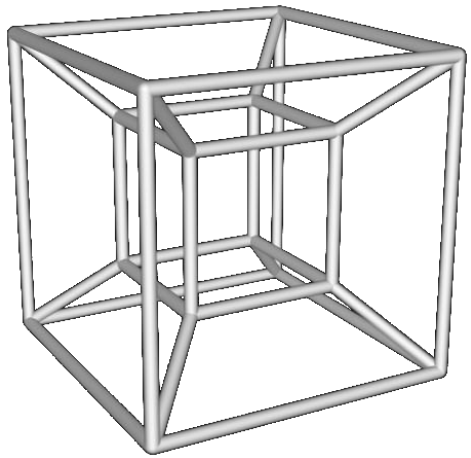
Tesseract: Reconciling Guest I/O and Hypervisor Swapping in a VM



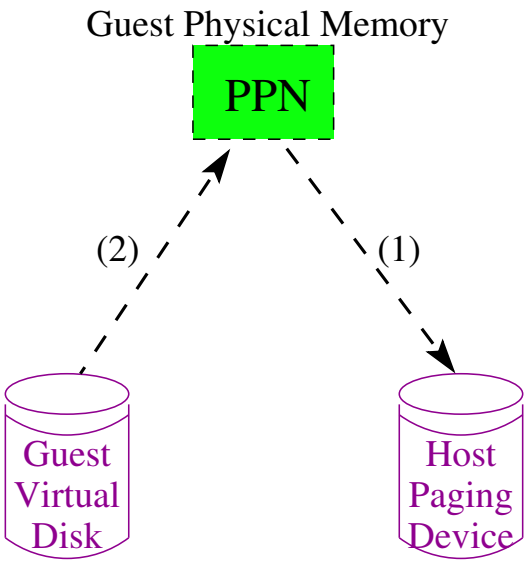
Kapil Arya
Northeastern University

Yury Baskakov
VMware, Inc.

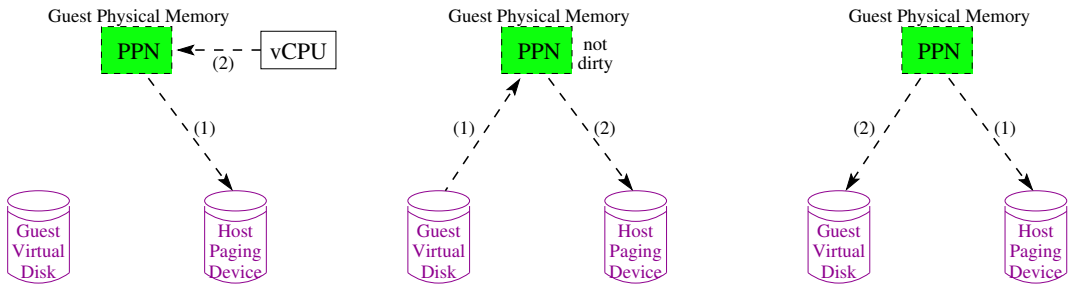
Alex Garthwaite
Twitter, Inc.

March 1, 2014

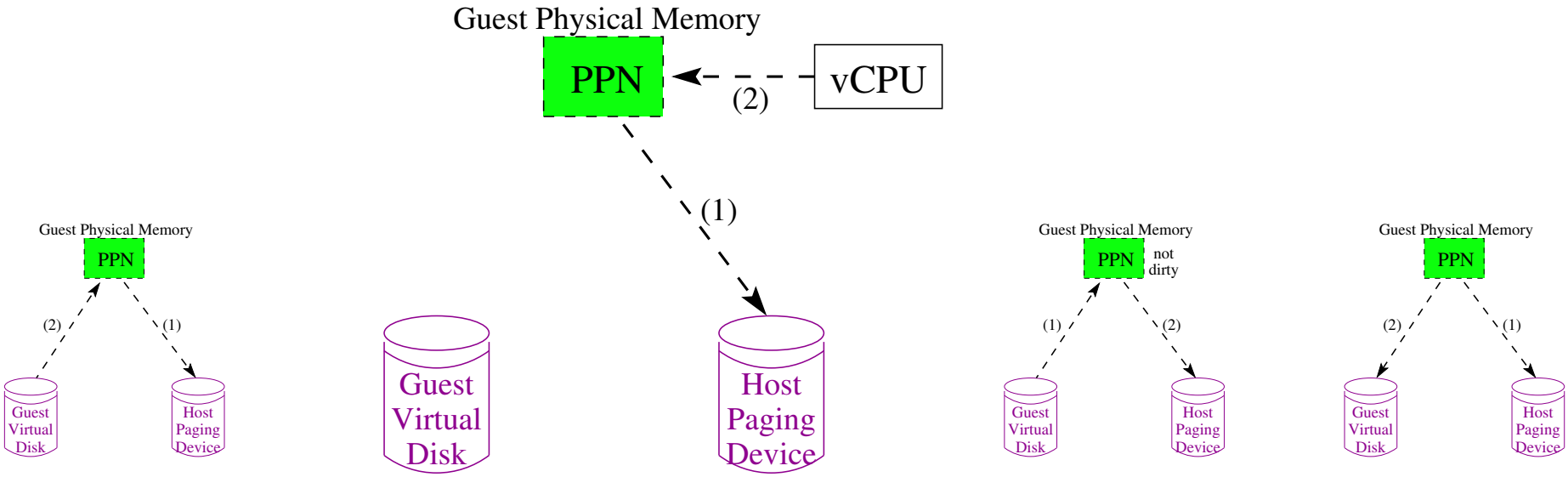
Hypervisor Swapping and Guest I/O: Sub-Optimal Behavior



Host swapout followed by guest disk read

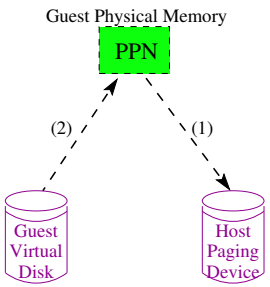


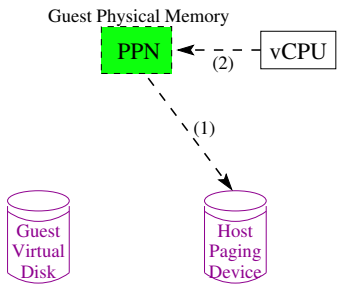
Hypervisor Swapping and Guest I/O: Sub-Optimal Behavior

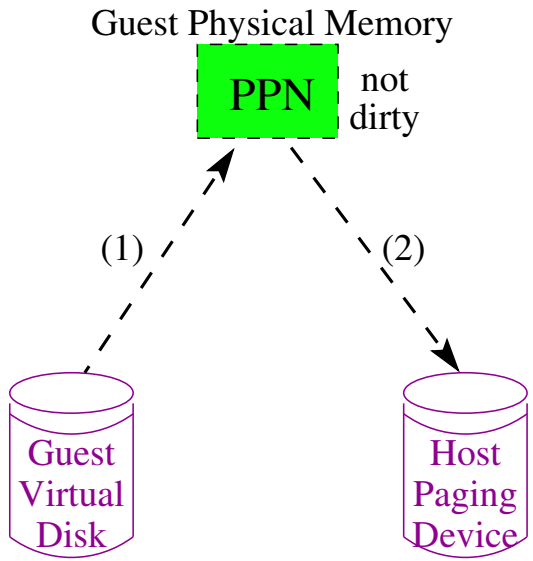


Host swapout followed by guest overwriting the entire page (e.g. zeroing)

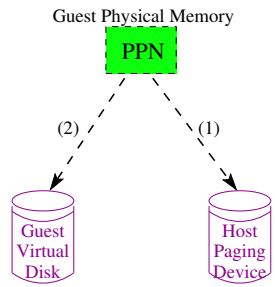
Hypervisor Swapping and Guest I/O: Sub-Optimal Behavior



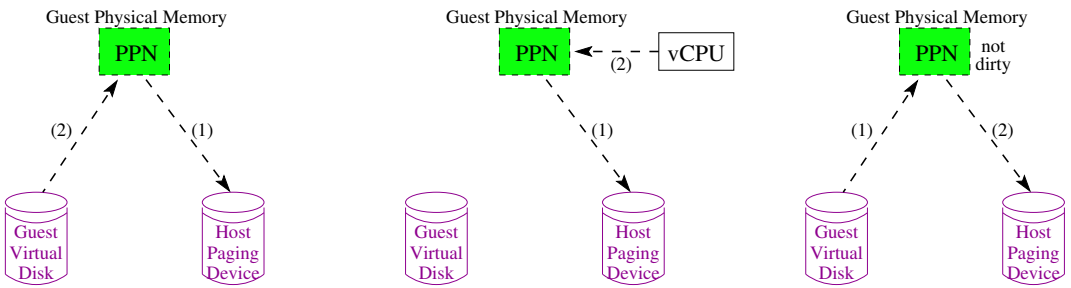


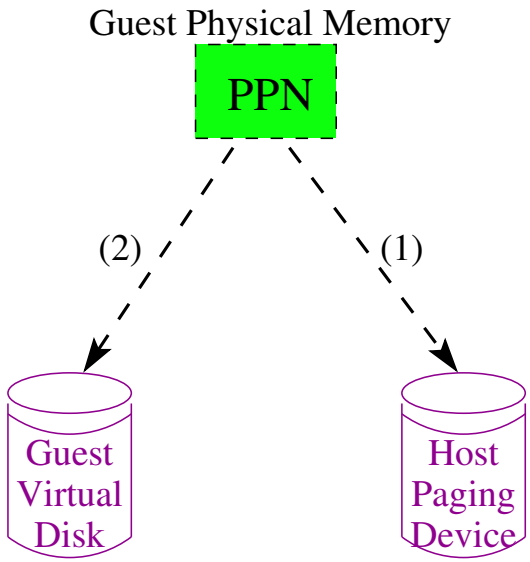


Host swap out of an unmodified guest page



Hypervisor Swapping and Guest I/O: Sub-Optimal Behavior

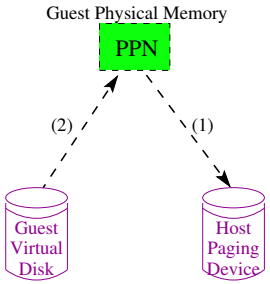




Host swapout followed by guest disk write

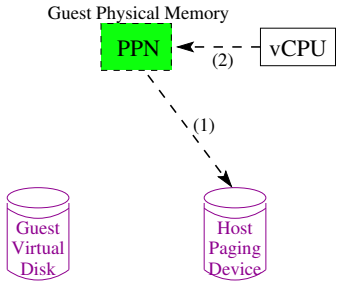
Double-Paging

Hypervisor Swapping and Guest I/O: Sub-Optimal Behavior



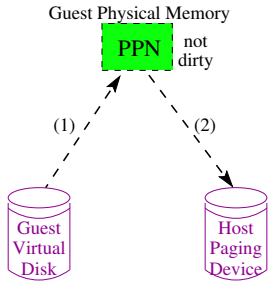
Host swapout followed by guest disk read

(stale swap reads)



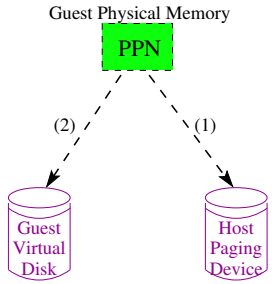
Host swapout followed by guest overwriting the entire page

(false swap reads)



Host swap out of an unmodified guest page

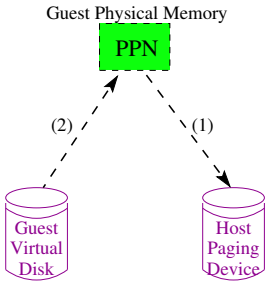
(slient swap writes)



Host swapout followed by guest disk write

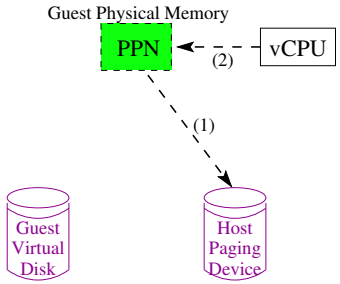
- VSwapper (Amit et al., ASPLOS'14) describes some more cases

Hypervisor Swapping and Guest I/O: Sub-Optimal Behavior



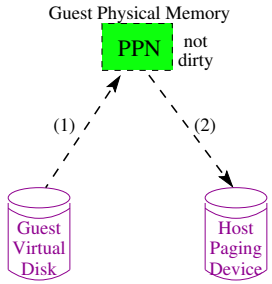
Host swapout followed by guest disk read

(stale swap reads)



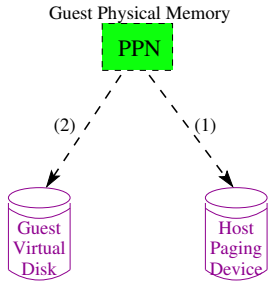
Host swapout followed by guest overwriting the entire page

(false swap reads)



Host swap out of an unmodified guest page

(slient swap writes)



Host swapout followed by guest disk write

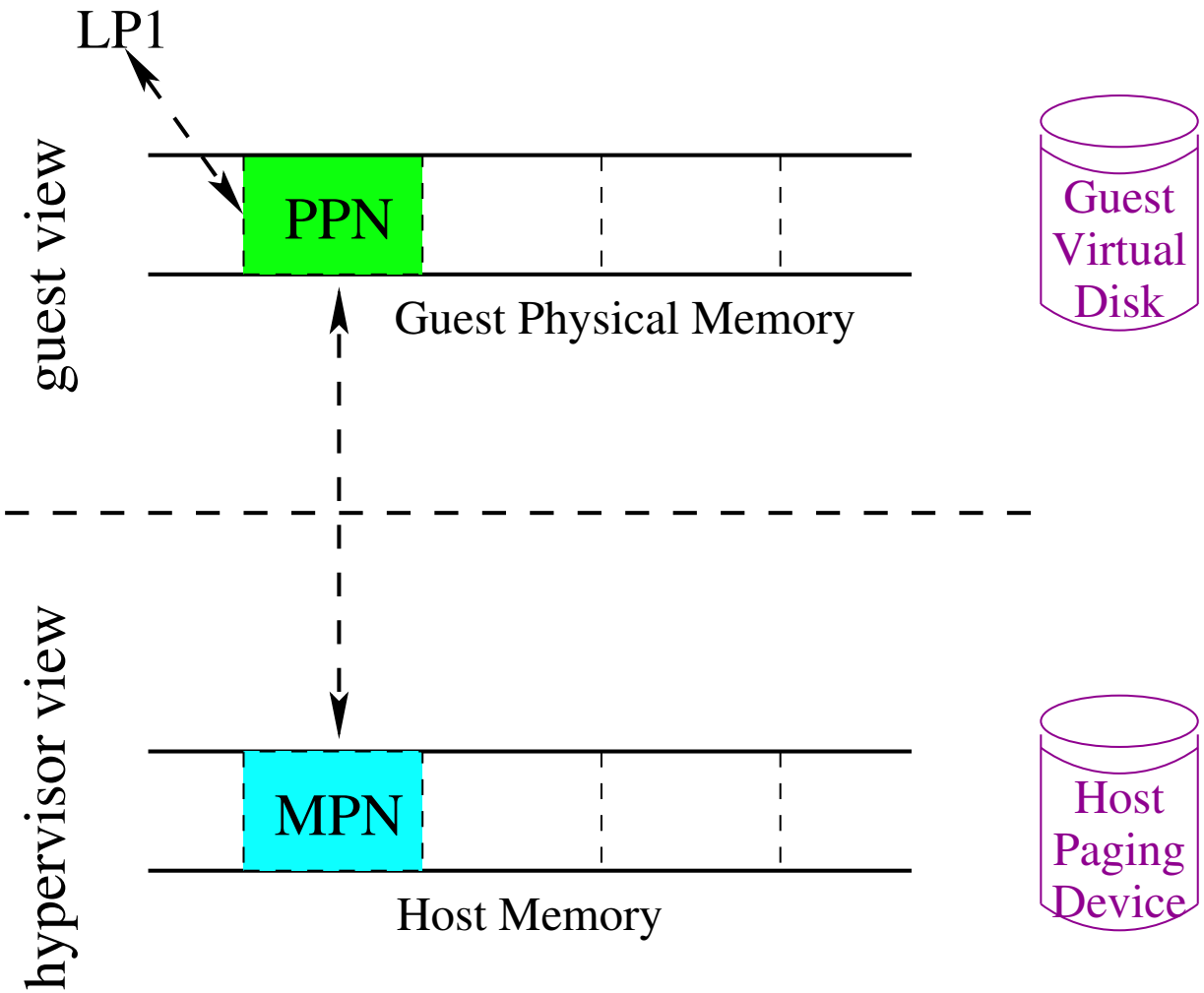
Double-Paging

- VSwapper (Amit et al., ASPLOS'14) describes some more cases
- Tesseract focuses on the *Double-Paging* problem
  - our baseline has solutions for stale-swap-reads and false-swap-reads

