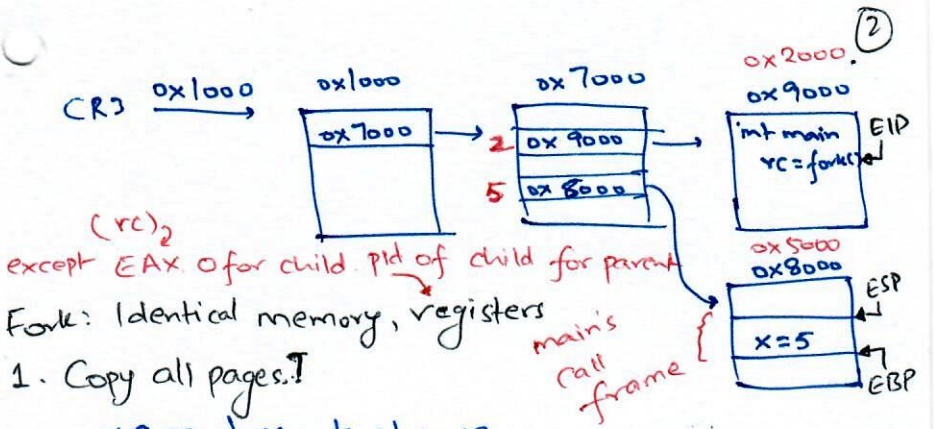
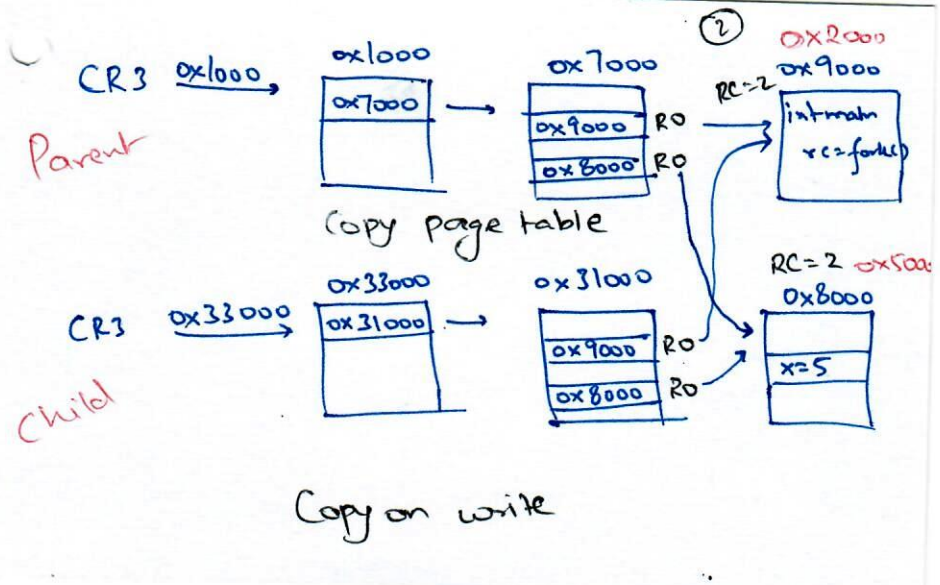
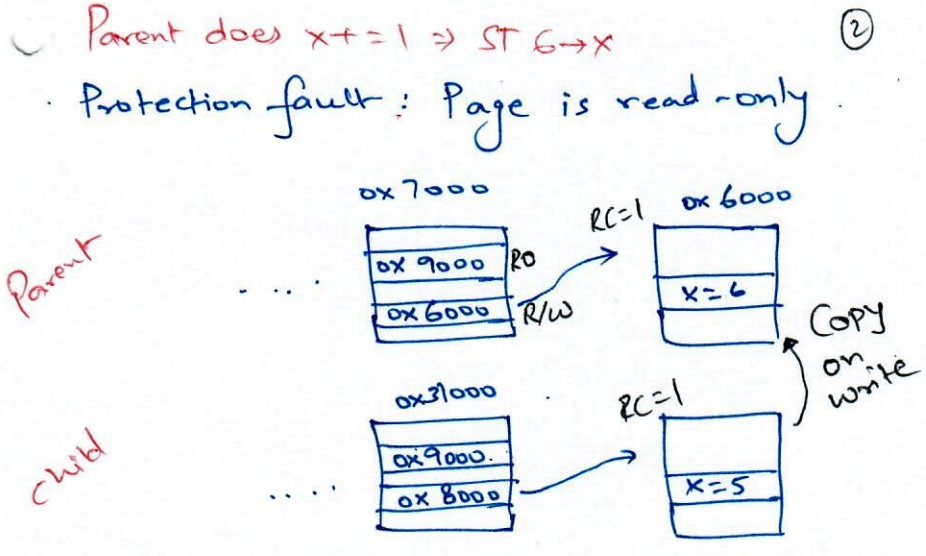

