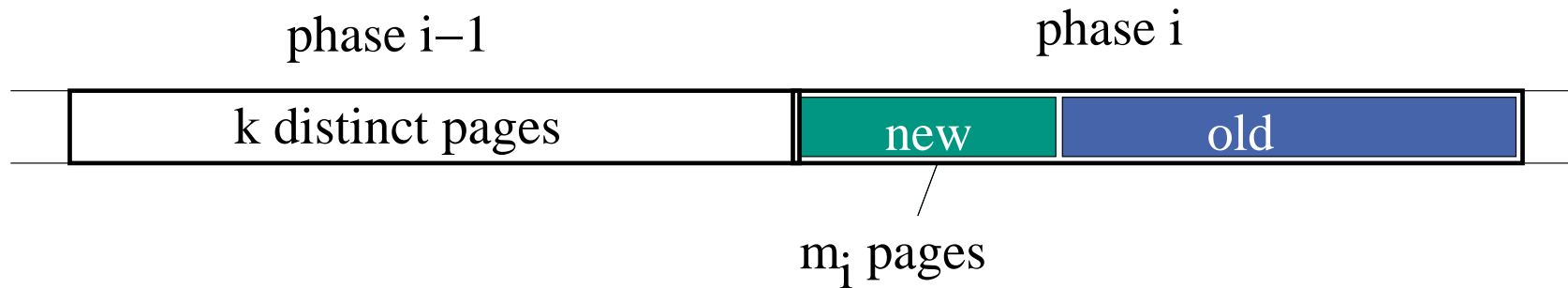
Miss probability when  $j + 1$ st old page becomes marked

$$1 - \frac{\text{\# old unmarked cached pages}}{\text{\# old unmarked pages}} \leq 1 - \frac{k - m - j}{k - j} = \frac{m}{k - j}$$

Overall expected number of faults (including new pages):

$$m + \sum_{j=0}^{k-m-1} \frac{m}{k-j} = m + m \sum_{i=m+1}^k \frac{1}{i} = m(1 + H_k - H_m) \leq mH_k$$

## Lower bound for OPT



- ☐ There are  $m_i$  new pages in phase  $i$
- ☐ Thus, in phases  $i - 1$  and  $i$  together,  $k + m_i$  pages are requested
- ☐ OPT makes at least  $m_i$  faults in phases  $i$  and  $i - 1$  for any  $i$
- ☐ Total number of OPT faults is at least  $\frac{1}{2} \sum_i m_i$

## Upper bound for RMARK

- ☐ Expected number of faults in phase  $i$  is at most  $m_i H_k$  for RMARK
- ☐ Total expected number of faults is at most  $H_k \sum_i m_i$
- ☐ OPT has at least  $\frac{1}{2} \sum_i m_i$  faults
- ☐ Conclusion: RMARK is  $2H_k$ -competitive

## Randomized lower bound

**Theorem:** No randomized can be better than  $H_k$ -competetive against an oblivious adversary.

**Proof:** not here



## Discussion

- ☐  $H_k \ll k$
- ☐ The upper bound for RMARK holds against an oblivious adversary  
(the input sequence is **fixed in advance**)
- ☐ No algorithm can be better than  $H_k$ -competitive
- ☐ Thus, RMARK is optimal apart from a factor of 2
- ☐ There is a (more complicated) algorithm that is  $H_k$  competitive

## Why **competitive analysis**?

There are many models for “decision making in the absence of complete information”

- ☐ Competitive analysis leads to algorithms that would not otherwise be considered
- ☐ Probability distributions are rarely known precisely
- ☐ Assumptions about distributions must often be unrealistically crude to allow for mathematical tractability
- ☐ Competitive analysis gives a **guarantee** on the performance of an algorithm, which is essential in e.g. financial planning

# Disadvantages of competitive analysis

- Results can be too pessimistic (adversary is too powerful)
  - Resource augmentation
  - Randomization
  - Restrictions on the input
  
- Unable to distinguish between some algorithms that perform differently in practice
  - Paging: LRU and FIFO
  - more refined models