The Double Paging Problem

- Goldberg/Hassinger 1974; Seawright/MacKinnon 1979
- Memory overcommitment at both host level and guest level
- Guest paging of page swapped out by hypervisor
- Swap in contents only to write it out again
- Existing solutions:
  - Specialized guest paging device (Govil et al. 1999)
  - Paravirtualization (Geiger: Jones 2002; Lu and Shen 2007)

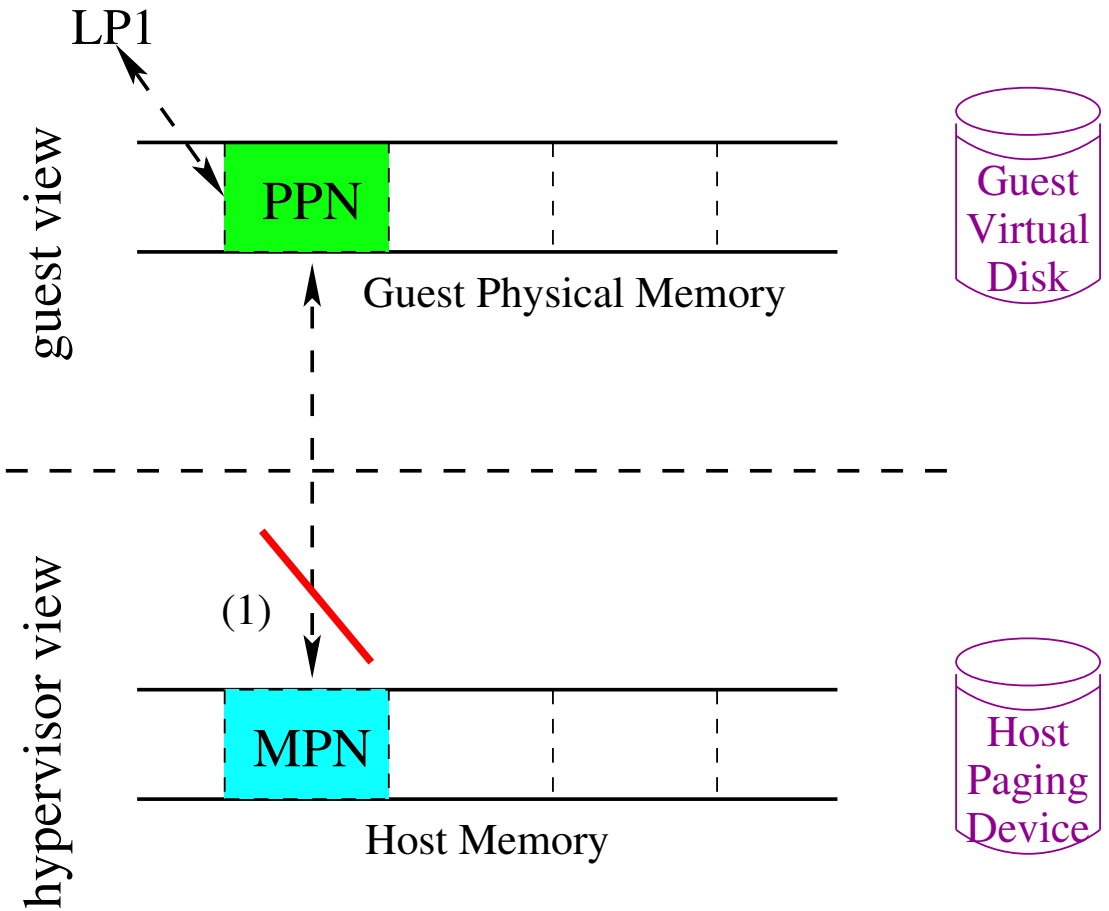
Two Levels of Memory Scheduling



LP: (guest) Logical Page; PPN : Physical Page Number; MPN : Machine Page Number;

Host Level Swap Out

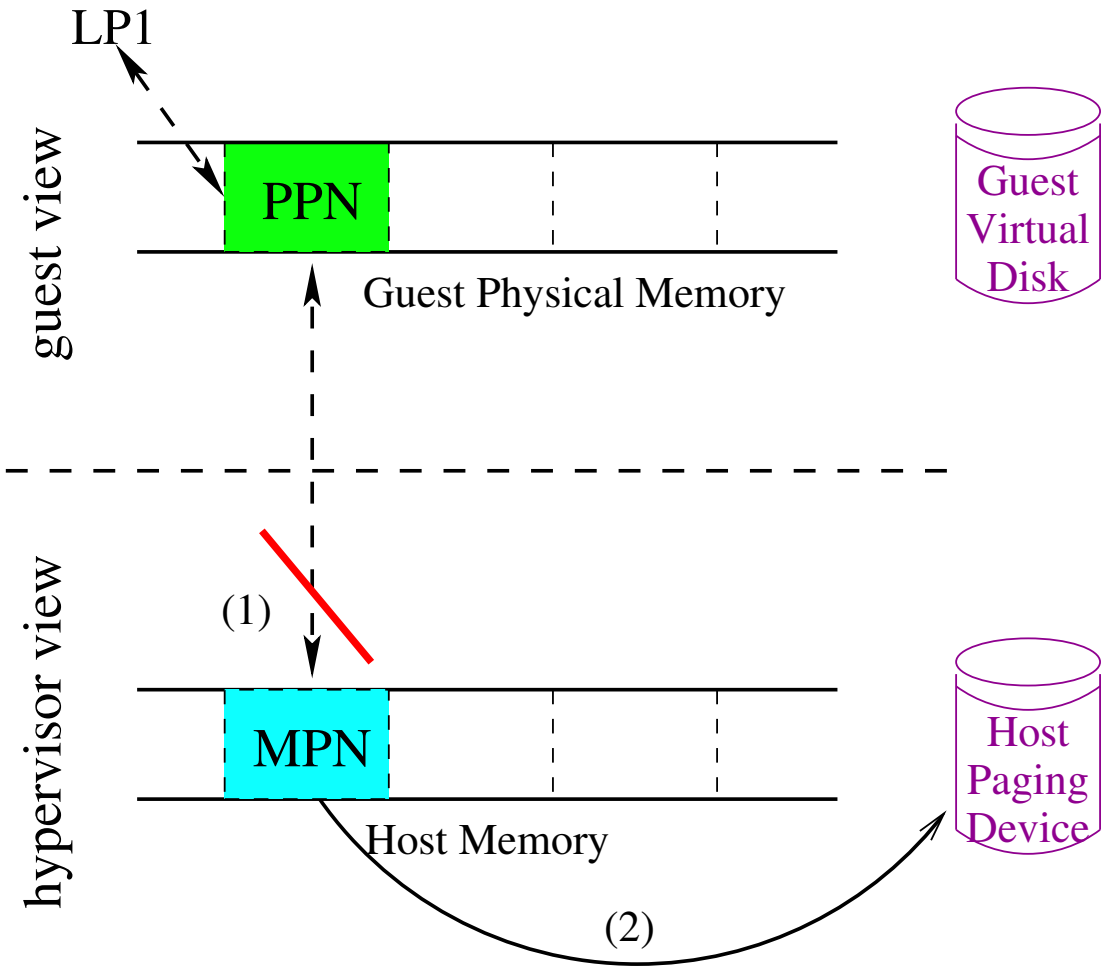
(1) Remove PPN to MPN mapping



Host Level Swap Out

(1) Remove PPN to MPN mapping

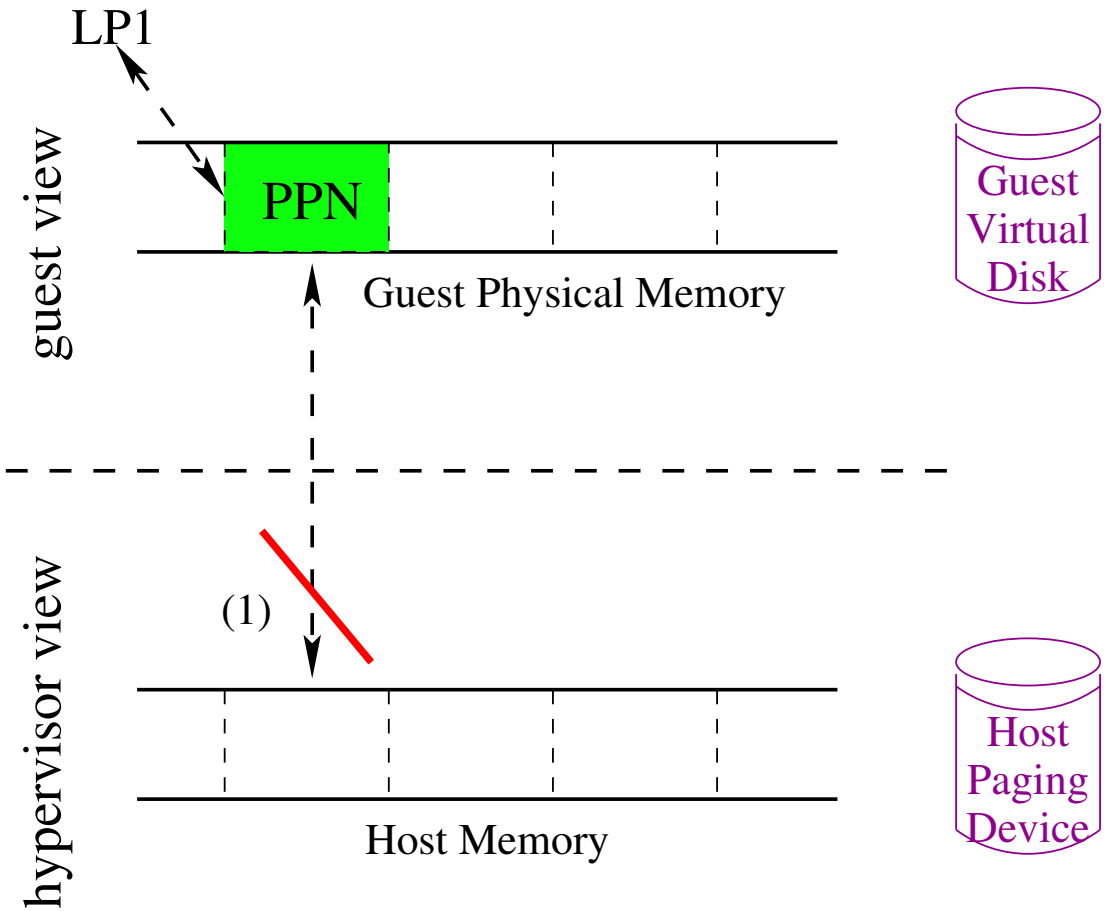
(2) Write MPN to host paging device;