CR3 0x1000
0x1000
0x7000
0x7000
2
0x9000
5
0x8000
0x2000
0x9000
int main
rc = fork()
EIP
0x5000
0x8000
ESP
x=5
EBP
main's call frame
(rc)2
except EAX 0 for child pid of child for parent
Fork: Identical memory, registers
1. Copy all pages.
0x9000 does not change
2. Copy on write


CR3 0x1000
Parent
0x1000
0x7000
0x7000
0x9000
RO
0x8000
RO
RC=2
0x2000
0x9000
int main
rc = fork()
Copy page table
CR3 0x33000
Child
0x33000
0x31000
0x31000
0x9000
RO
0x8000
RO
RC=2
0x8000
x=5
Copy on write


Parent does x+=1 ⇒ ST 6→x
· Protection fault: Page is read-only
Parent
0x7000
0x9000
RO
0x6000
R/W
RC=1
0x6000
x=6
Copy on write
Child
0x31000
0x9000
0x8000
RC=1
x=5

## Shadow page table overheads ③

→ 1 fault for every PTE R/w → R/O

→ Changing CR3 to child
: Copy page table to shadow page table

→ At a write, protection fault at VMM → forwarded to guest → guest copies page, updates PTE → again a fault → hypervisor updates shadow page table.

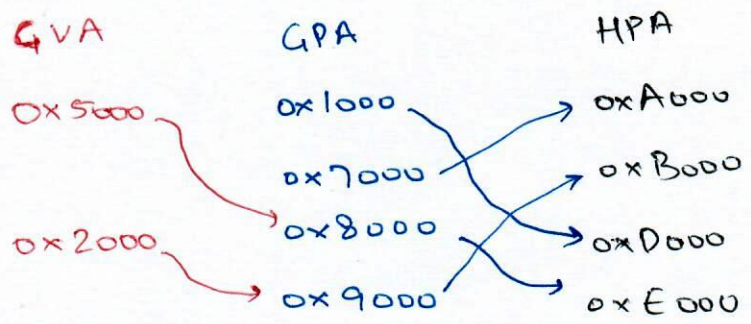
---

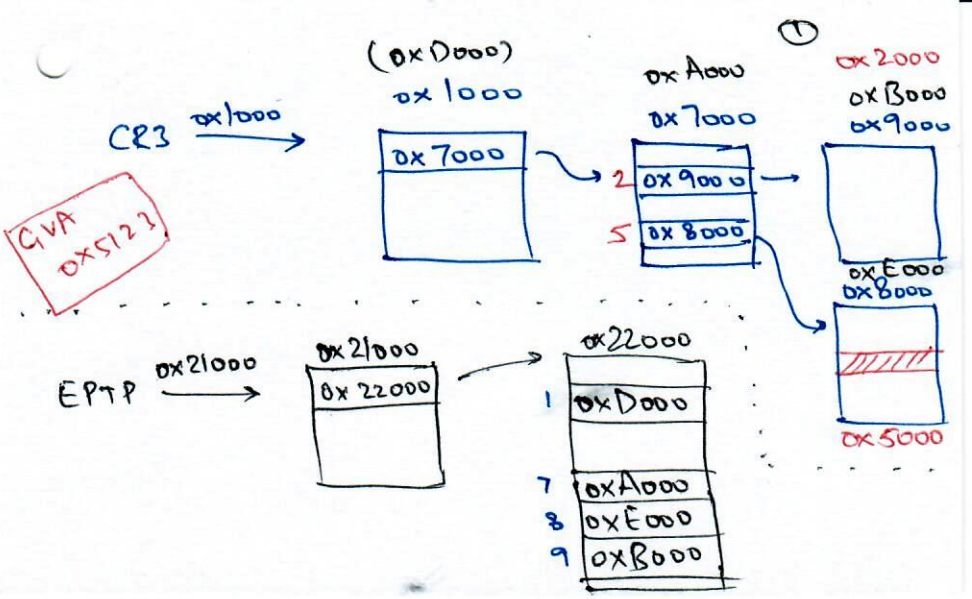
## Overheads: Pentium 4 672 ②

- Fork a proc, wait for child to end, fork again → 4.4 x slowdown
- PTE modification
  native → single cycle store
  Shadow PT → 12733 cycles

