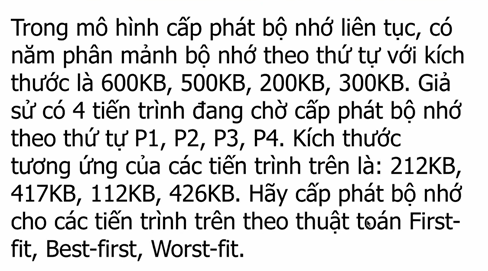
Bộ nhớ

**1. Mô hình cấp phát bộ nhớ liên tục:**



1 phân mảnh sau khi cấp phát xong, tiếp tục sử dụng phần dư cấp phát

First-fit: Trong ds các phân mảnh, lựa chọn phân mảnh đầu tiên đáp ứng yêu cầu

P1: 388, 500, 200, 300

P2: 388, 83, 200, 300

P3: 276, 83, 200, 300

P4: Chờ

Best-fit: Trong ds các phân mảnh, lựa chọn phân mảnh đủ đáp ứng có kích thước nhỏ nhất

P1: 600, 500, 200, 88

P2: 600, 83, 200, 88

P3: 600, 83, 88, 88

P4, 174, 83 ,88 ,88

Worst-fit: Trong ds các phân mảnh, lựa chọn phân mảnh đủ đáp ứng có kích thước lớn nhất

P1: 388, 500, 200, 300

P2: 388, 83, 200, 300

P3: 276, 83, 200, 300

P4: Chờ

**2. Phân đoạn**

Giả sử trong quá trình quản lý bộ nhớ ảo dạng phân đoạn, hệ điều hành duy trì Segment Table:

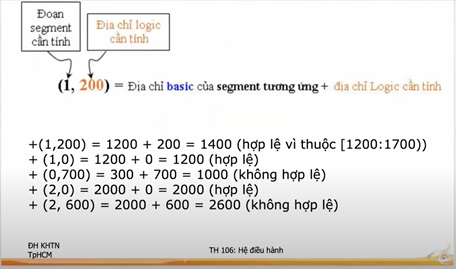
Segment Base Limit

0 300 700

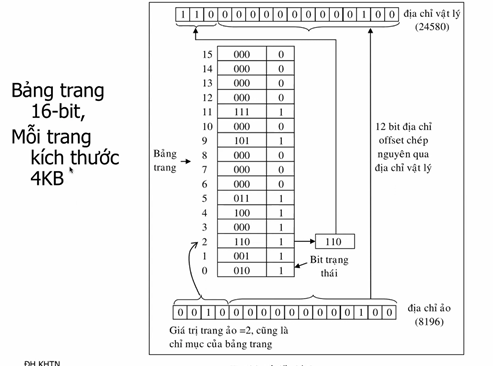
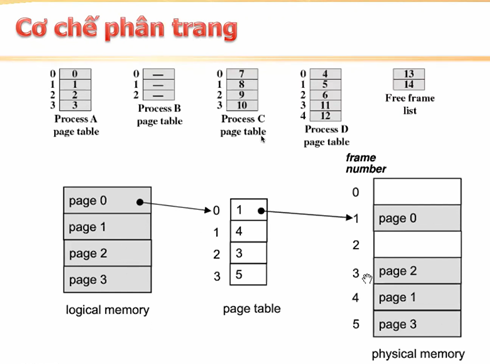
1 1200 500