Host Level Swap Out

(1) Remove PPN to MPN mapping

(2) Write MPN to host paging device; reuse MPN

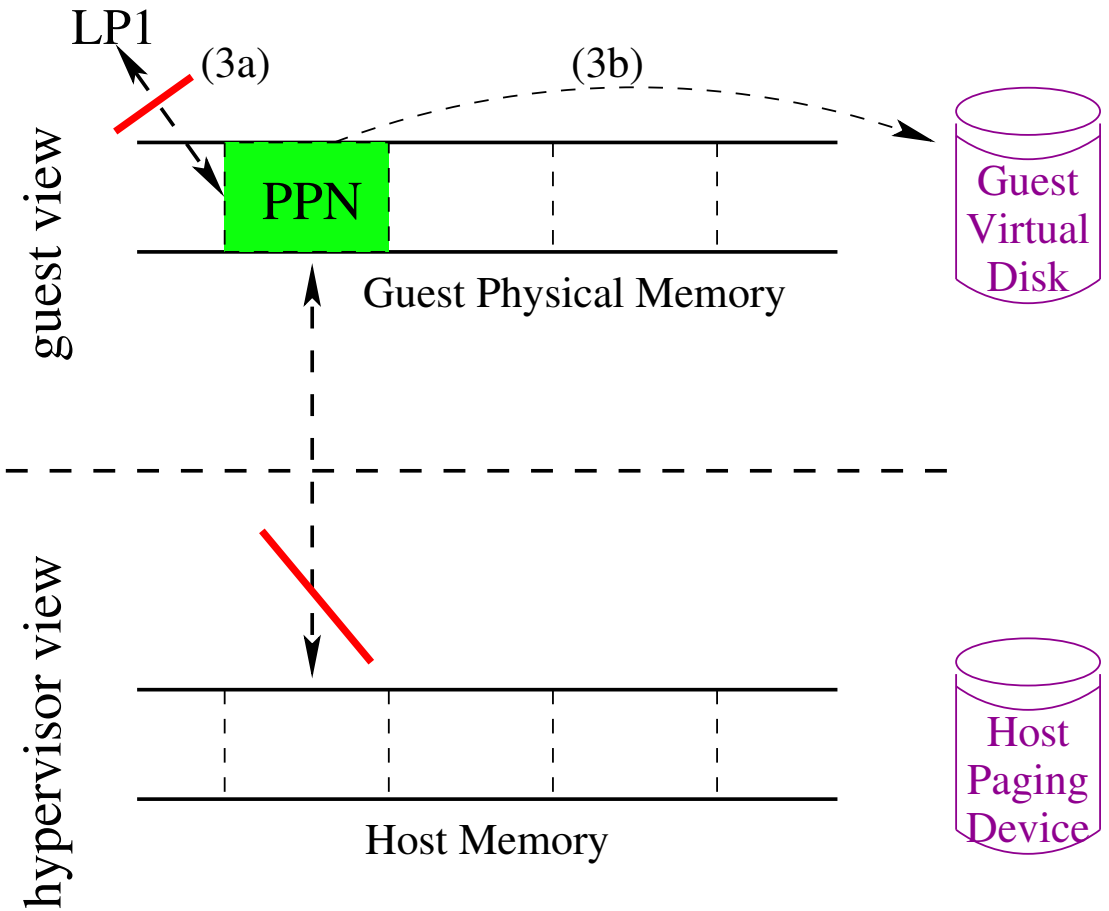


Guest Level Swap Out: Resulting in a Double Paging

(1) Remove PPN to MPN mapping

(2) Write MPN to host paging device; reuse MPN

(3) Guest block write request



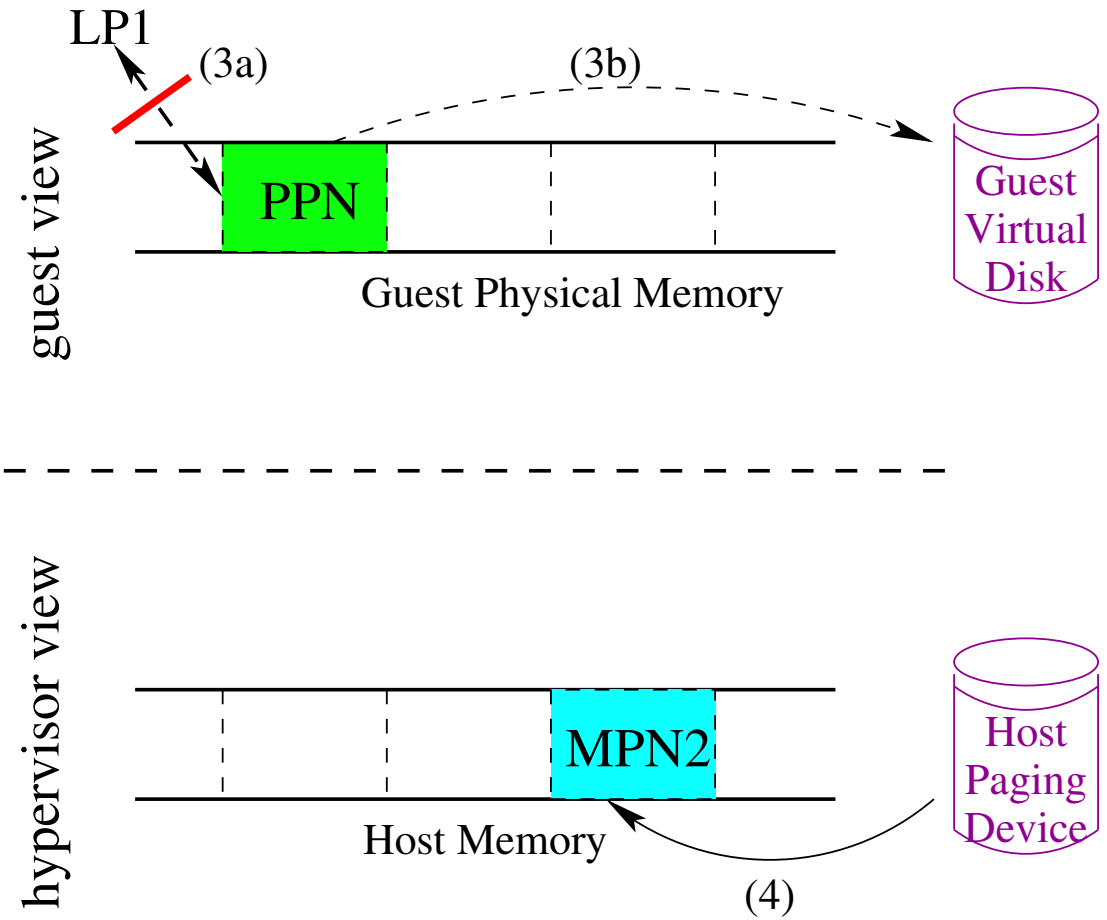
Guest Level Swap Out: Resulting in a Double Paging

(1) Remove PPN to MPN mapping

(2) Write MPN to host paging device; reuse MPN

(3) Guest block write request

(4) Memory allocation and swap in



Guest Level Swap Out: Resulting in a Double Paging

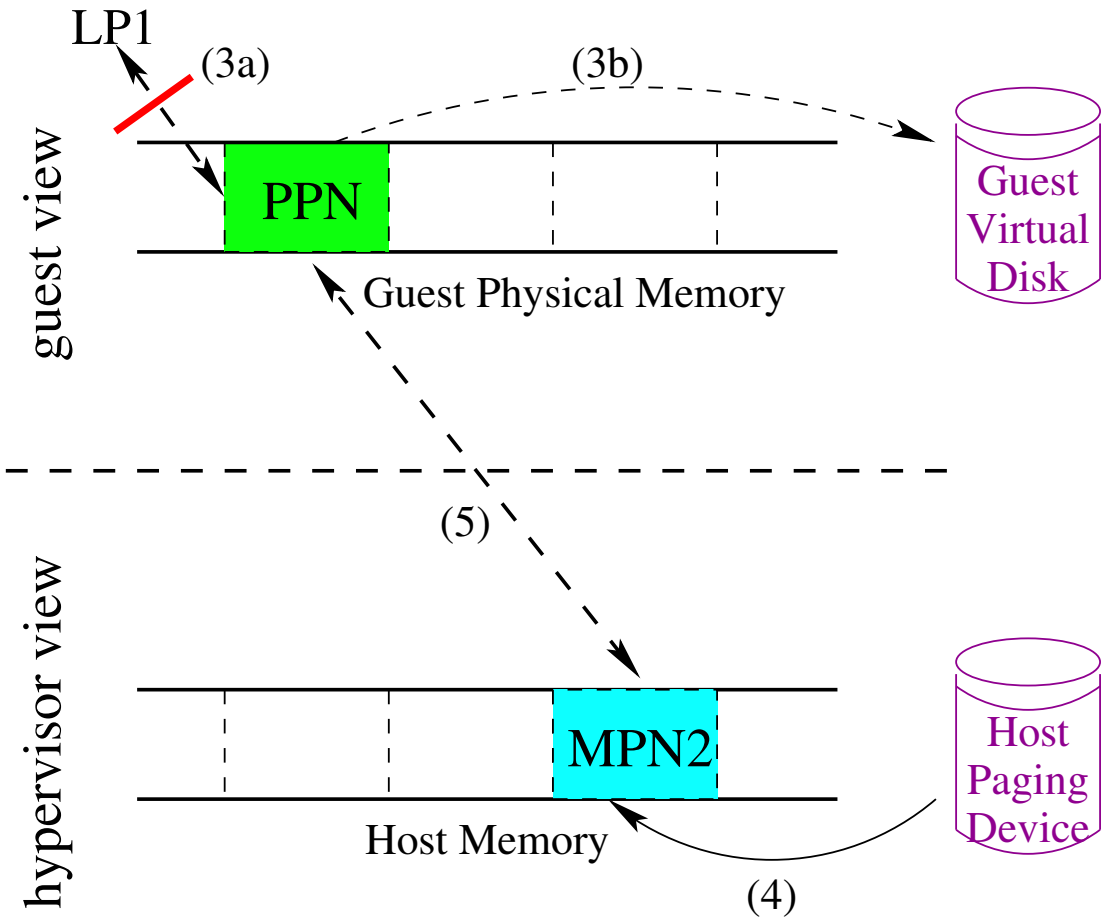
(1) Remove PPN to MPN mapping

(2) Write MPN to host paging device; reuse MPN

(3) Guest block write request

(4) Memory allocation and swap in

(5) Establish PPN to MPN mapping



Guest Level Swap Out: Resulting in a Double Paging

(1) Remove PPN to MPN mapping

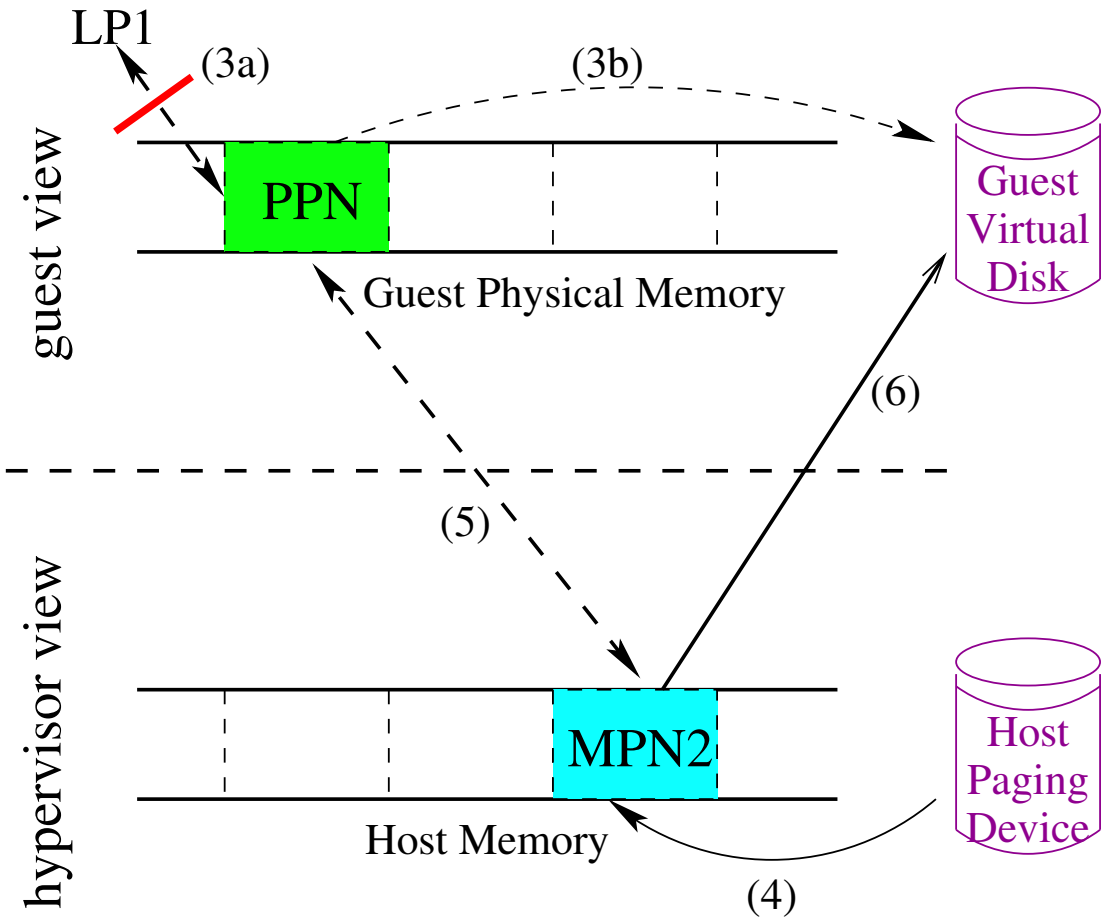
(2) Write MPN to host paging device; reuse MPN

