



# ***GLASS: A Share Everything Architecture for Seaside***

*Dale Henrichs*

*6/20/2008*



- ▶ Wikipedia on Share Nothing
  - ...a distributed computing architecture in which each node is independent and self-sufficient...
  - an SN system may partition data among many nodes...or may require every node to maintain its own copy of the application's data...
  - there is some debate about whether a shared database (clustered or otherwise) should be counted as SN...

- ▶ Every time you need data you hit the database.

- ▶ As Smalltalkers, 'Share Everything' is second nature
- ▶ We do all of our work in an image
  - the image is the database for most of our work
- ▶ We expect to have all of our objects at our finger tips – a message send away

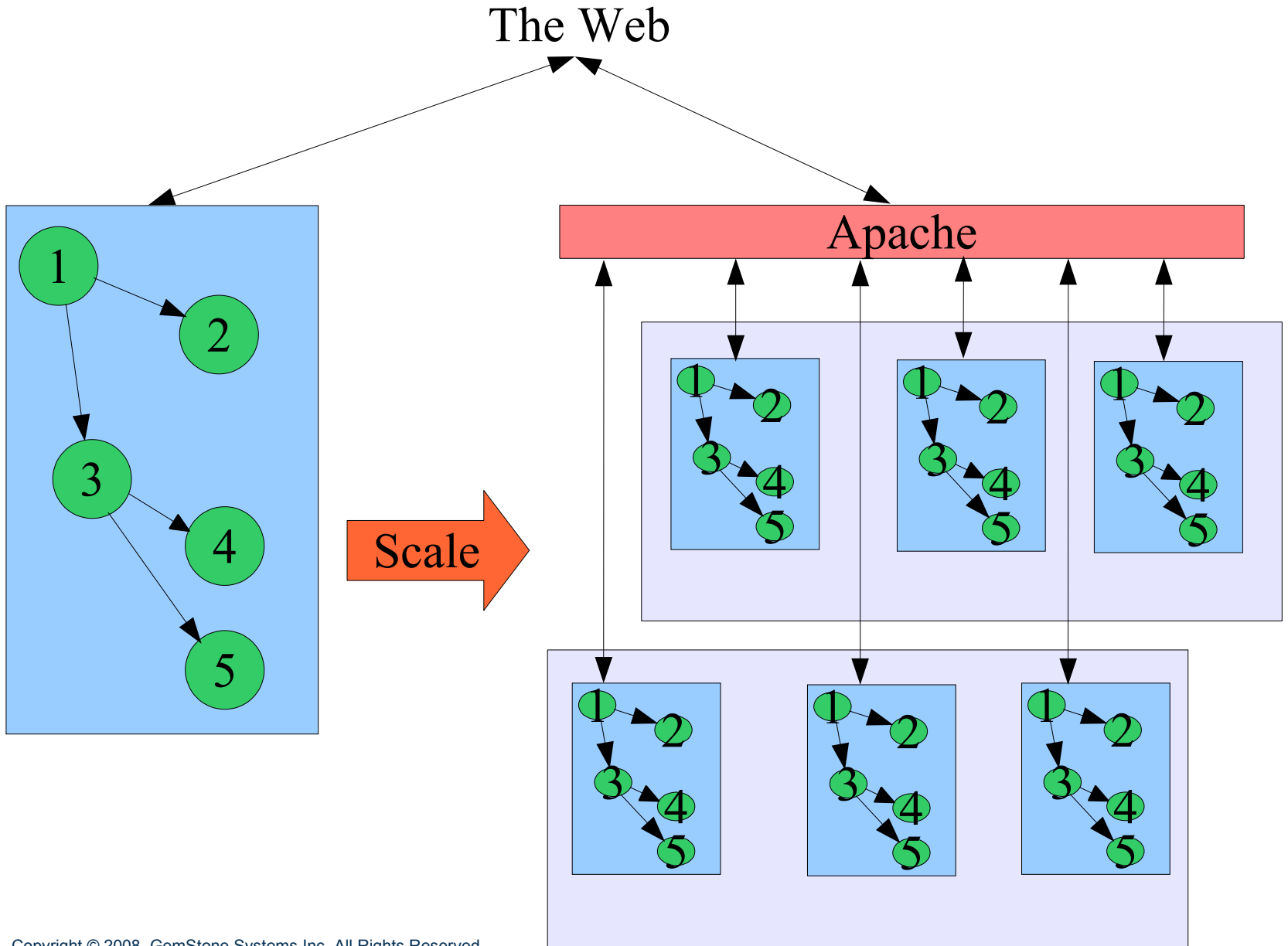
- ▶ **GemStone, Linux, Apache, Seaside, Smalltalk**
  - transparent persistence
  - transparent scalability
  - preserve 'intimacy' of image-based development