2 2000 600

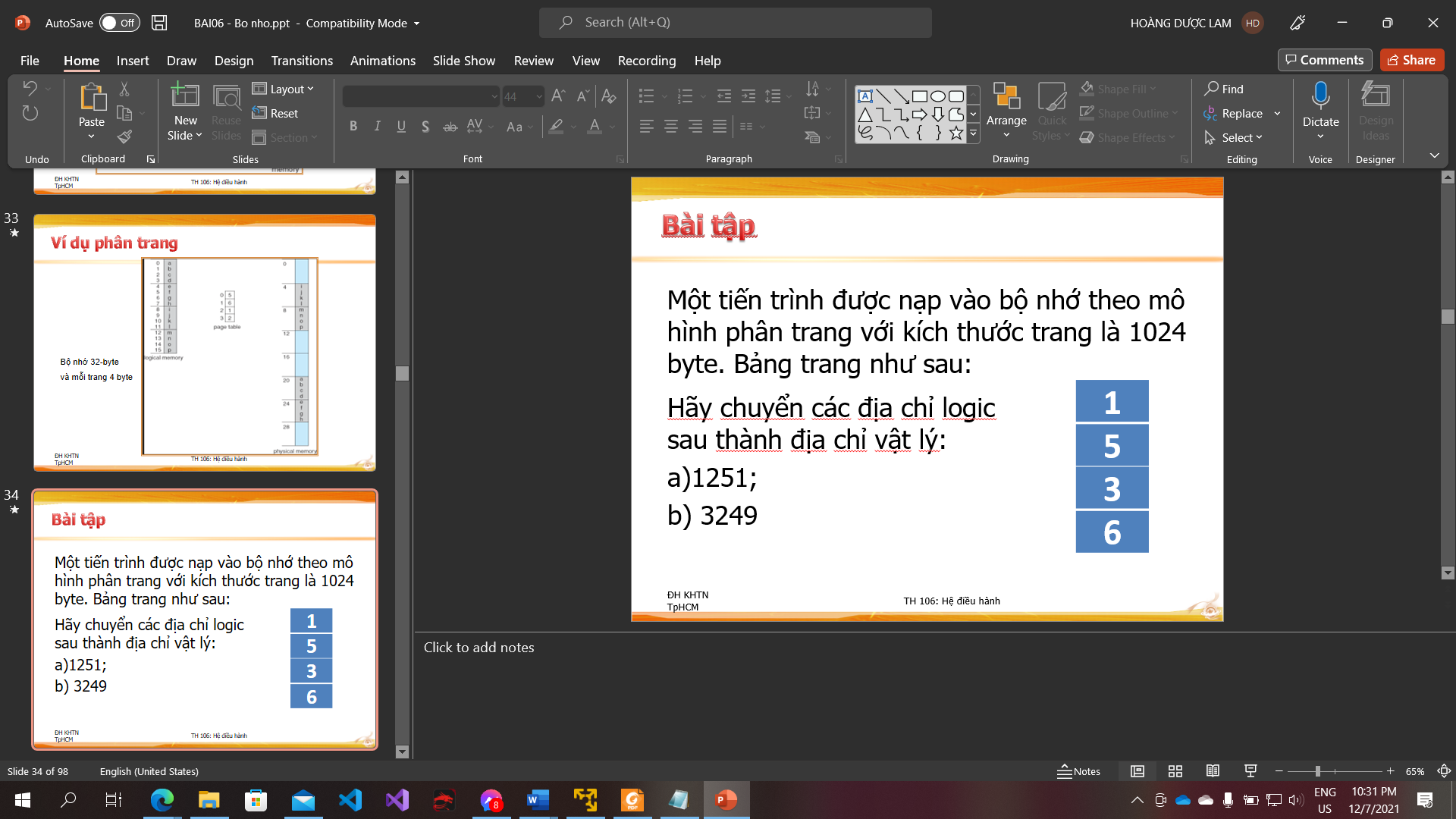
Tính địa chỉ vật lý cho mỗi địa chỉ logic sau: (1,200), (1,0), (0,700), (2,0), (2,600)



**3. Phân trang**



Bài tập:



Giải:

a. 1251

pagesize = 1024

p = 1251 / pagesize = 1251 / 1024 = 1

d = 1251 % pagesize = 1251 % 1024 = 227

Tra bảng phân trang với p = 1 -> f=5

—> Đ/c vật lý = f \* pagesize + d = 5 \* 1024 + 227 = 5347

b. 3249

pagesize = 1024

p = 3249 / pagesize = 3249 / 1024 = 3

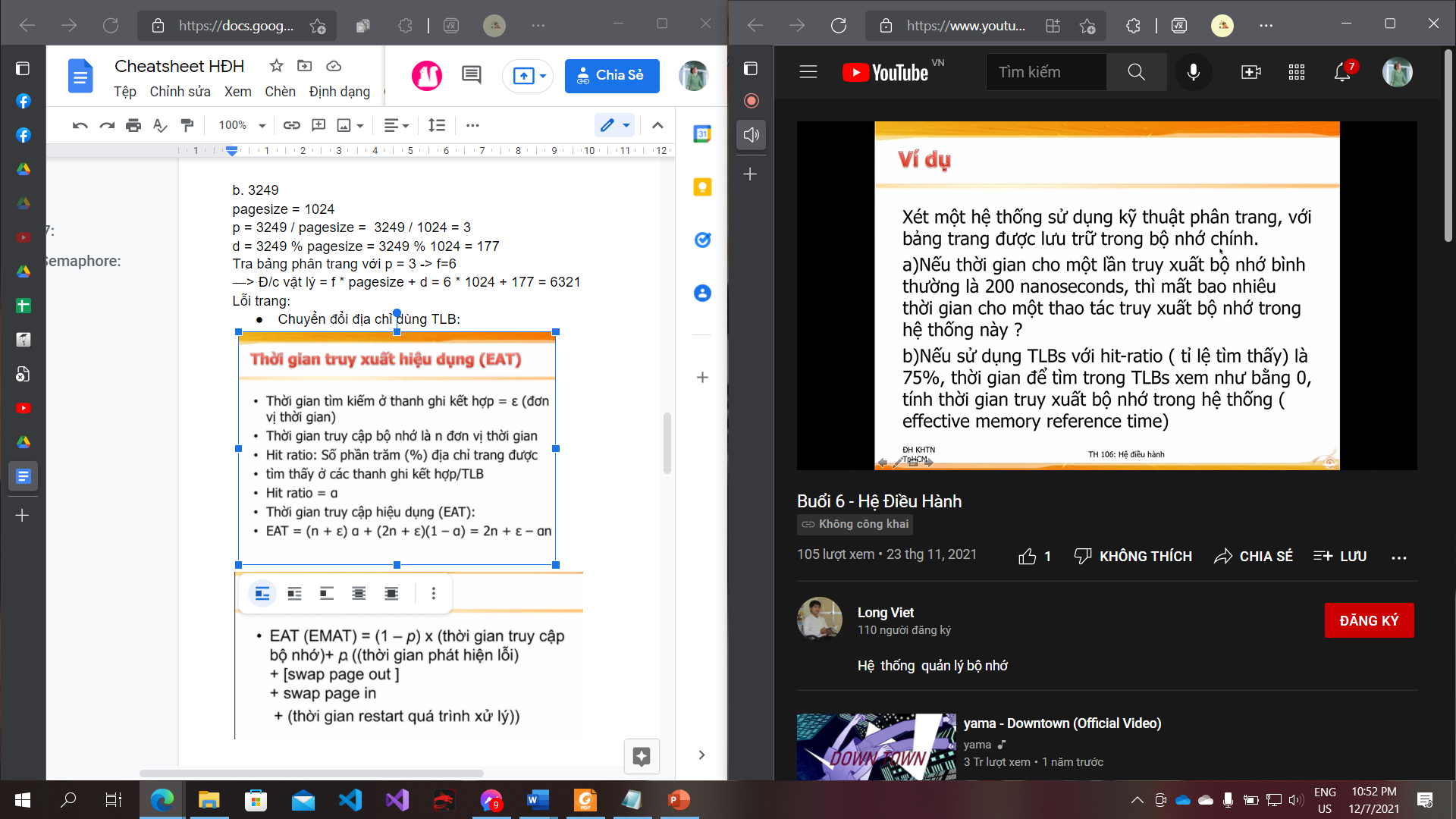
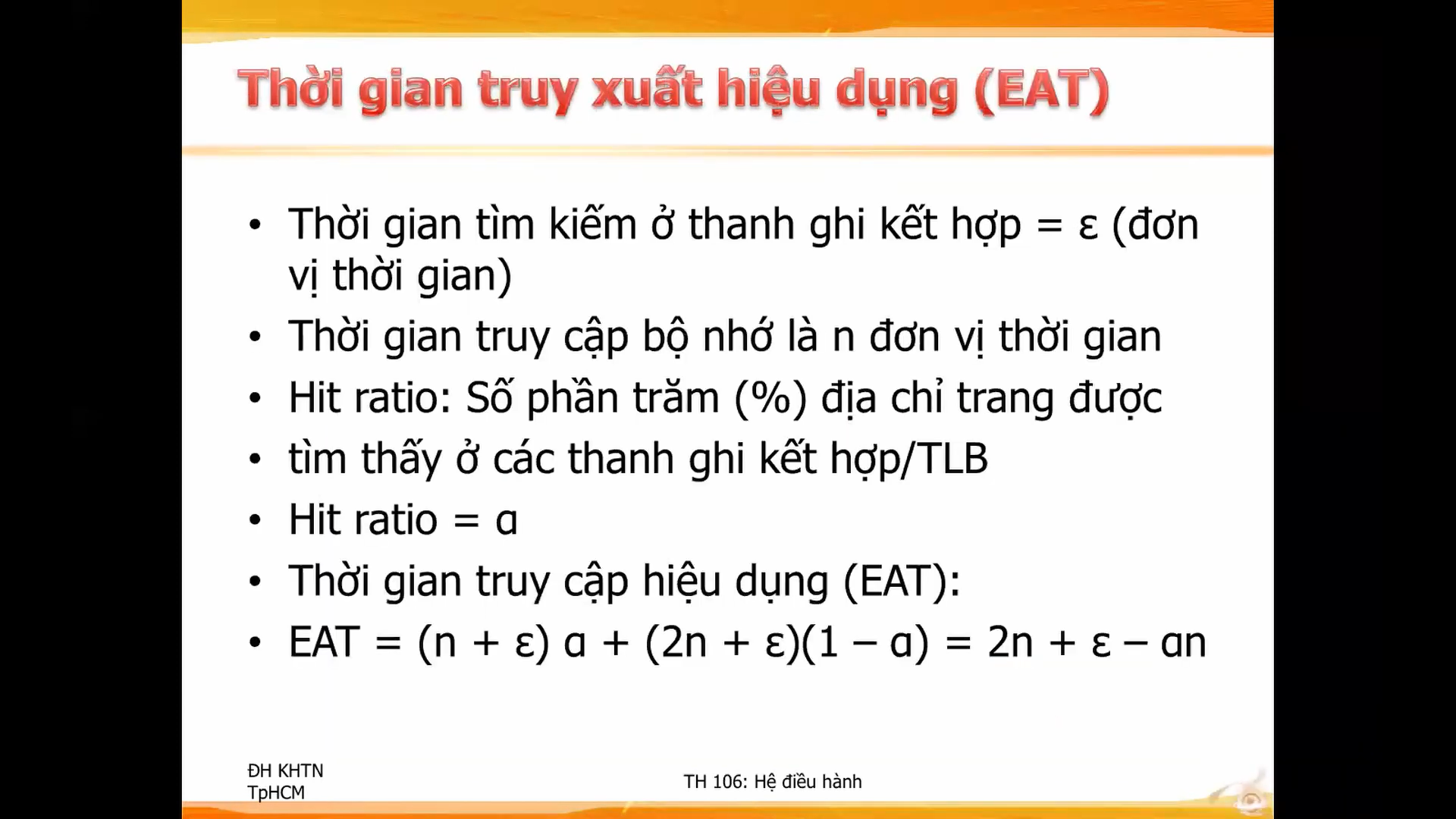
d = 3249 % pagesize = 3249 % 1024 = 177