(3) Guest block write request

(4) Memory allocation and swap in

(5) Establish PPN to MPN mapping

(6) Write block to guest disk



Guest Level Swap Out: Resulting in a Double Paging

(1) Remove PPN to MPN mapping

(2) Write MPN to host paging device; reuse MPN

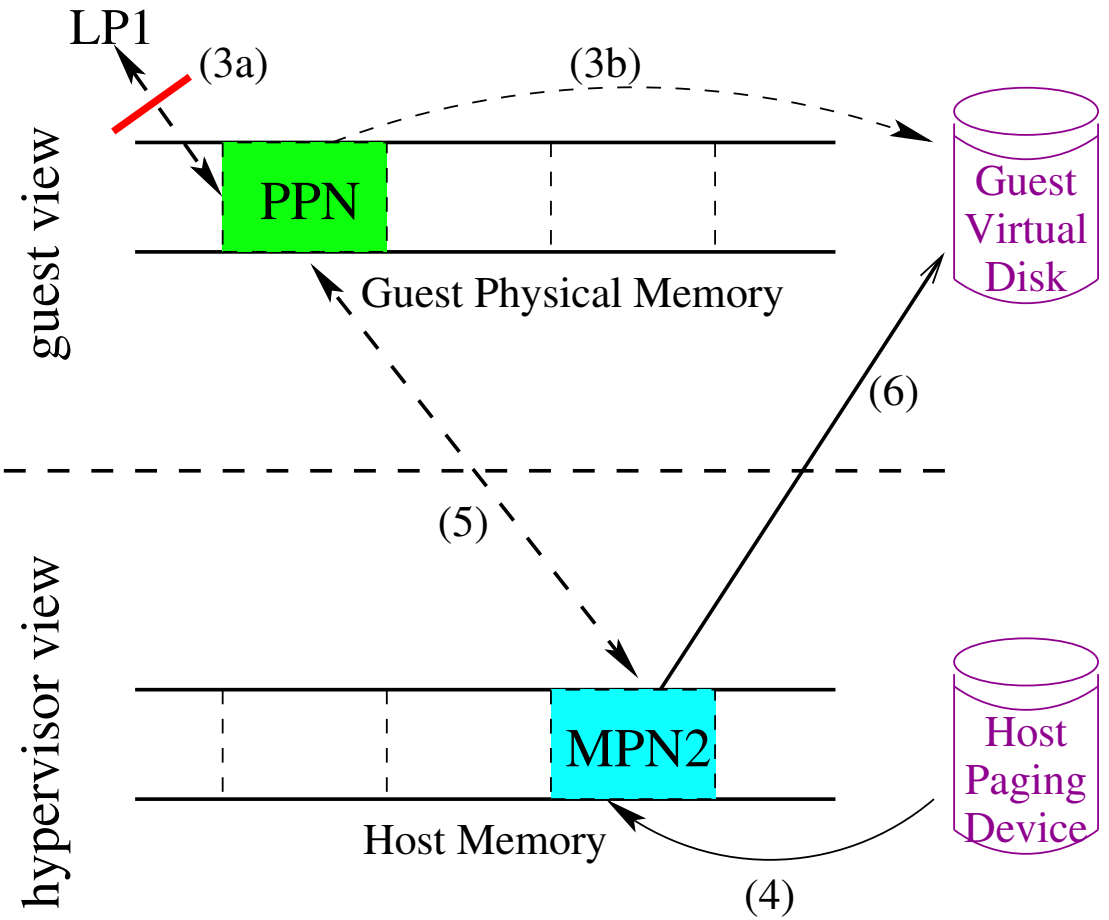
(3) Guest block write request

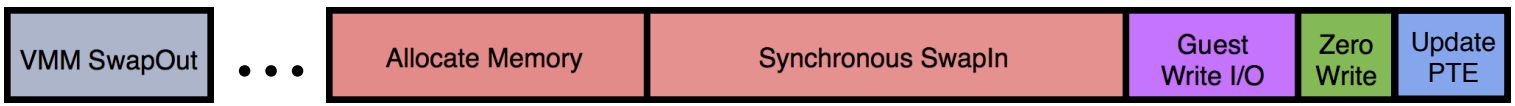
(4) Memory allocation and swap in

(5) Establish PPN to MPN mapping

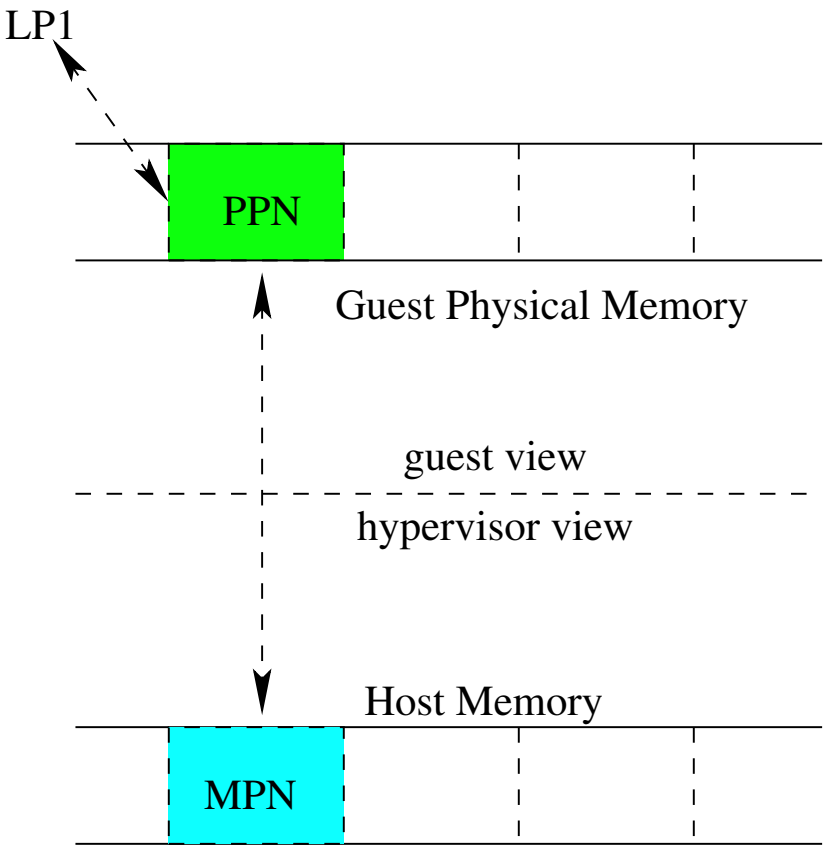
(6) Write block to guest disk

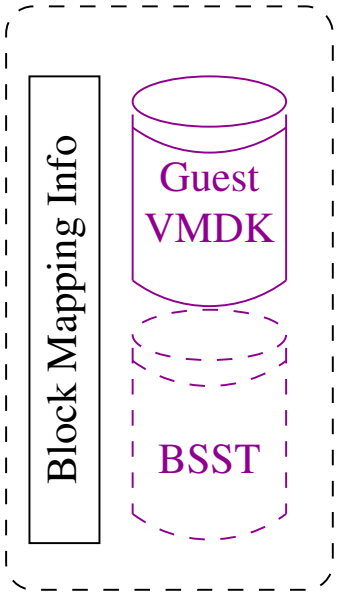
(7) Zero the new MPN for reuse



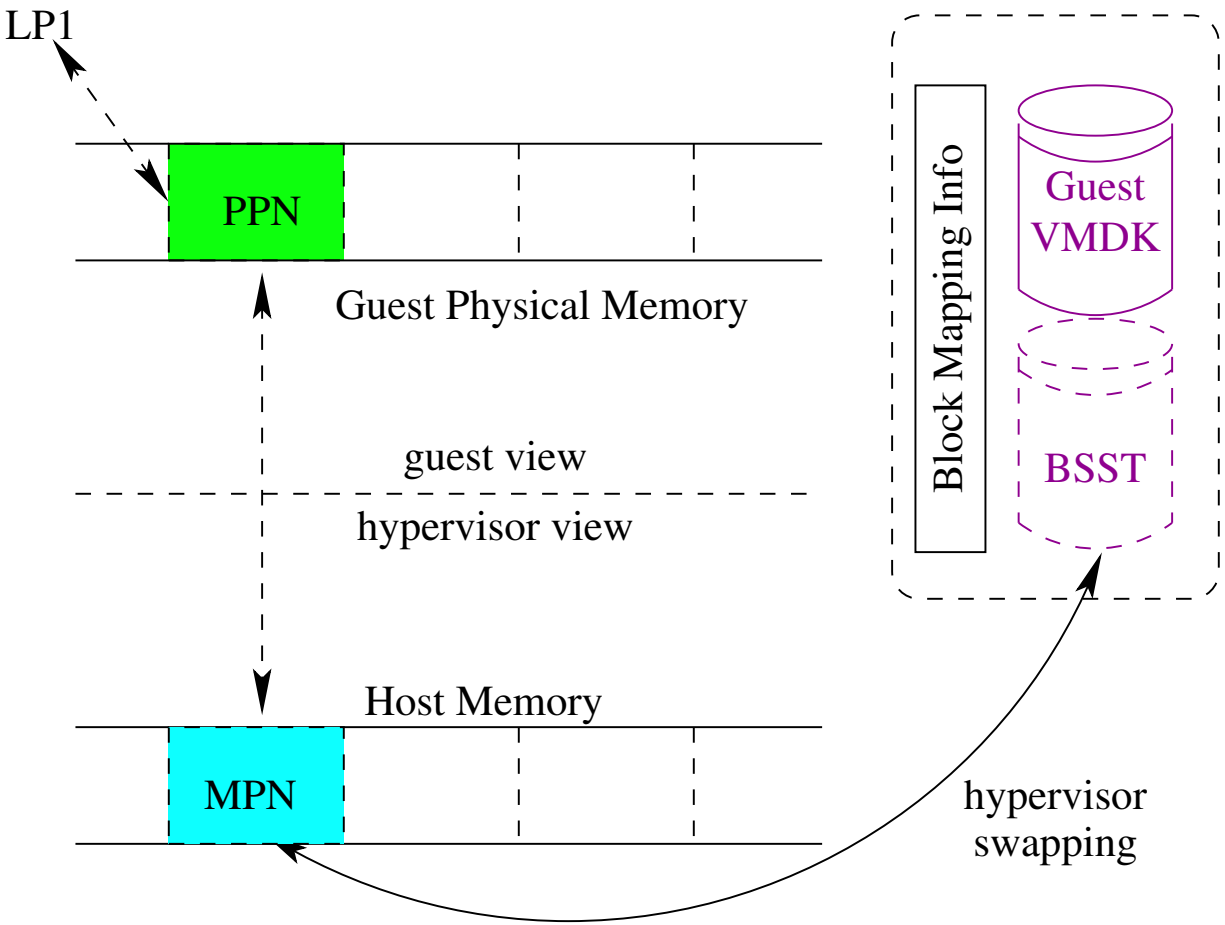


Solution: Block-Swap Store (BSST)





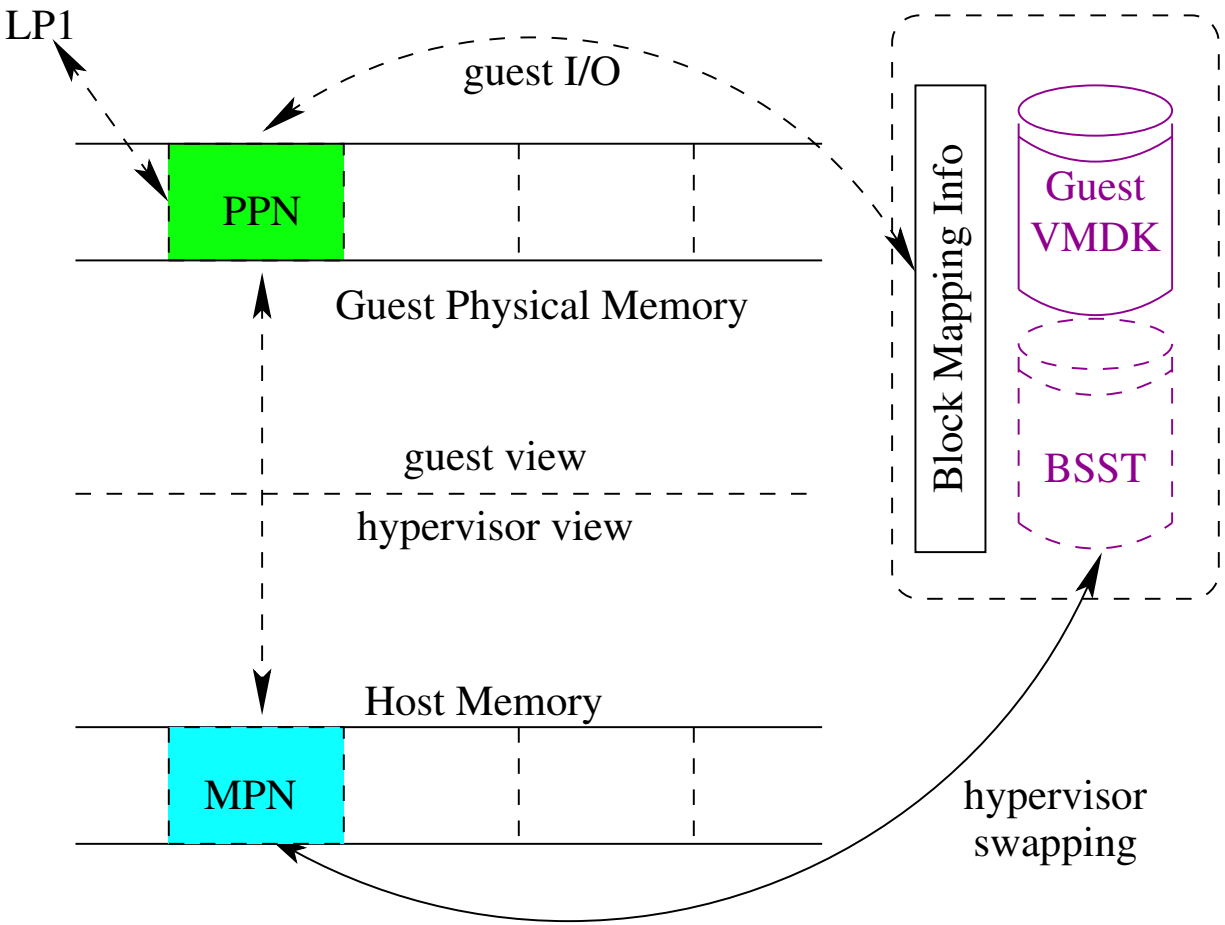
Solution: Block-Swap Store (BSST)



PPN to BSST mapping

PPN	BSST
...	...
< *MPN* 1 >	< *bBlock*1 >
...	...

Solution: Block-Swap Store (BSST)



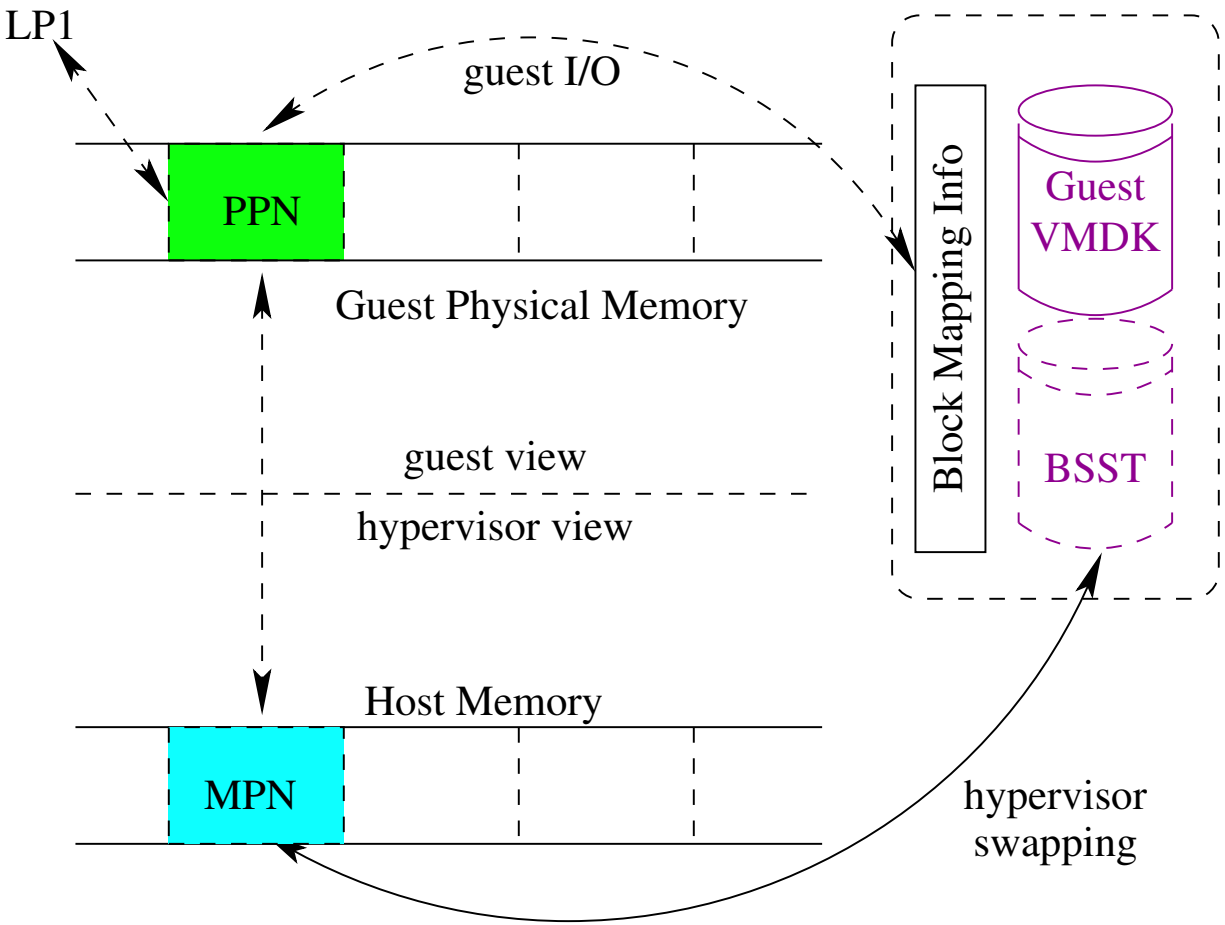
guest VMDK to BSST indirection

guest block	BSST
...	...
< *gBlock*1 >	< *bBlock*1 >
...	...

PPN to BSST mapping

PPN	BSST
...	...
< *MPN* 1 >	< *bBlock*1 >
...	...

Solution: Block-Swap Store (BSST)

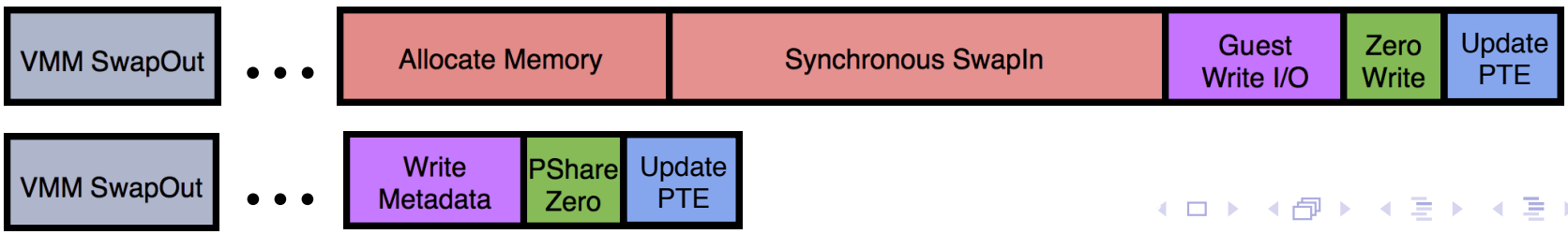


guest VMDK to BSST indirection

guest block	BSST
...	...
< *gBlock*1 >	< *bBlock*1 >
...	...

PPN to BSST mapping

PPN	BSST
...	...
< *MPN* 1 >	< *bBlock*1 >
...	...