- ▶ GemStone/S persists all objects that are reachable from a persistent root
  - identical to image-based persistence
- ▶ Use class variables or class instance variables for your object model
  - Seaside application can run in Squeak and GLASS without any changes
- ▶ Transactional semantics for persistence
  - commit to save 'image' changes
  - abort to revert 'image' changes

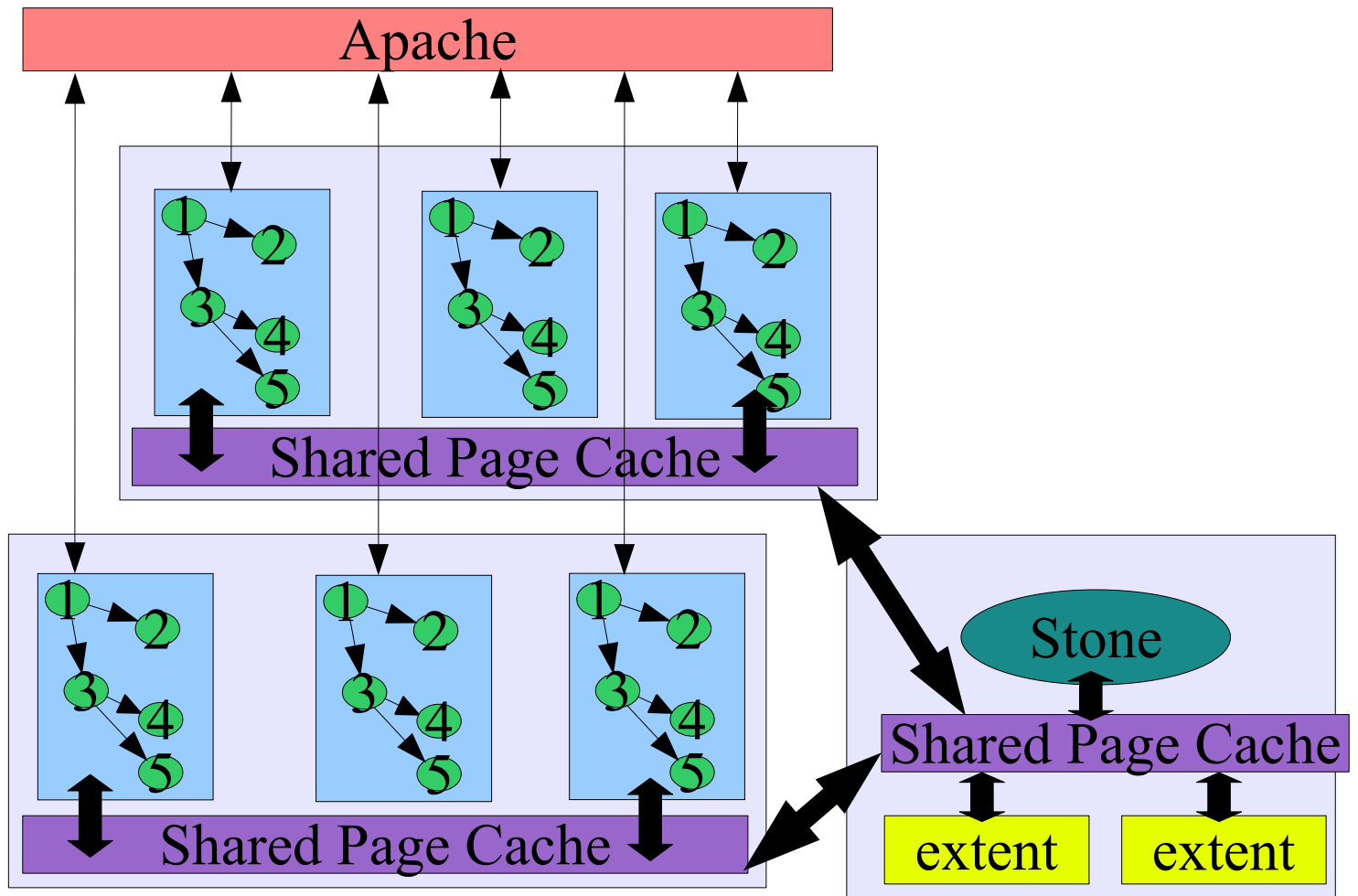
- ▶ Extended Seaside framework
  - abort when an HTTP request is received
  - commit before HTTP response is sent
- ▶ No need to embed transaction logic in a Seaside application
  - write application as if you are using image-based persistence