Tra bảng phân trang với p = 3 -> f=6

—> Đ/c vật lý = f \* pagesize + d = 6 \* 1024 + 177 = 6321

**EAT:**

* Sử dụng bảng trang: Mọi theo tác truy cập dữ liệu/lệnh cần tới 2 lần truy cập bộ nhớ( 1 cho bảng trang, 1 cho dữ liệu/lệnh)
* Chuyển đổi địa chỉ dùng TLB:



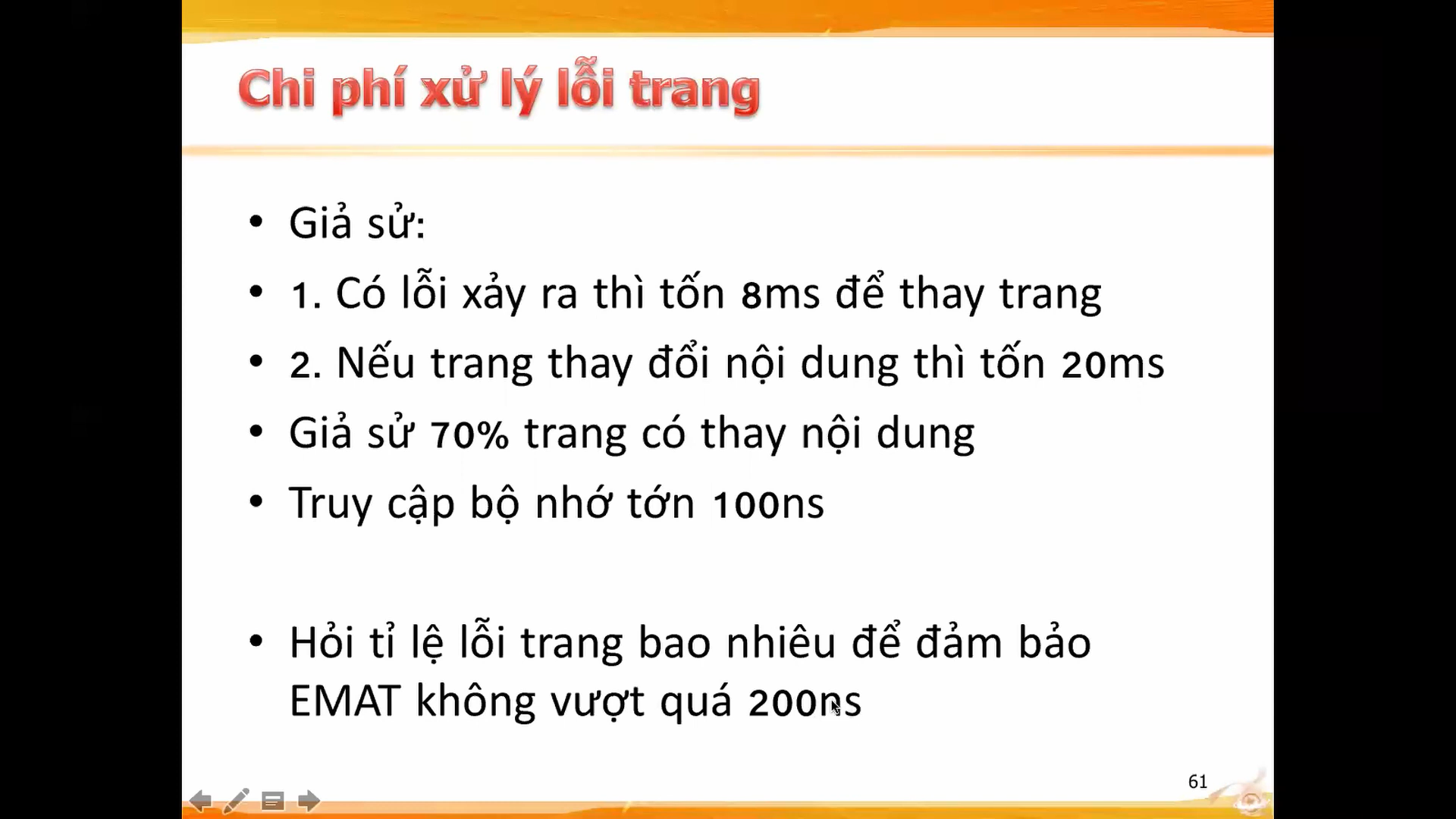
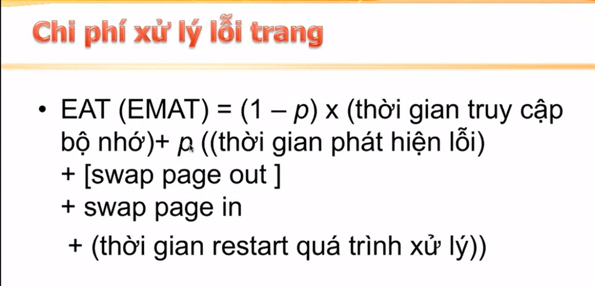
Giải:

1. Truy cập theo kĩ thuật phân trang bình thường => Mất 2 lần truy cập bộ nhớ

-> EAT = 400

1. EAT = (n+e)\*a+(2n+e)(1-a) = 200\*0.75+400\*0.25=250

**4. Lỗi trang:**



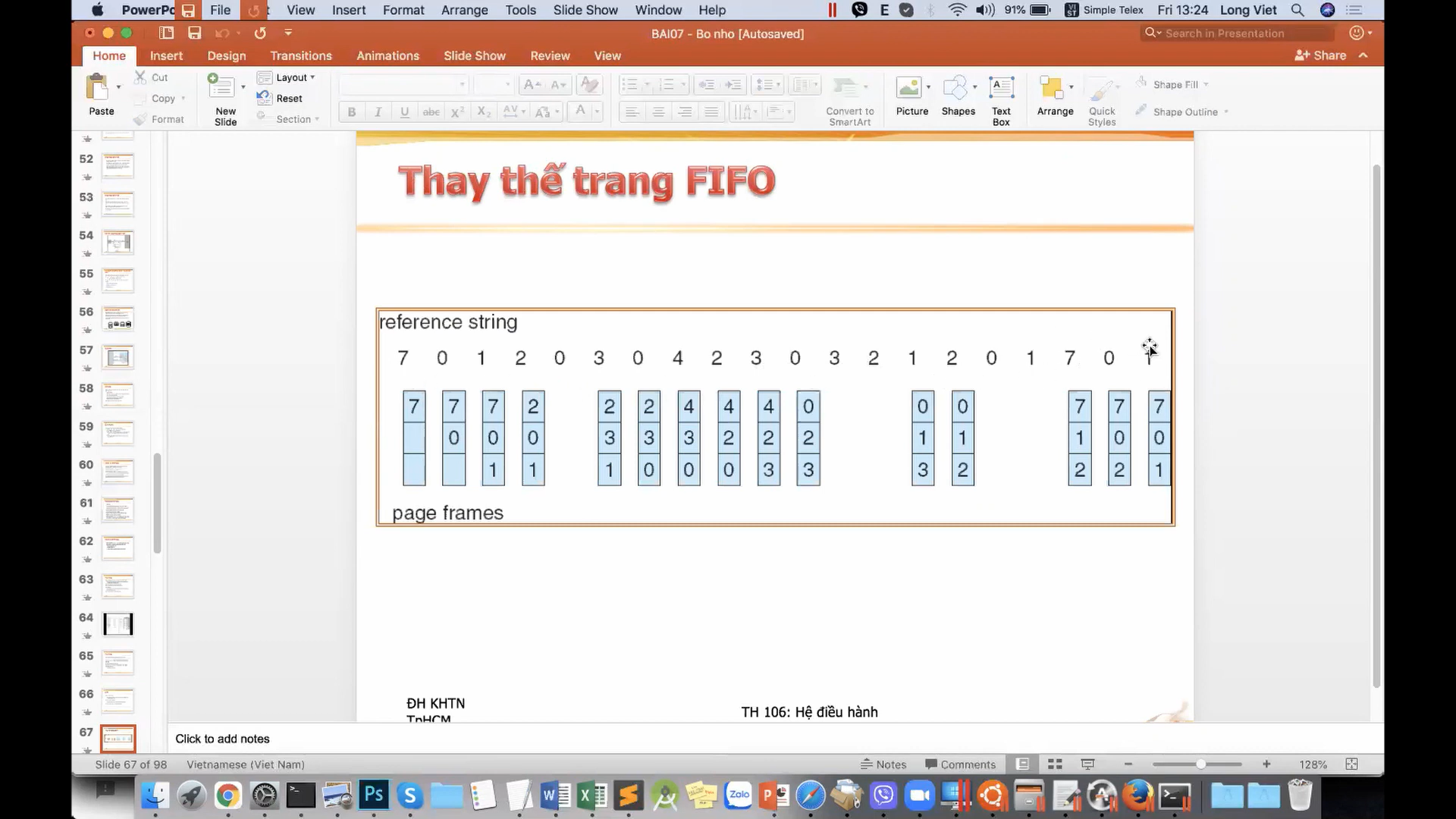
Giải: Gọi c là tỷ lệ thay trang có thay đổi nội dung = 0.7

200 > (1-p) x 100 + p ((1-0.7) \* 8 \* 10^6 + 0.7 \* 20 \* 10^6))