Tesseract: Under the Hood

- Explicit swapper and block-swap store VMDK (BSST)
- Track associations between memory pages and disk blocks
- Allow indirections between guest VMDKs and BSST
- Defragment guest VMDKs

Select Cold Pages for Swapping

- The existing hypervisor swapper avoids picking pages the guest OS is likely to page out
  - this is done through random selection

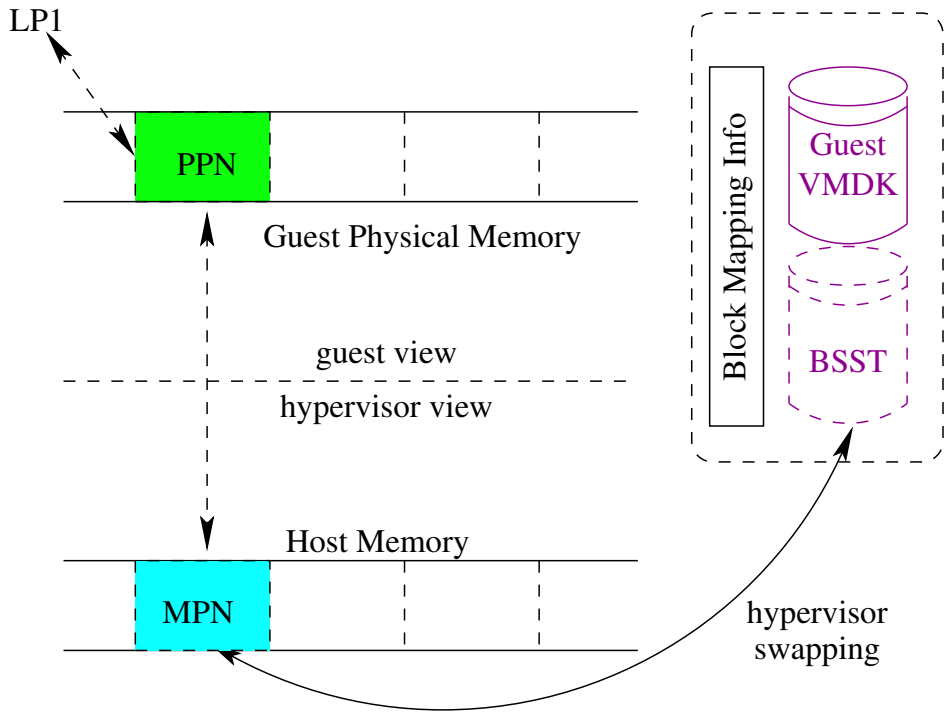
Select Cold Pages for Swapping

- The existing hypervisor swapper avoids picking pages the guest OS is likely to page out
  - this is done through random selection

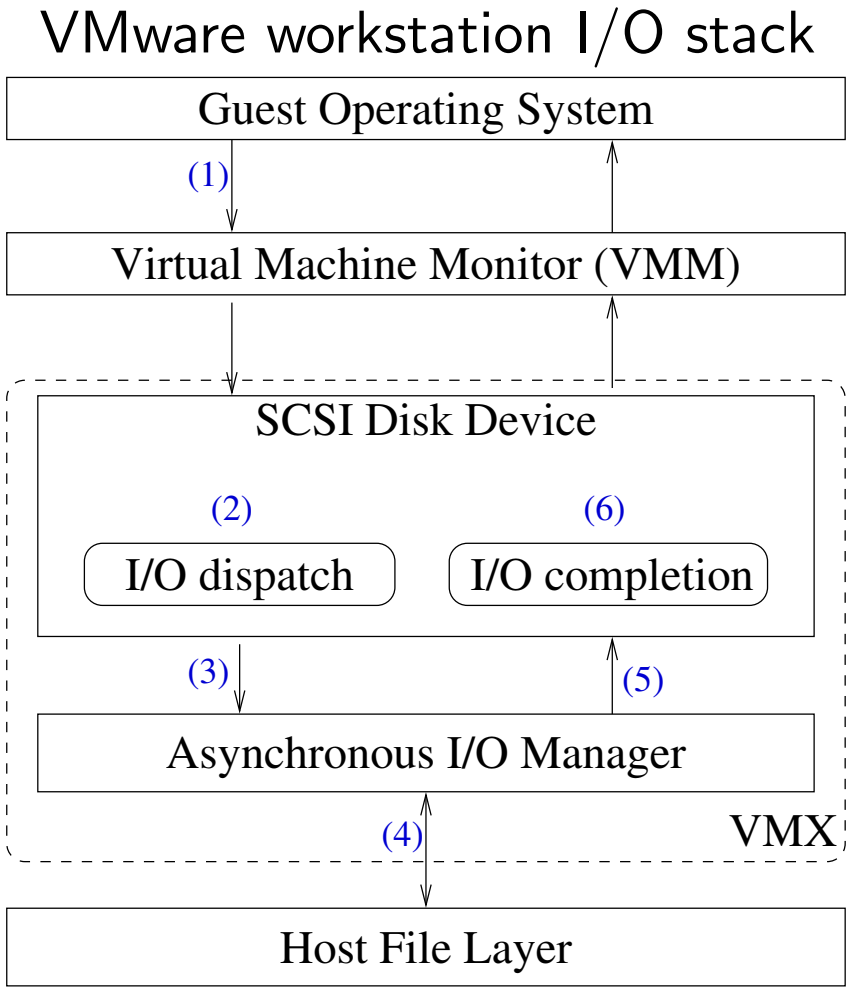
- Tesseract doesn't suffer from double-paging
  - it is good if the guest and hypervisor both page out the same set of pages
  - swap out coldest pages
  - uses random page selection with victim cache

Explicit Page Swapper

- Choose cold(er) pages for swapping
- Create PPN to BSST mappings on swap out
- Intercept writes to memory to update mappings
  - guest disk reads
  - zeroing of page
  - cancel swap-in if the entire page is being overwritten



Guest VMDK to BSST Mappings: Guest Write Request

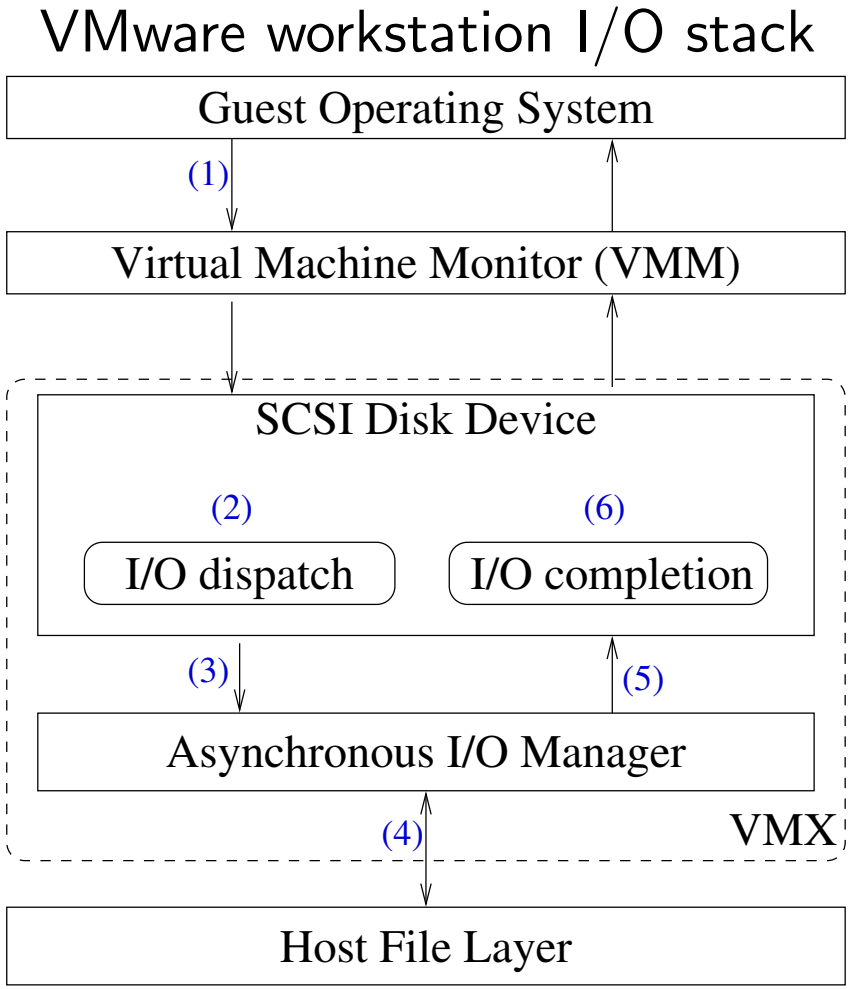


Guest VMDK to BSST Mappings: Guest Write Request

Write request issued:

(1) Original scatter-gather list

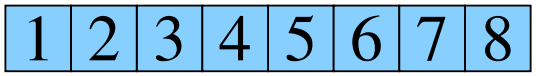
1	2	3	4	5	6	7	8



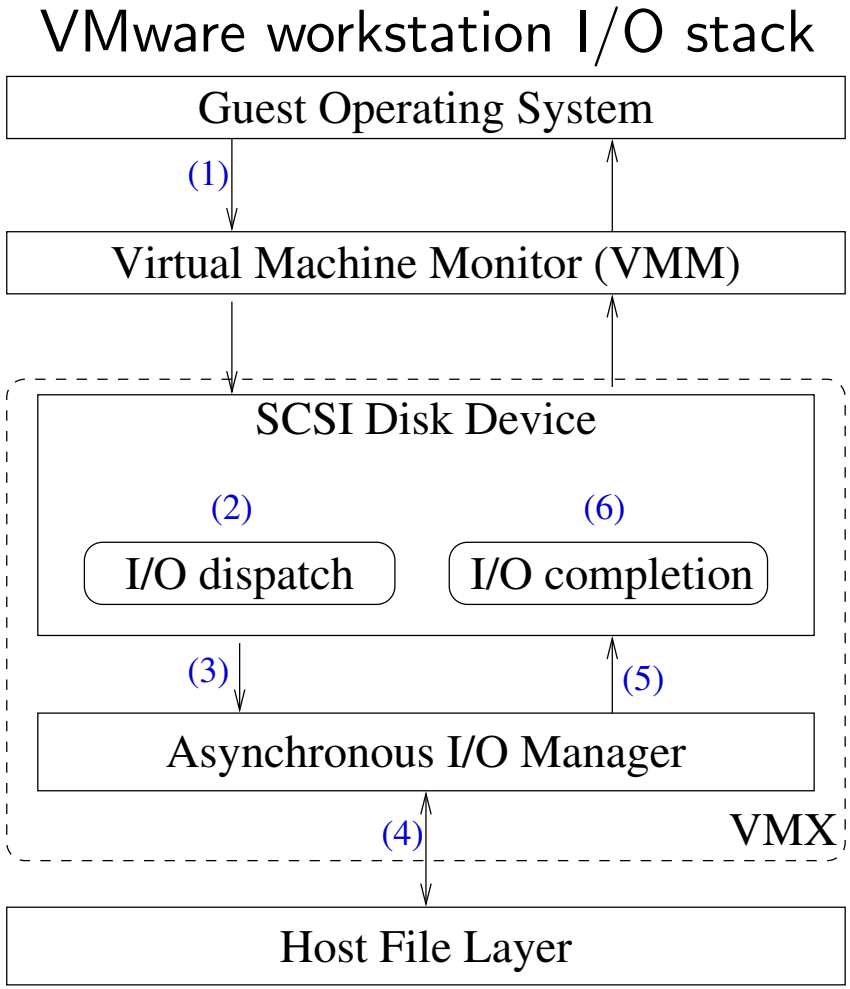
Guest VMDK to BSST Mappings: Guest Write Request

Write request issued:

(1) Original scatter-gather list



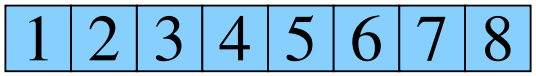
(2) Tesseract intercepts guest write requests and finds swapped pages



Guest VMDK to BSST Mappings: Guest Write Request

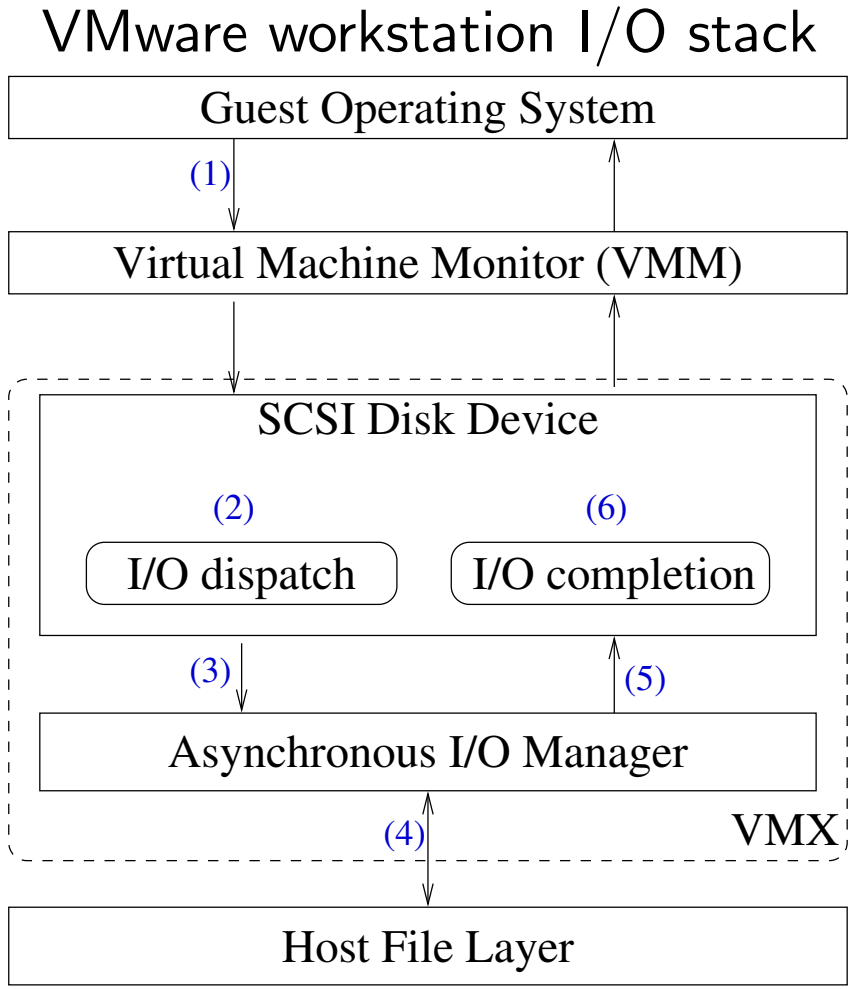
Write request issued:

(1) Original scatter-gather list



(2) Tesseract intercepts guest write requests and finds swapped pages

(3) Rewrite scatter-gather lists based on indirections to avoid double-paging



Guest VMDK to BSST Mappings: Guest Write Request

Write request issued:

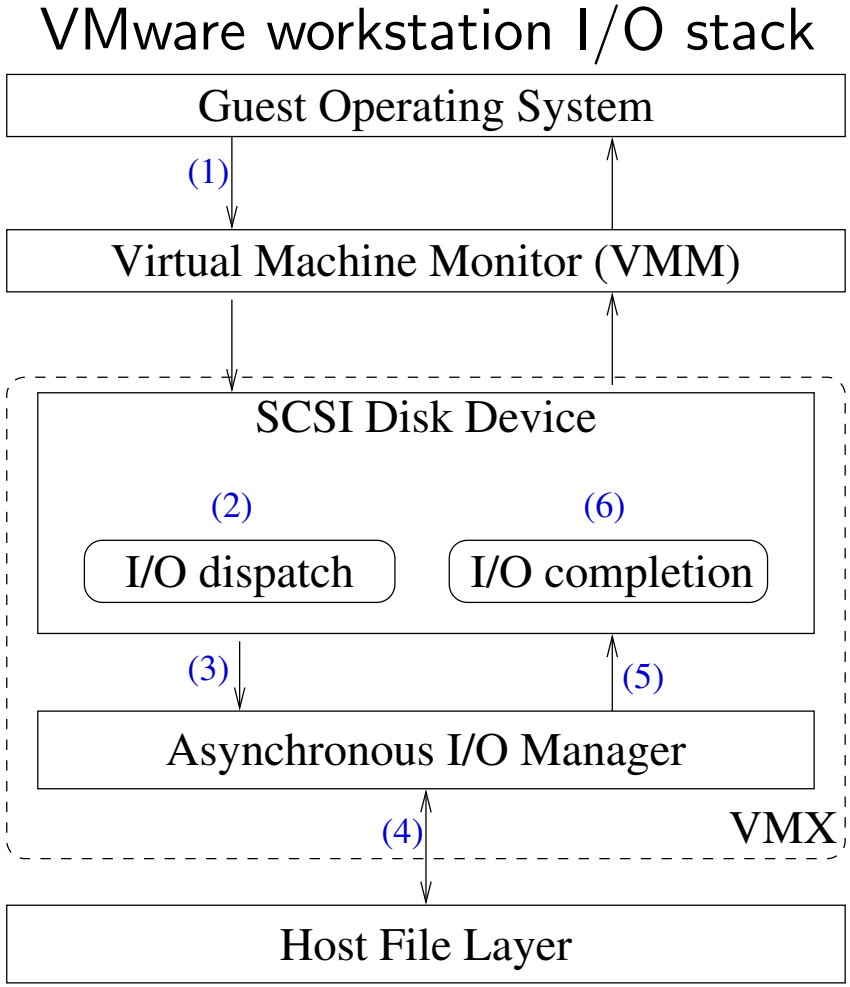
(1) Original scatter-gather list

(2) Tesseract intercepts guest write requests and finds swapped pages

(3) Rewrite scatter-gather lists based on indirections to avoid double-paging

Write completion phase:

(6) Establish guest VMDK to BSST mappings

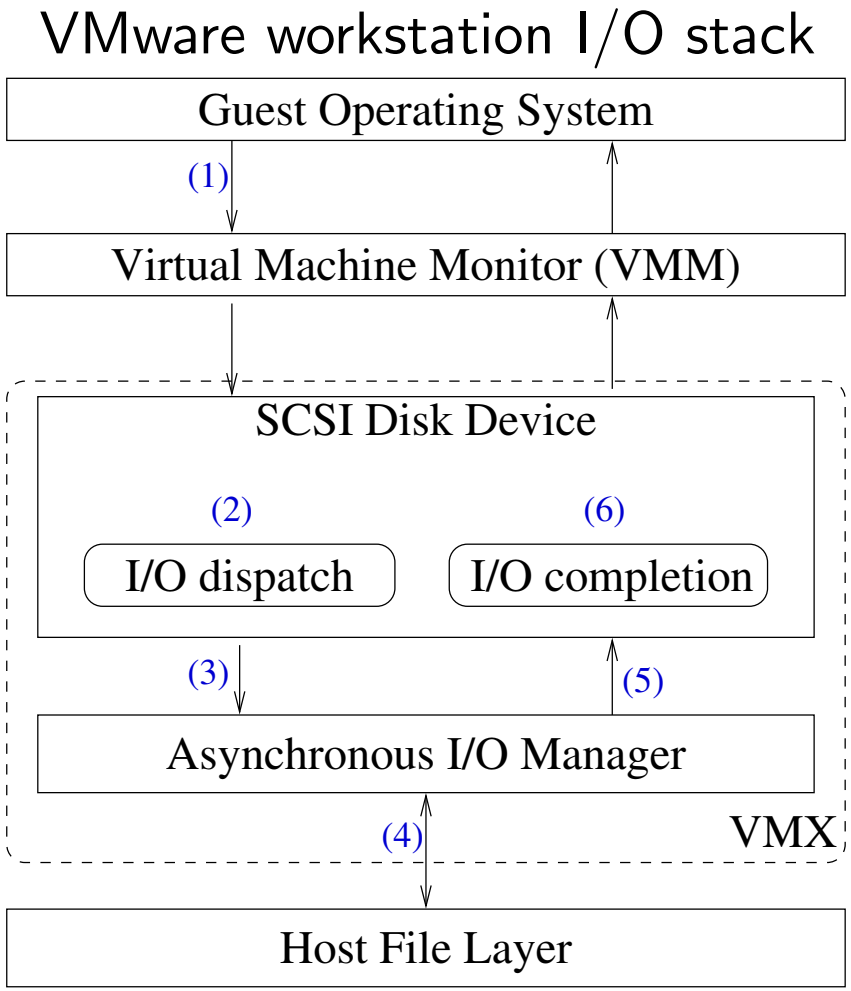


Guest VMDK to BSST Mappings: Guest Read Request

Read request issued:

(1) Original scatter-gather list

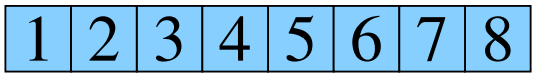
1	2	3	4	5	6	7	8



Guest VMDK to BSST Mappings: Guest Read Request

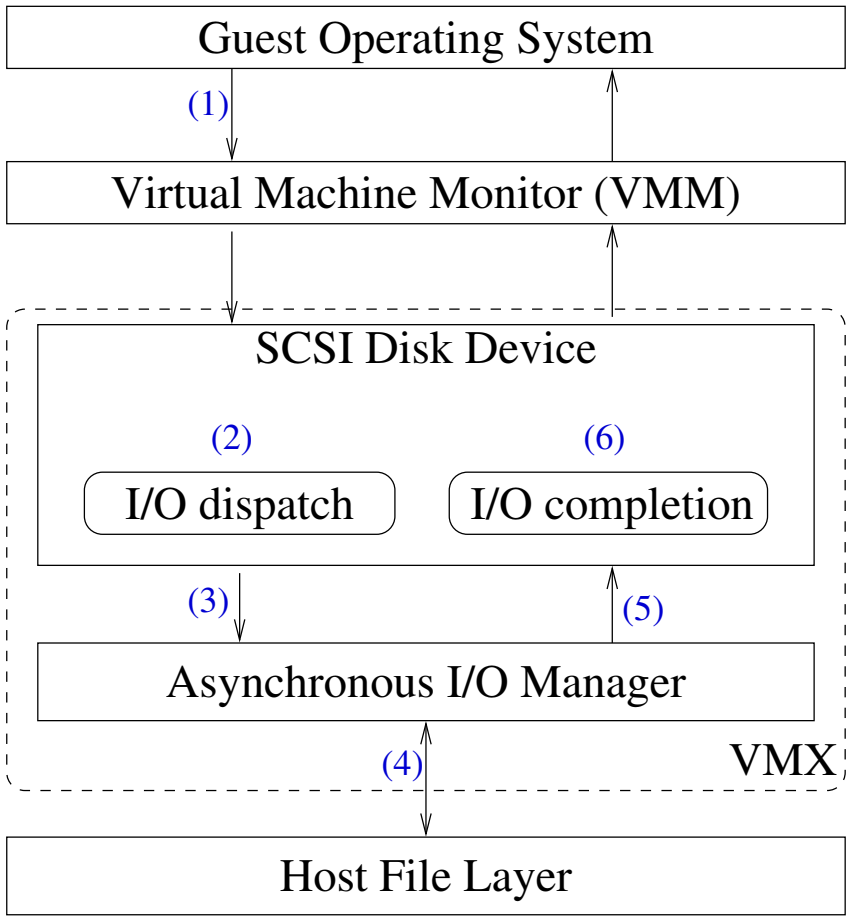
Read request issued:

(1) Original scatter-gather list



(2) lookup guest VMDK indirections

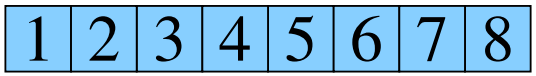
VMware workstation I/O stack



Guest VMDK to BSST Mappings: Guest Read Request

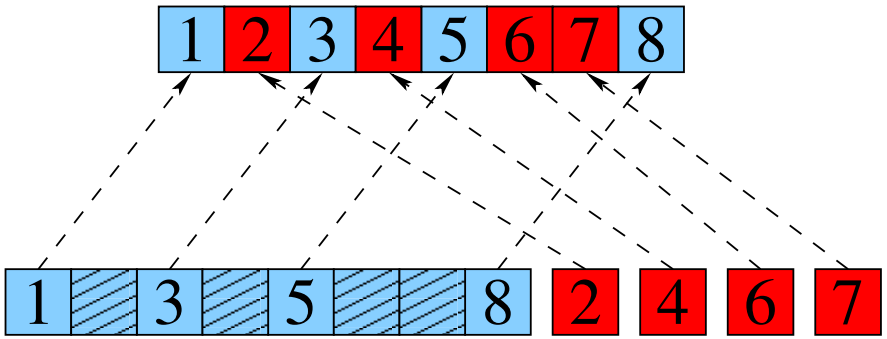
Read request issued:

(1) Original scatter-gather list

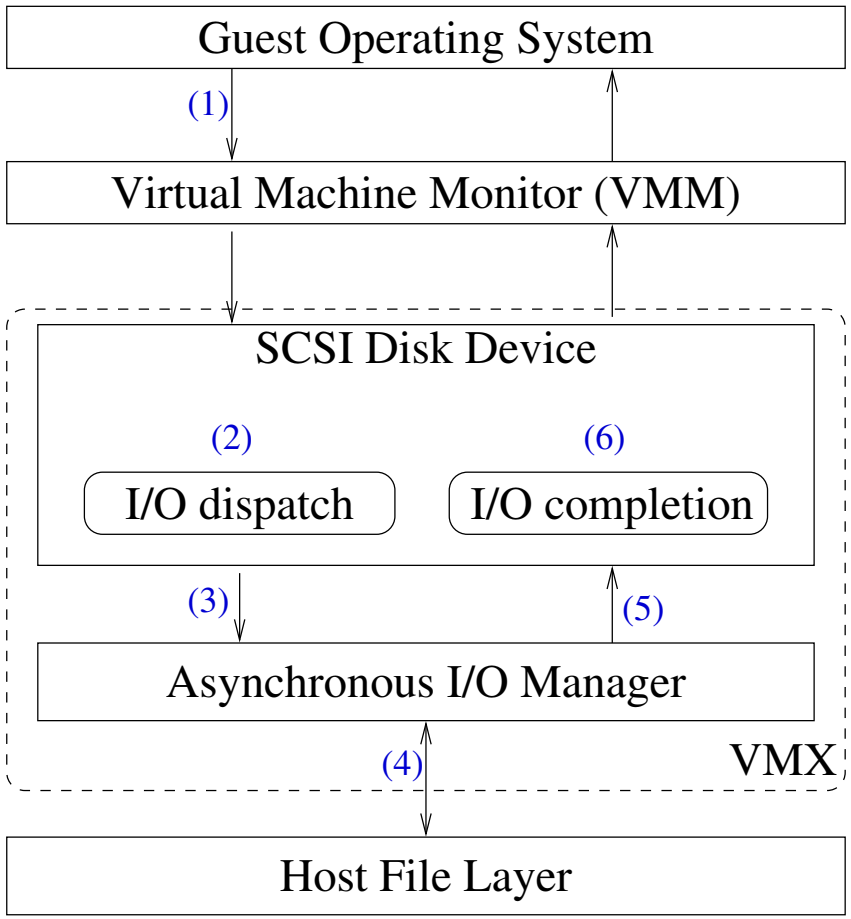


(2) lookup guest VMDK indirections

(3) Split original scatter-gather list



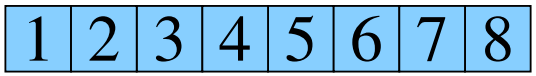
VMware workstation I/O stack



Guest VMDK to BSST Mappings: Guest Read Request

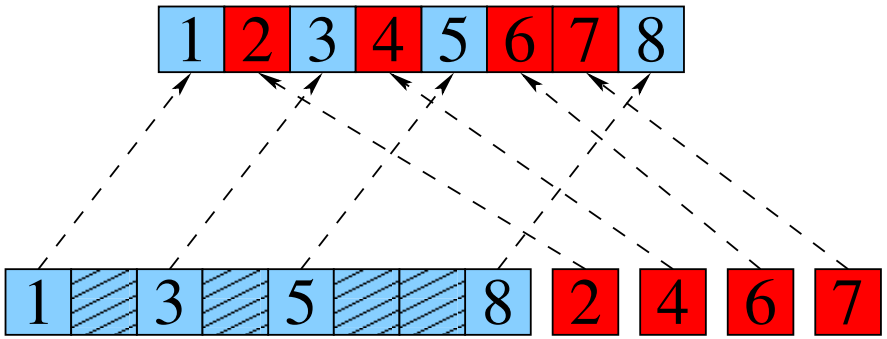
Read request issued:

(1) Original scatter-gather list



(2) lookup guest VMDK indirections

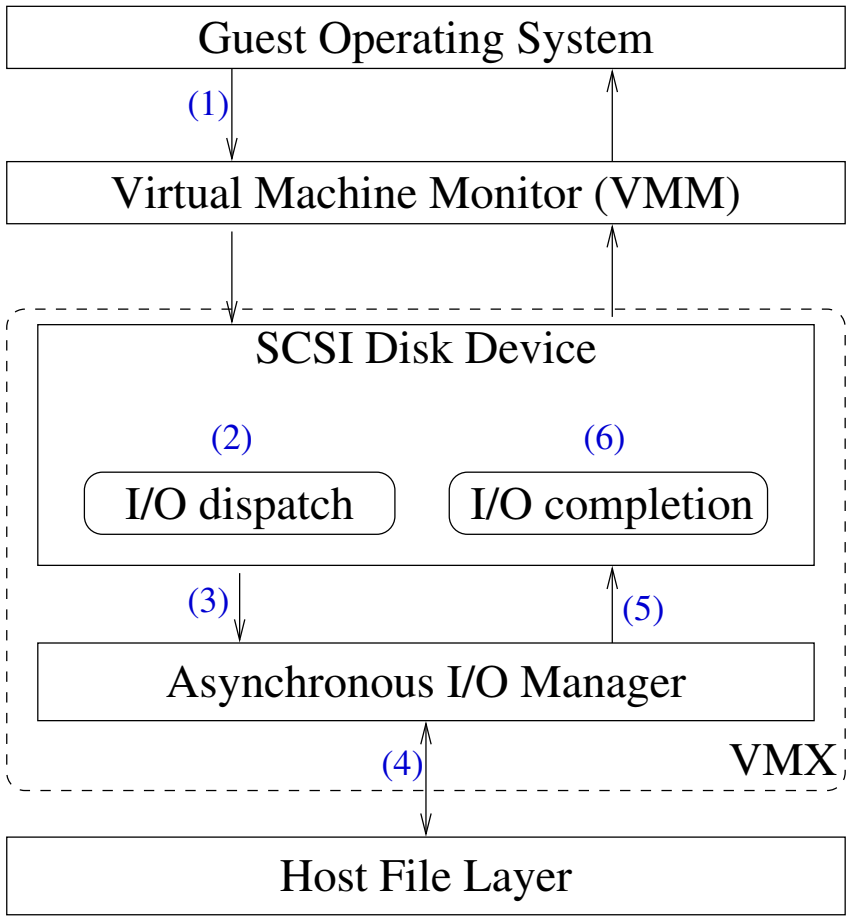
(3) Split original scatter-gather list



Read completion:

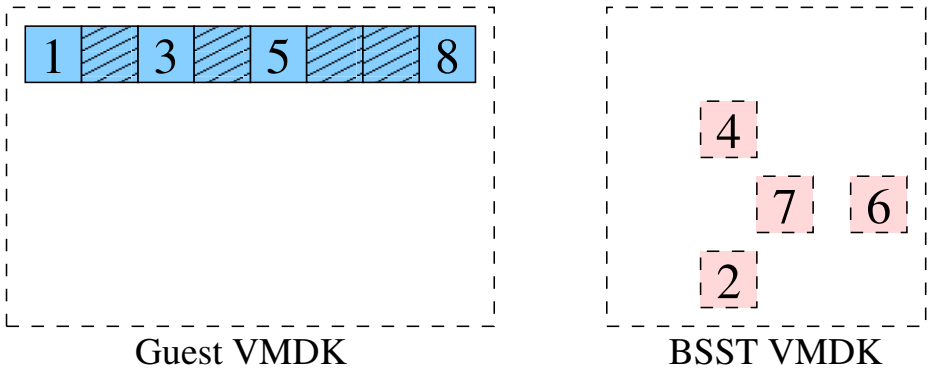
(6) Wait for all completion events and notify the guest.