- ▶ Scalability for a Seaside application means that to handle additional load, you just add more resources without changing your application
  - more VMs to handle requests
  - more CPUs/machines/disk

# GLASS Scalability



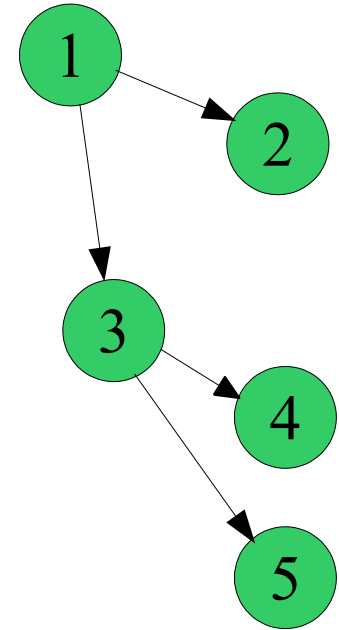
# GLASS Architecture



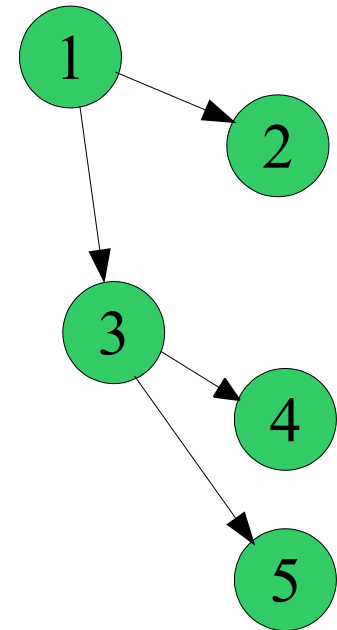
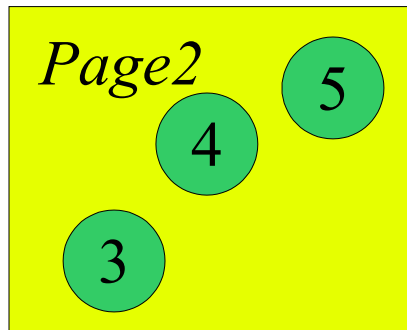
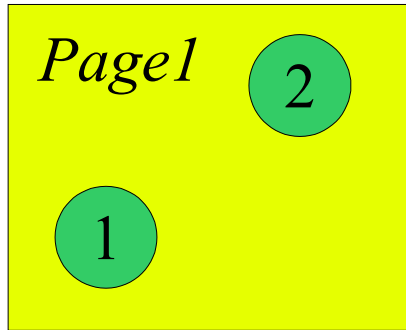
- ▶ A Shared image is the key to GemStone scalability story
  - Multiple VMs running on multiple machines can operate on single shared image

- ▶ Image Sharing is possible
  - Object Table (OT)
  - Shared Page Cache (SPC)

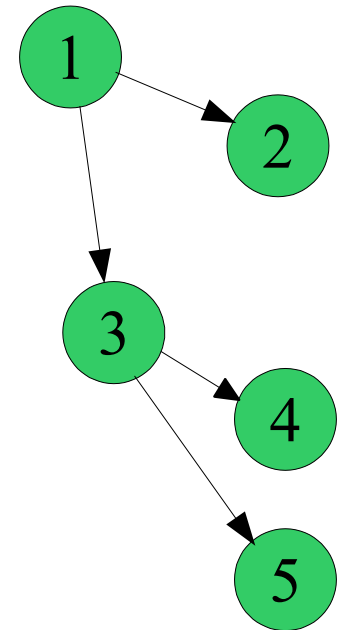
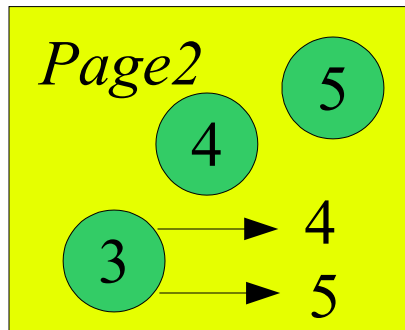
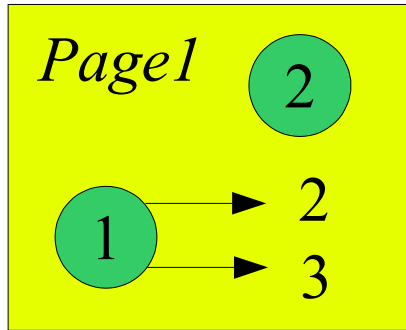
# Every Object has an Oop