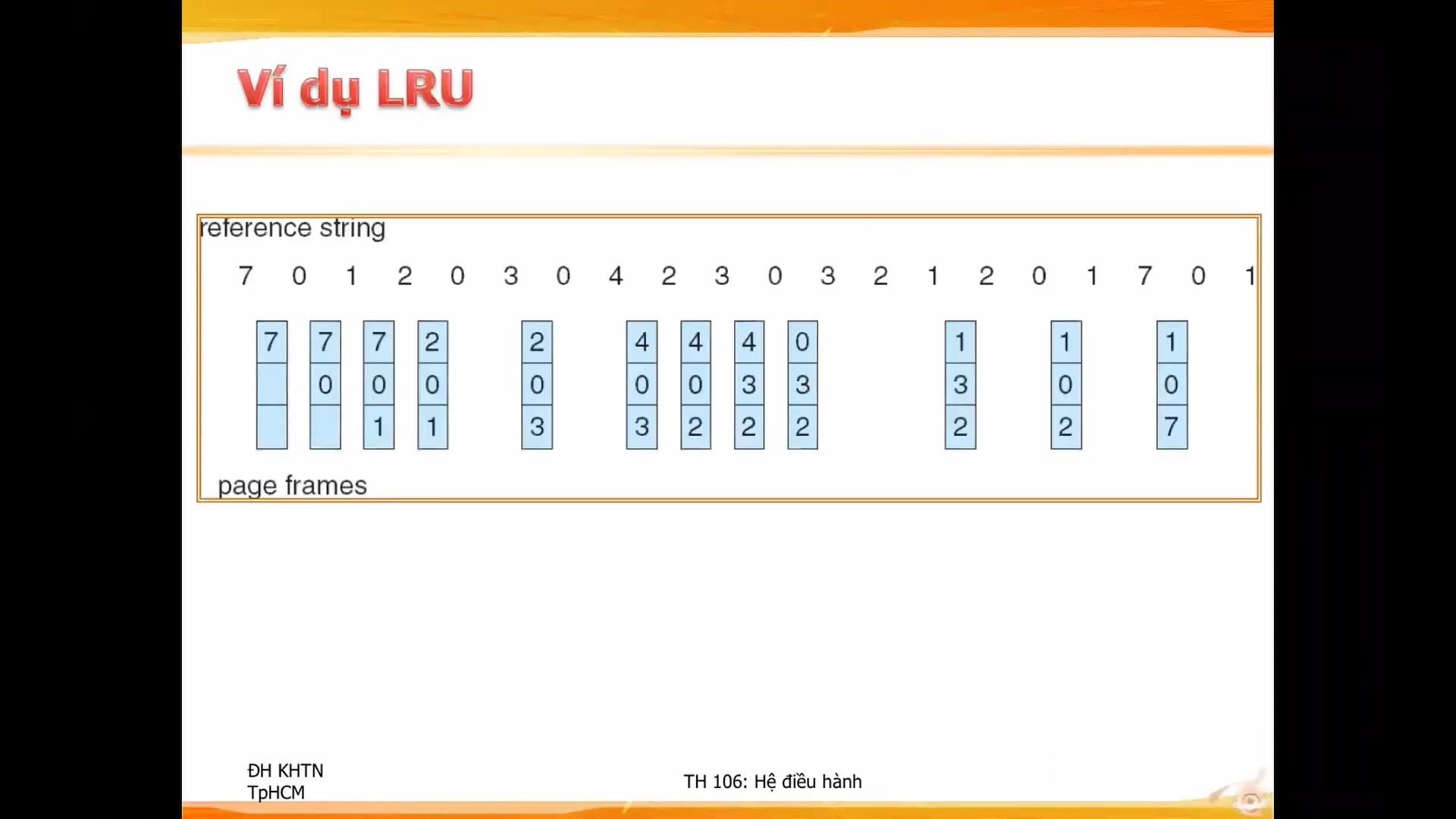
200 > 100 - 100p + p\*10^6\*16.4

p< 6.097 \* 10^-6 => 6 lỗi trang / 1 triệu lần truy cập

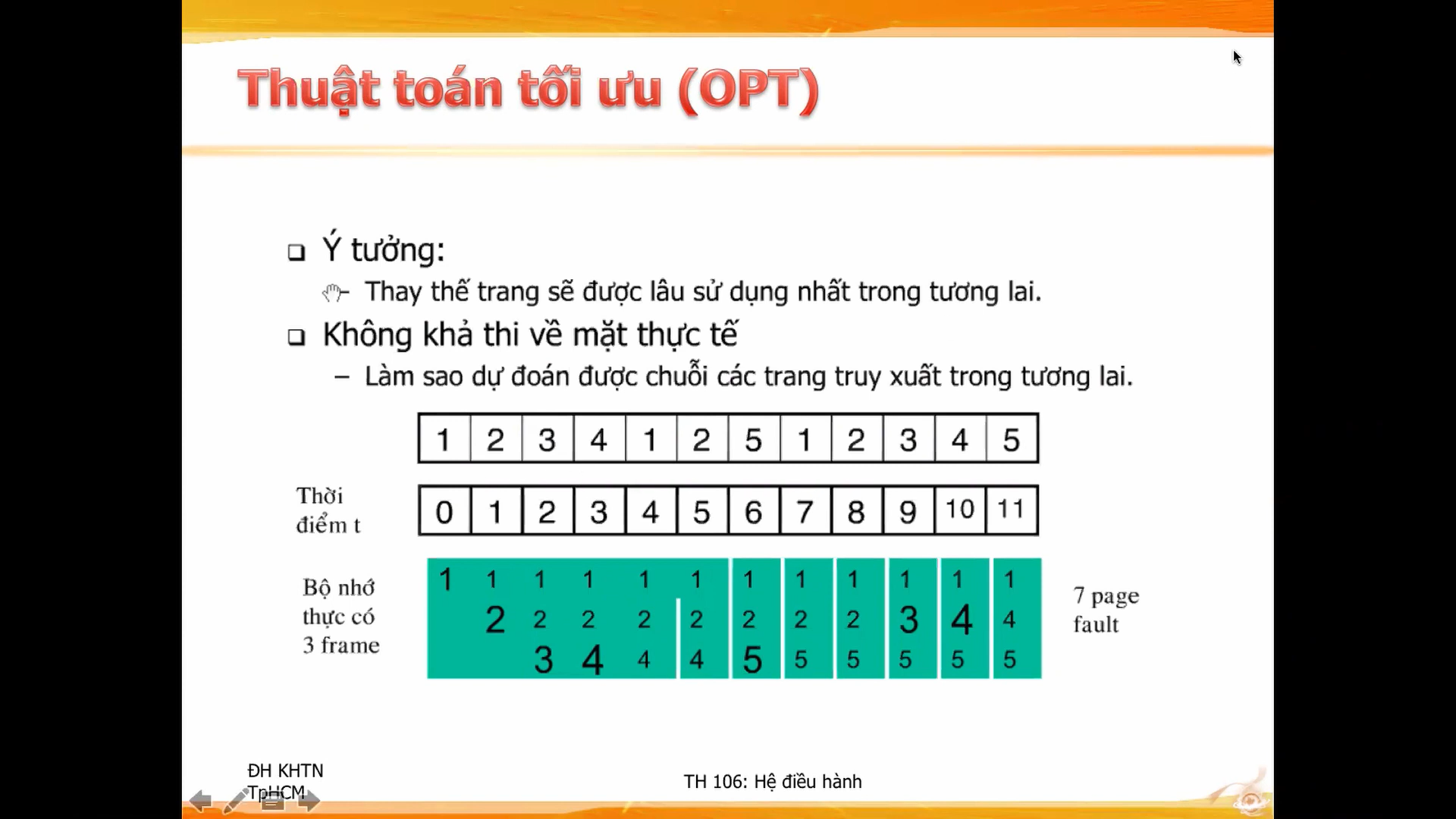
**5. Thay thế trang:**



FIFO: lựa chọn frame đầu tiên để thay thế



LRU: Chọn frame ít sử dụng gần đây nhất để thay thế



OPT: Chọn frame sẽ được lâu sử dụng nhất trong tương lai để thay thế

**Tóm tắt chương 7:**

* Nhiều tiến trình truy xuất tài nguyên không nên được chia sẻ => **Tranh đoạt điều khiển (Race condition)**
* Các tiến trình cần phải thực hiện đúng trình tự của nó => **Phối hợp xử lý**
* **Miền găng (CS)** là đoạn code có chứa tài nguyên có khả năng bị Race condition

***Cách giải quyết Semaphore:***

* Gồm 2 hàm Up và Down
* Down sẽ hạ value Semaphore xuống 1. Và nếu value này < 0 thì sẽ add tiến trình vào hàng đợi
* Up sẽ nâng value Semaphore lên 1. Và nếu value này >= 0 thì sẽ remove tiến trình gần nhất muốn thực hiện khỏi hàng đợi

*Ví dụ 1*: 1 quầy chỉ được phục vụ 3 khách hàng => Semaphore S = 3

Down(S); //Nếu có 3 khách tương ứng S bị down 3 lần và = 0, khách sau muốn vào sẽ bị down xuống  
 -1 và vào hàng đợi do < 0, không thể enter

EnterCounter();

LeaveCounter();

Up(S); //Up Semaphore thành 0 và remove thằng khách đang đợi khỏi hàng đợi, khách có thể Enter