---

## Extended Page Tables ①






---

Page walk GVA 0x5123 ③

0x1000 ? → 0x21000 → 0x22000 → 0xD000

0x7000 ? → 0x21000 → 0x22000 → 0xA000

0x8000 ? → 0x21000 → 0x22000 → 0xE000

8 memory references

m level from GPA → HPA

n level for GVA → GPA

⇒ m+n+m*n references

0x5 → 0xE cached in TLB

---

- Hardware support for 2-D address translations ②
- Changing CR3 need not trap ⇒ fast ctx sw
- Modifying PTEs need not trap ⇒ fast fork

(4)

Table 1:- Instruction and data translations

3.9x - 4.57 x slowdown

Figure 3:- Percentage of unique page entries for each 2D page walk reference

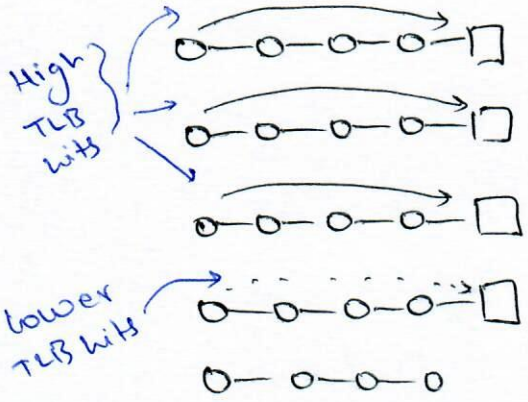
Most PTEs are common on higher levels

Figure 6:- 86-93% of native performance

Accelerating 2D page walks for virtualized systems ← Paper [ASPLOS'08]

---

Two main ideas: 1) Nested TLB (2)




Also cache GPA → HPA in NTLB

Figure 8

+7.3% performance improvement end-to-end

---

2) Large pages (2)

→ Increases TLB reach → Also helps in shadow page tables

→ Reduces m and n

⇒ Less number of references in 2D page walk

Reduces PFs

2MB nested pages give 21-43% fewer TLB misses than 4K nested pages

(1)

# Memory management

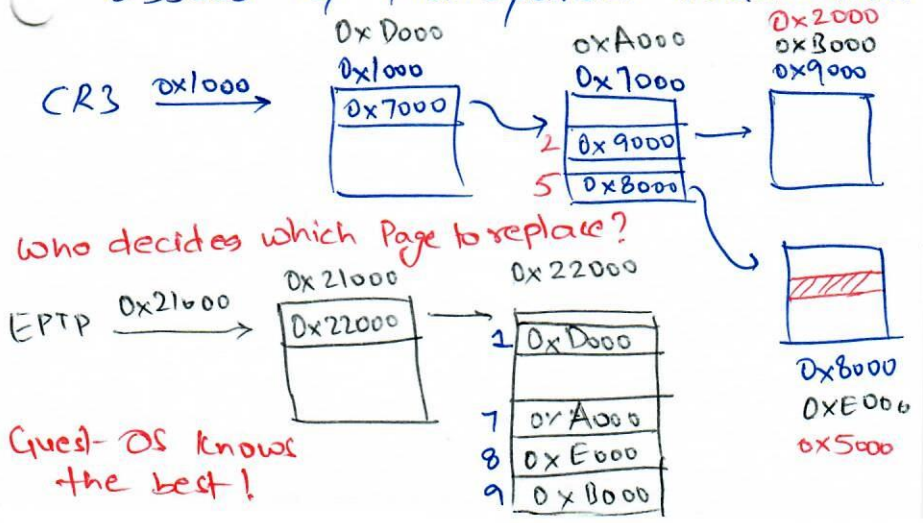
Reclamation
- Ballooning

Sharing memory
- Content based page sharing

---

(2)