# Objects stored on Pages

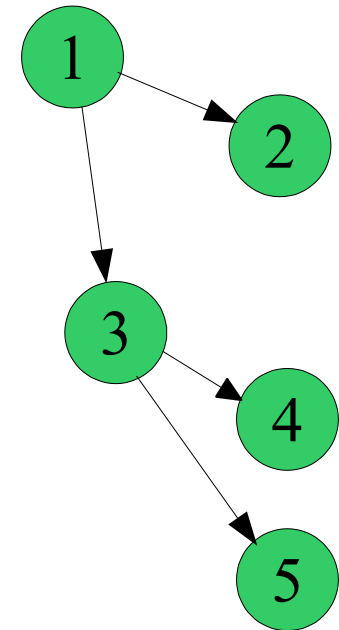
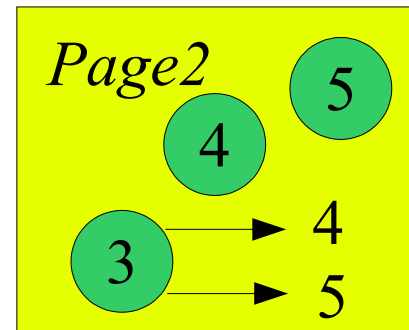
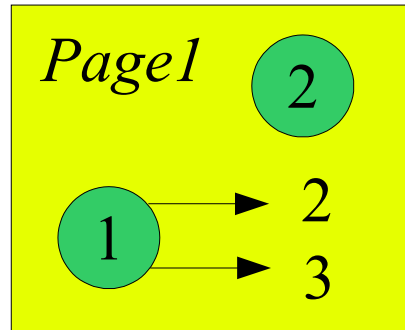