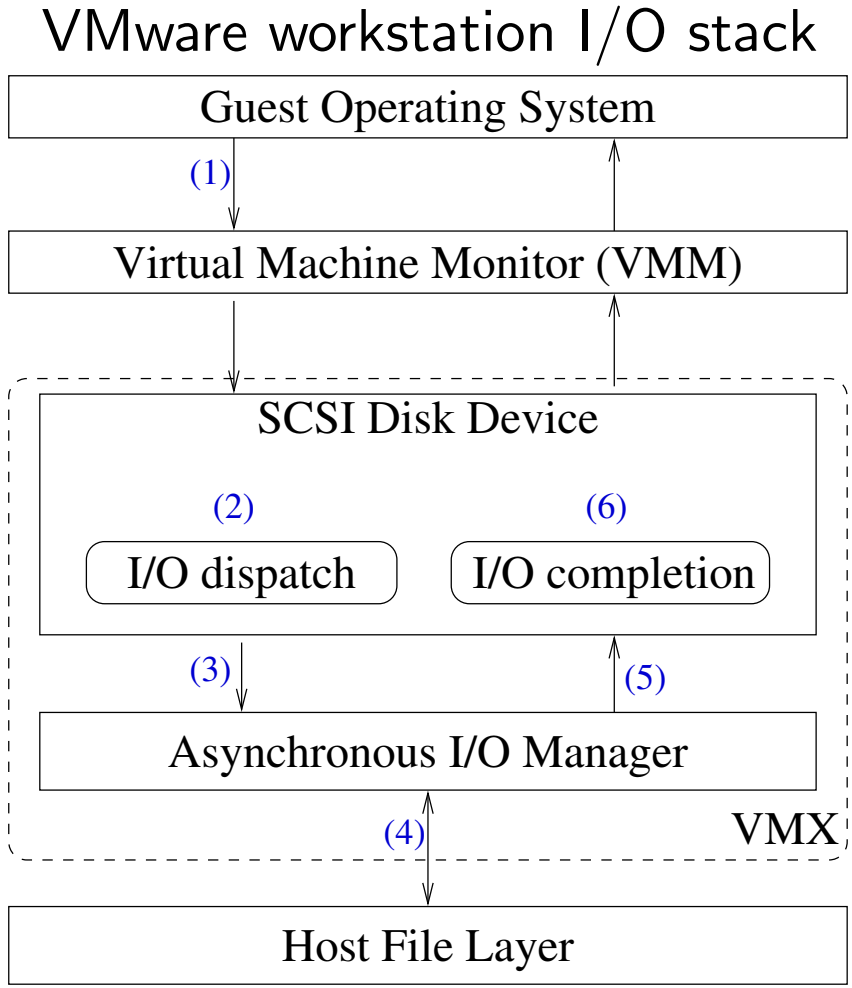
VMware workstation I/O stack



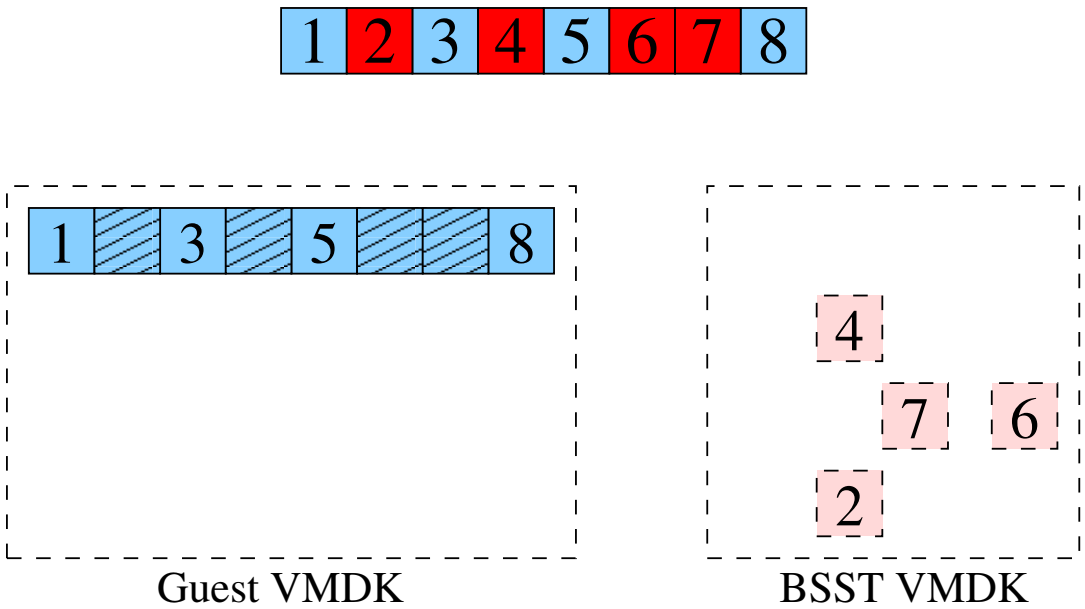
BSST Side Effects – Fragmented Guest VMDKs!

- Trade-off between write-prepare (2) and read-completion (6) times.
- Increased seek times.
- Increased load on disks.



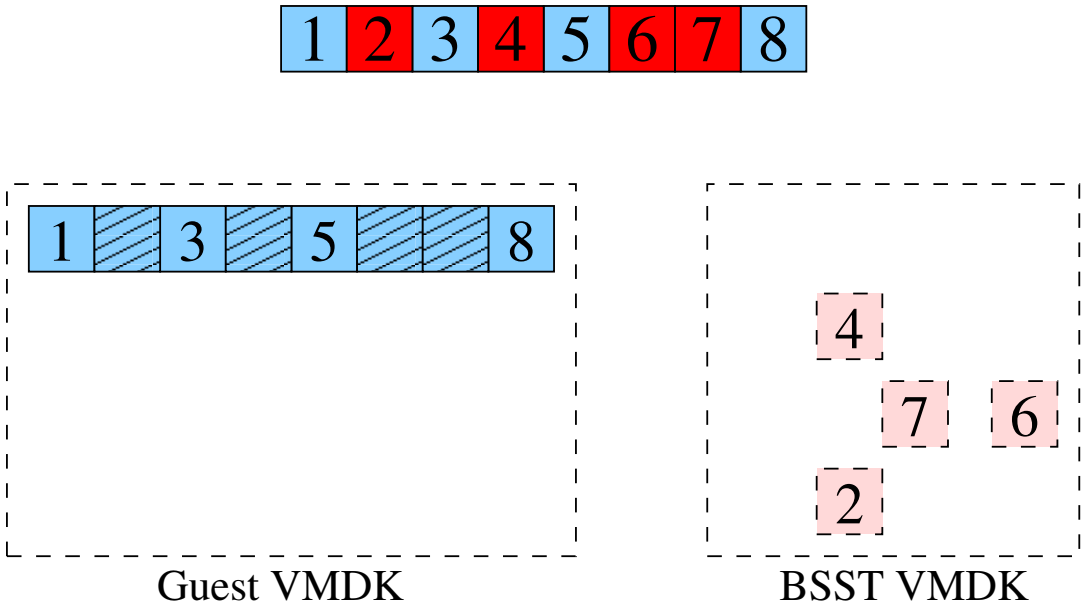


Guest Defragmentation



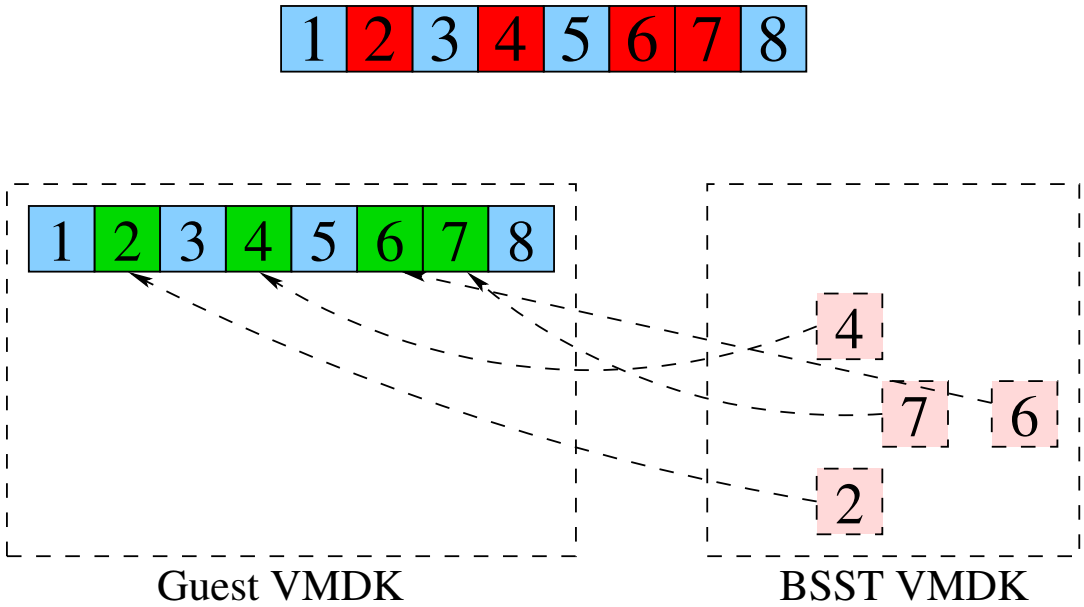
Guest Defragmentation

- Copy blocks from BSST VMDK to guest VMDK in background



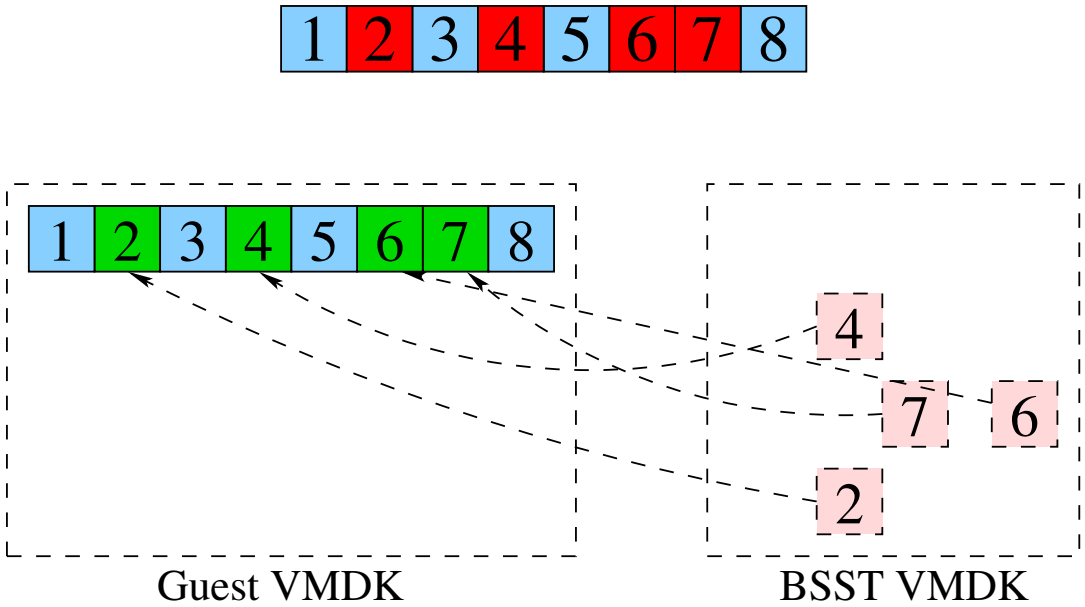
Guest Defragmentation

- Copy blocks from BSST VMDK to guest VMDK in background



Guest Defragmentation

- Copy blocks from BSST VMDK to guest VMDK in background
- Remove guest block to BSST block mapping
- In almost all cases, background defragmentation completes before the corresponding guest requests arrive