* Ví dụ trên là để giải quyết Phối hợp xử lý. Còn nếu muốn giải quyết Race Condition thì như sau:

*Ví dụ 2*: Một buổi phỏng vấn chỉ được phỏng vấn 1 ứng viên và không có ứng viên nào khác xen ngang

Sempahore S = 1;

Down(S);

Interview;

Up(S)

**Tóm gọn**:

* Xử lý Race Condition thì down ngay trước tiến trình và up ngay sau khi nó kết thúc. Semaphore thường bằng 1
* Xử lý Đồng bộ (P1 rồi mới tới P2) thì đặt down ngay trước P2 và Up ngay sau P1. Semaphore thường = 0. Một số trường hợp khác có quy ước số lượng thì sẽ khác.

***Cách giải quyết Monitor:*** Giải quyết cả Race Condition và Đồng bộ xử lý bằng cơ chế Wait() và Signal()

* Tạo **1 class Monitor** gồm các hàm để quản lý các hàm tiến trình => Sẽ không có tiến trình nào khác xen ngang hàm của class đang chạy => Thỏa **Race Condition**
* Gồm **điều kiện c**. Tiến trình chưa được chạy thì sẽ có hàm Wait(c). Khi tiến trước đó hoàn thành thì sẽ Signal(c) để hủy hàm Wait(c) đi => Thỏa **Đồng bộ xử lý**

***Cụ thể:***

Class Monitor {

condition c;

Process P1 {

CS1; // **CS** là miền găng ở trên giảng rồi tự hiểu nhe

Signal(c);

}

Process P2 {

Wait(c);

CS2

}

}

Monitor M;

M.P1;

M.P2;

***Ví dụ:***Không được remove khi Giỏ đang empty và không được add khi Giỏ đang full