# Object reference by Oop

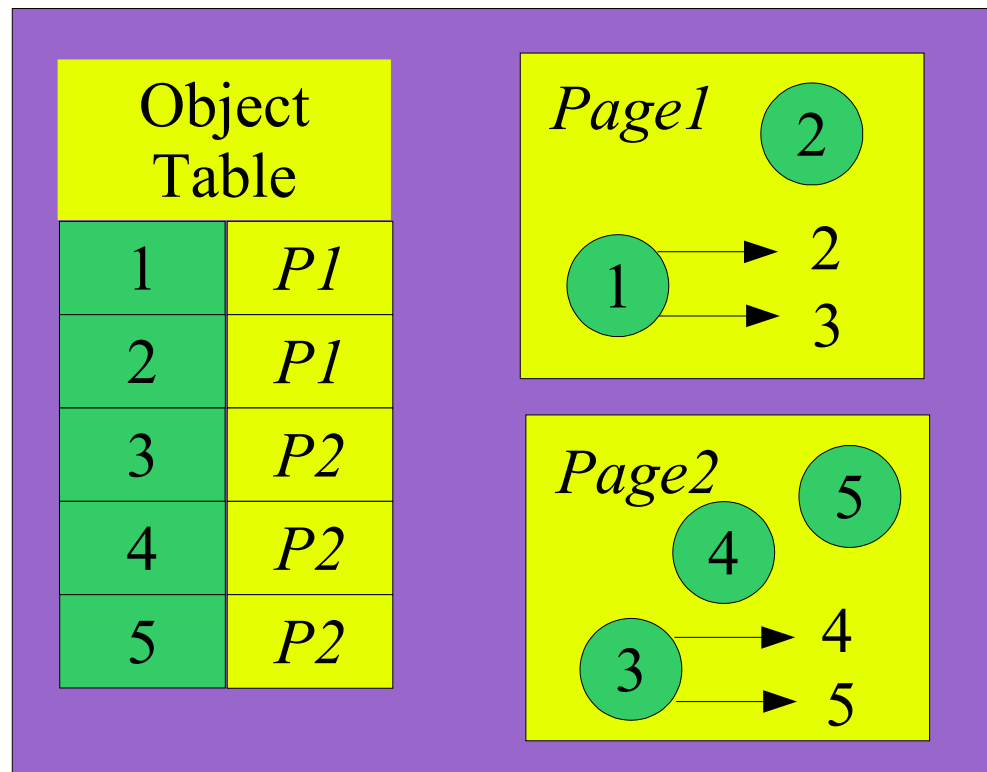


# Object Table maps oops to pages

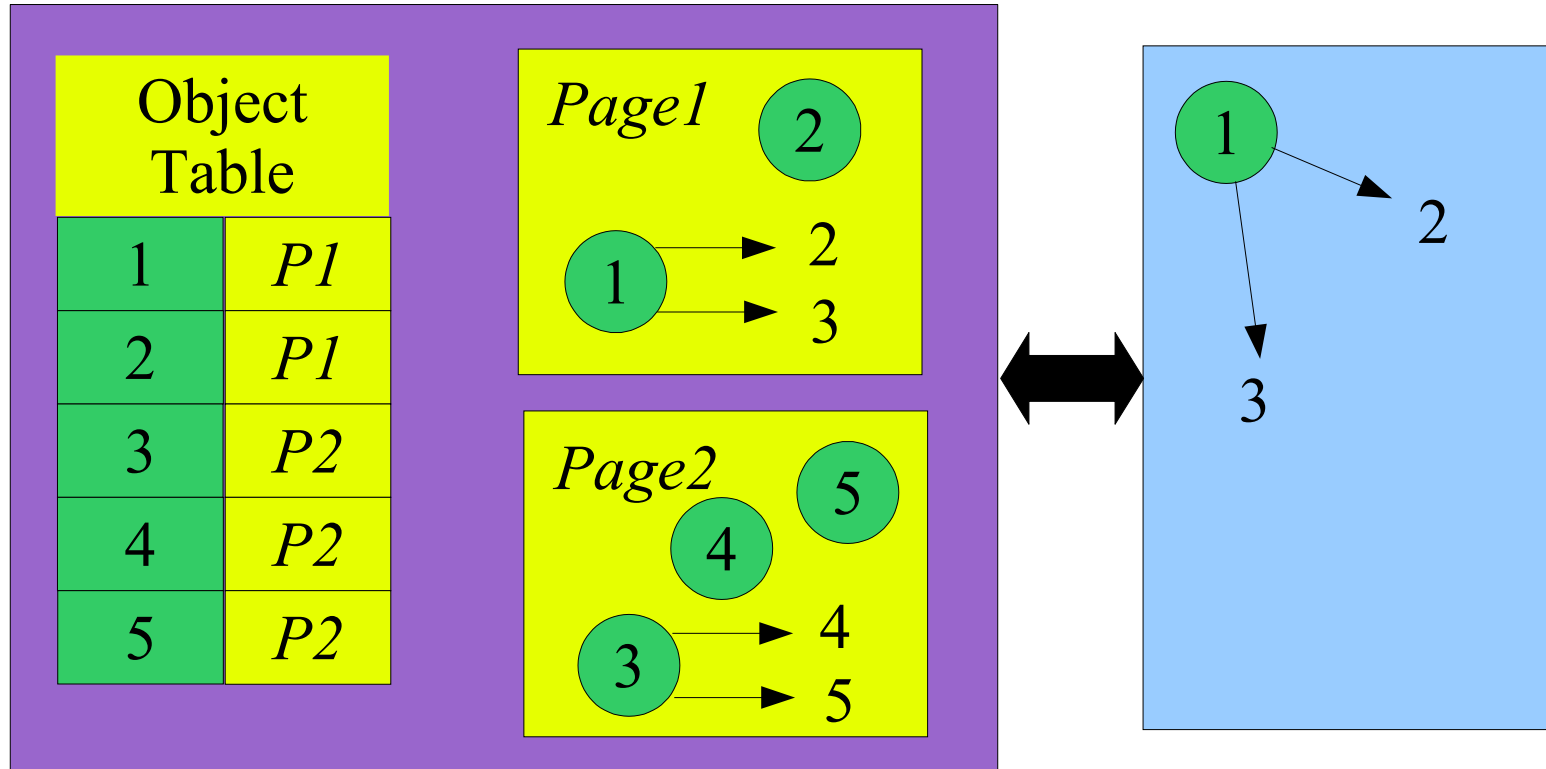
Object Table	
1	<i>P1</i>
2	<i>P1</i>
3	<i>P2</i>
4	<i>P2</i>
5	<i>P2</i>