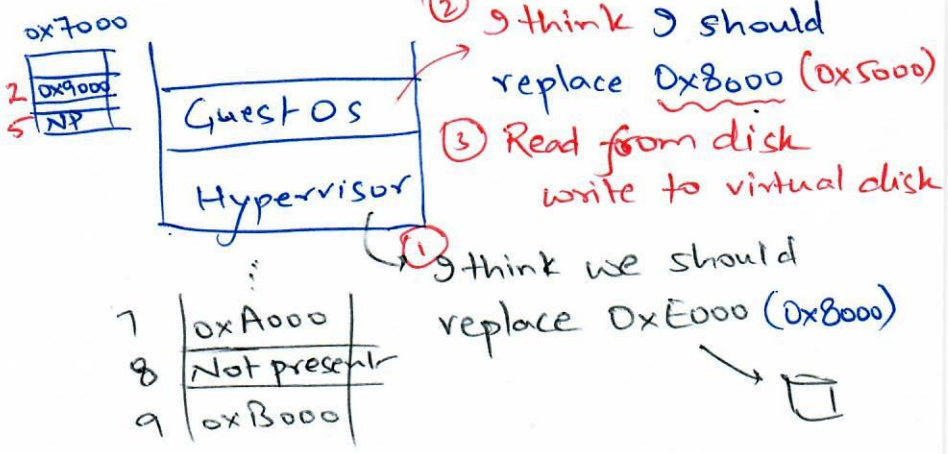
## Issues w/ transparent reclamation



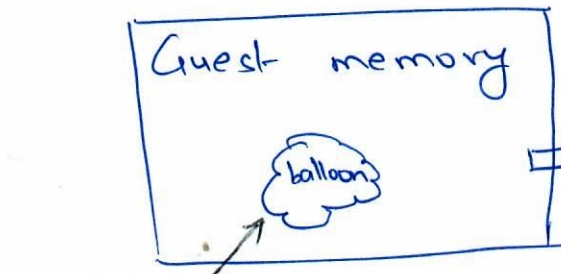

---

(3)

## Double paging problem



## Memory ballooning (2)




Guest OS moves pages to disk to give to balloon driver

Hypervisor inflates on memory pressure

Figure 2 OSDI'02 — similar perf to VM initialized w/ less memory

## Memory ballooning (2)

- Gives control to Guest OS
- Effective in taking memory

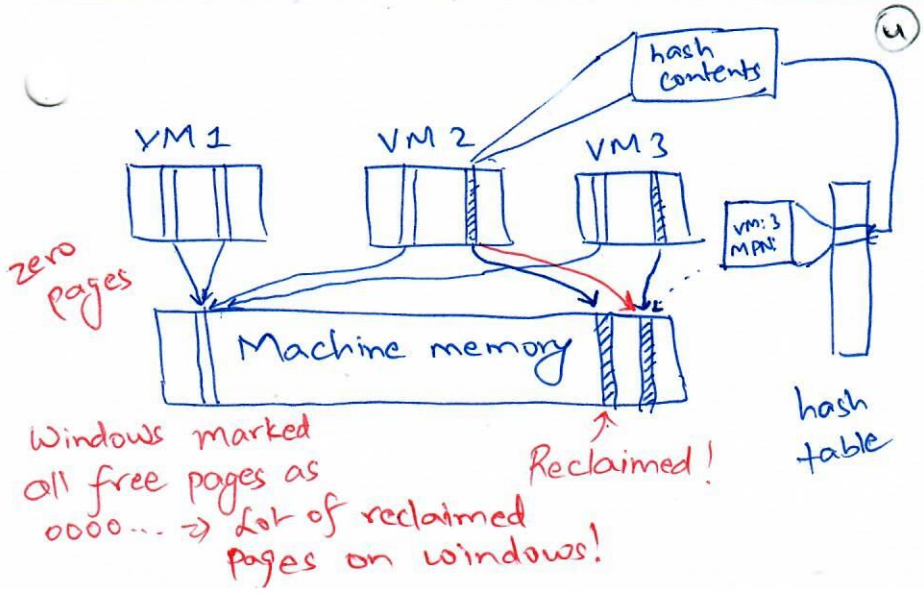
If Guest OS refuses/disables ballooning driver
⇒ hypervisor forcibly takes pages
Guest OS may suffer from double paging

## Content-based page sharing (4)

- scan a random page periodically
- Hash ↳ opt: R/o page marked by OS
- Match ⇒ fully match page
- Match ⇒ Mark CoW. Reclaim

⇒ Transparent to Guests!

Figure 5 [OSDI02] 7-32% reclaimed memory




## Interesting trivia



Linux KSM (kernel samepage merging) / .. shared memory

- Similar hash-based implementation [Nov '08]
- VM Ware Patent!
- Red-black tree based implementation [Apr '09]