Challenges in Evaluating Performance

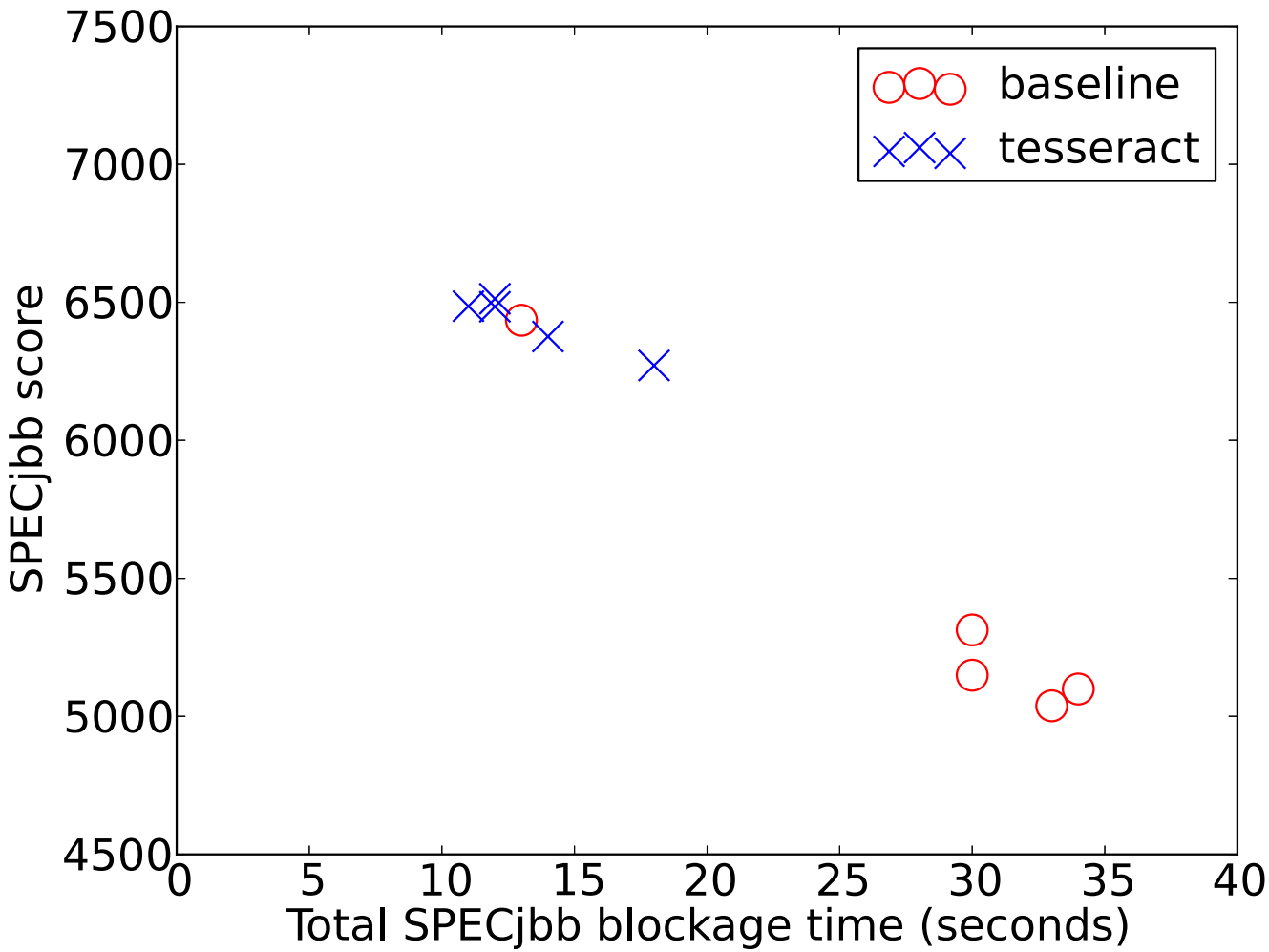
- How to deterministically introduce double-paging activity?
  - hypervisor memory-limit set on VM
  - memhog customized to map/pin pages inside the VM
  - inflation with pinning followed by inflation without pinning
  - first forces hypervisor to swap pages; second induces guest paging
- How to correlate double-paging with application performance?
  - VM responsiveness
  - modified SpecJBB2005 benchmark that emits instantaneous scores
  - application pauses observed in the instantaneous scores emitted by SpecJBB

Experimental Setup

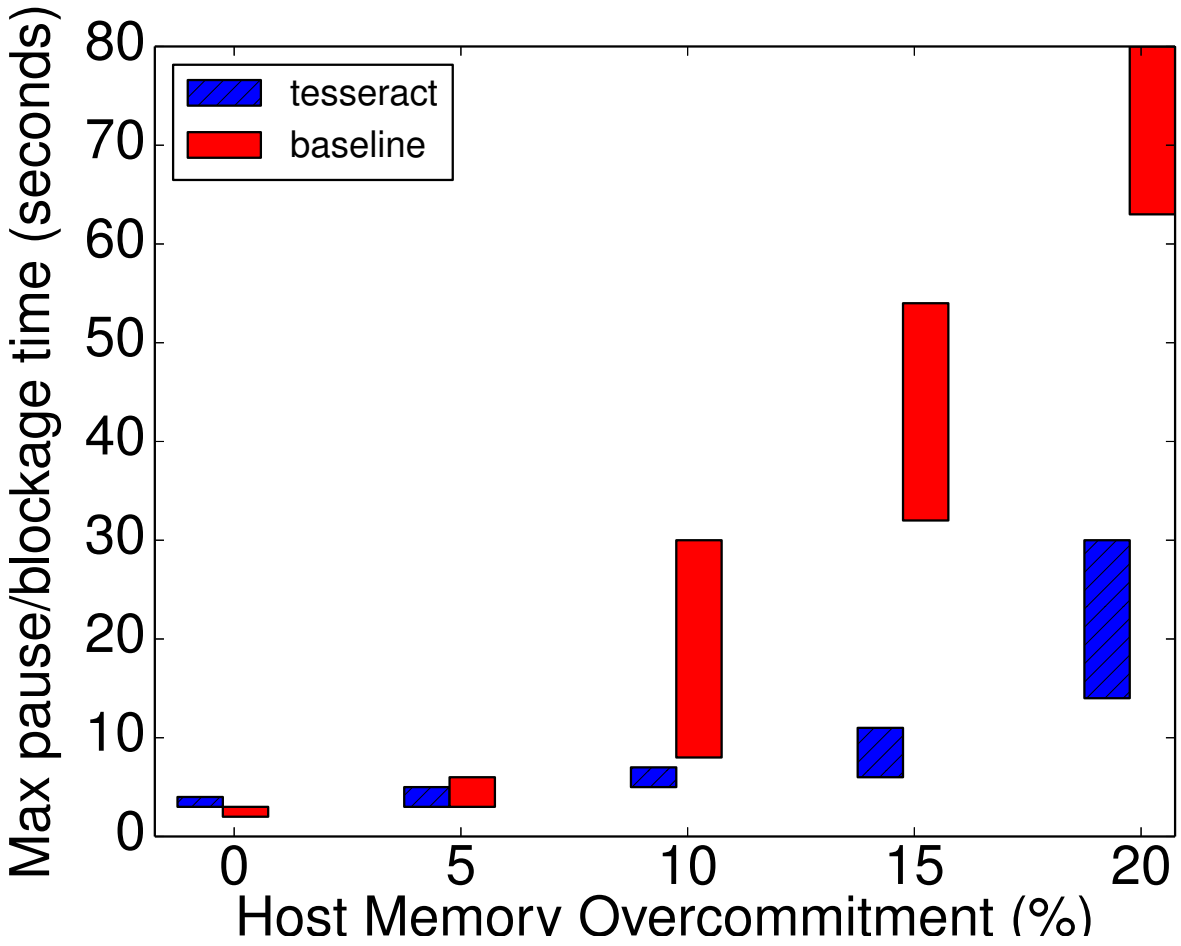
- Host setup
  - AMD Opteron 6168 (Magny-Cours) with 12 1.9 GHz cores
  - 1.5 GB RAM
  - 64-bit OpenSUSE 11.4
- Guest setup:
  - 6 VCPUs
  - 700 MB RAM
  - 64-bit Ubuntu 11.04
- SpecJBB2005 Benchmark:
  - 6 warehouses
  - 120 second run
- Unless specified otherwise, test runs are with 10% host overcommitment and 60 MB memhog inside the guest

Application Responsiveness: Total Pause/Blockage Time

Gaps in the per-second logged instantaneous scores

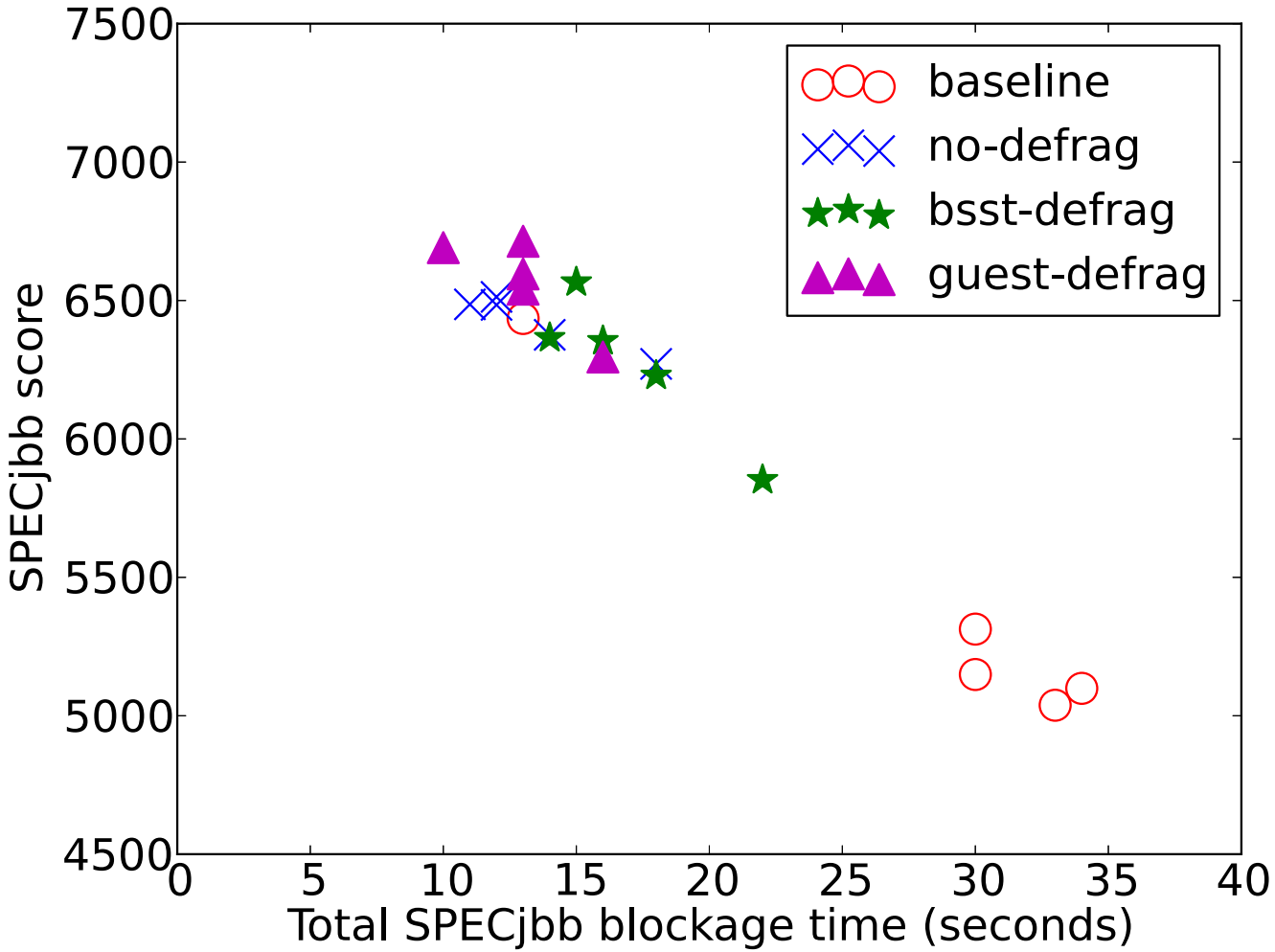


Application Responsiveness: Largest Single Pause



Memhog size: 60 MB

Application Performance with Defragmentation

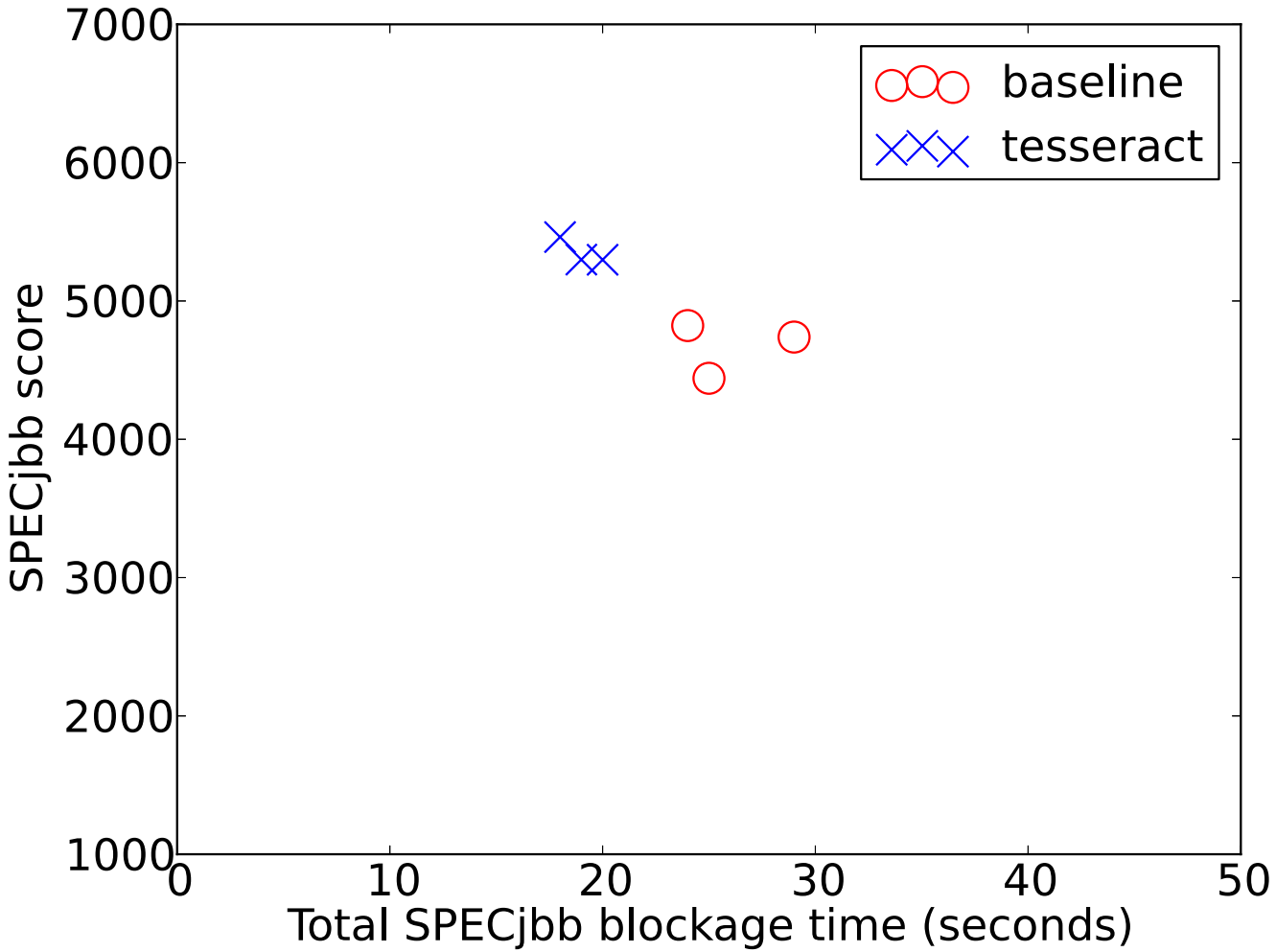


Time Spent in various I/O Paths

Average read and write prepare/completion times (in microseconds)

I/O Path	Baseline	Tesseract	
		No-defrag	Guest defrag
Write prepare	24,262	220	265
Write completion	0	49	101
Read prepare	0	37	109
Read completion	0	232	55

Application Performance with BSST placed on an SSD



Memhog size was 60 MB.

Conclusion and Future Directions

- Scatter-gather lists too closely tied to the traditional physical disks
- Extend to guest-write followed by swap scenario
- Also extend to deduplicate guest I/Os following guest I/Os
  - complete I/O deduplication with sea-of-blocks containing all data blocks
  - guest VMDKs simply contain indirections to the sea-of-blocks
- Throttle defragmentation according to disk load
  - VMware ESX throttles disk I/O (Manning and Dieckhans 2010)

Question?