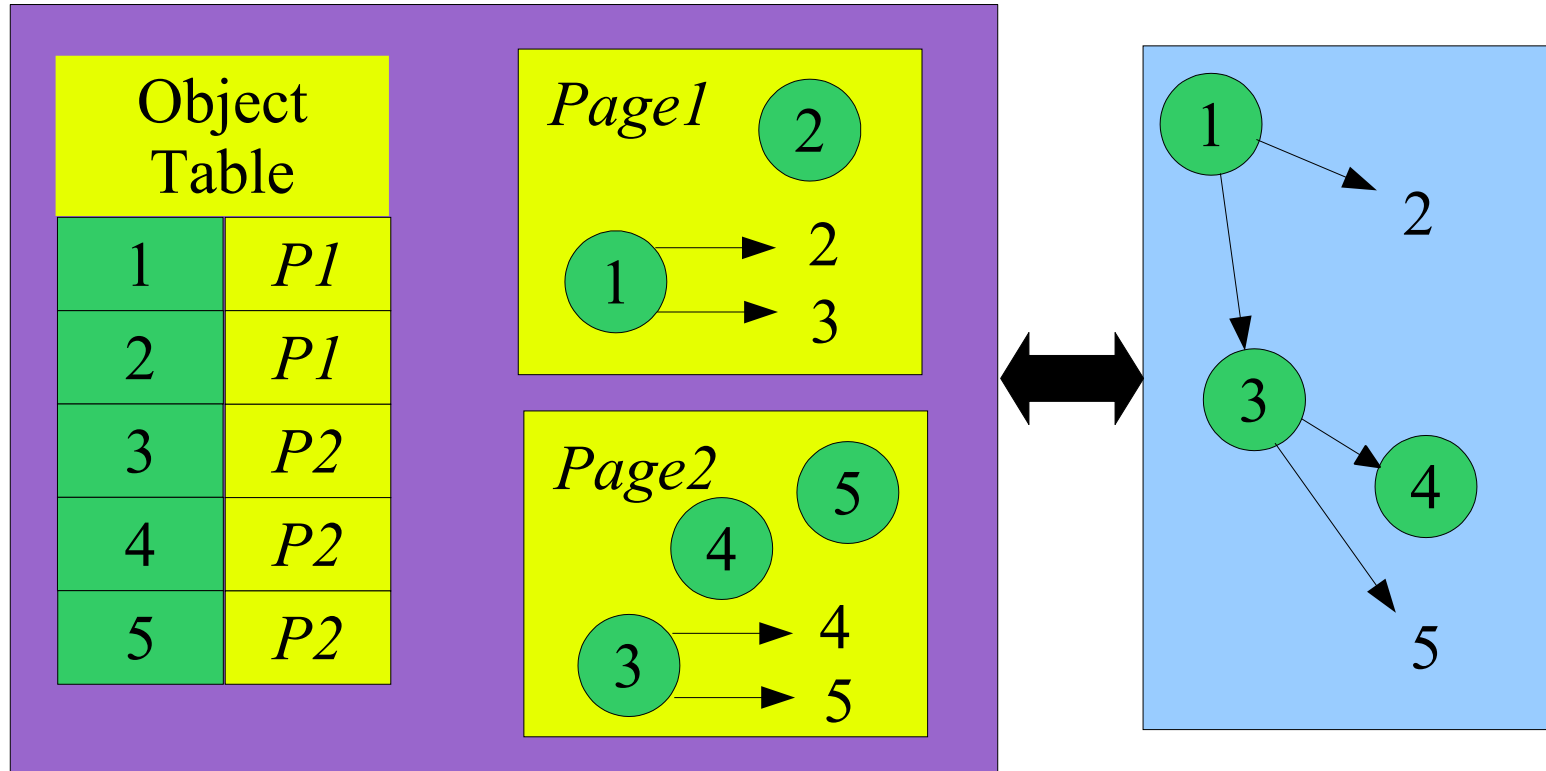
# Pages copied into SPC (in shared memory) from disk



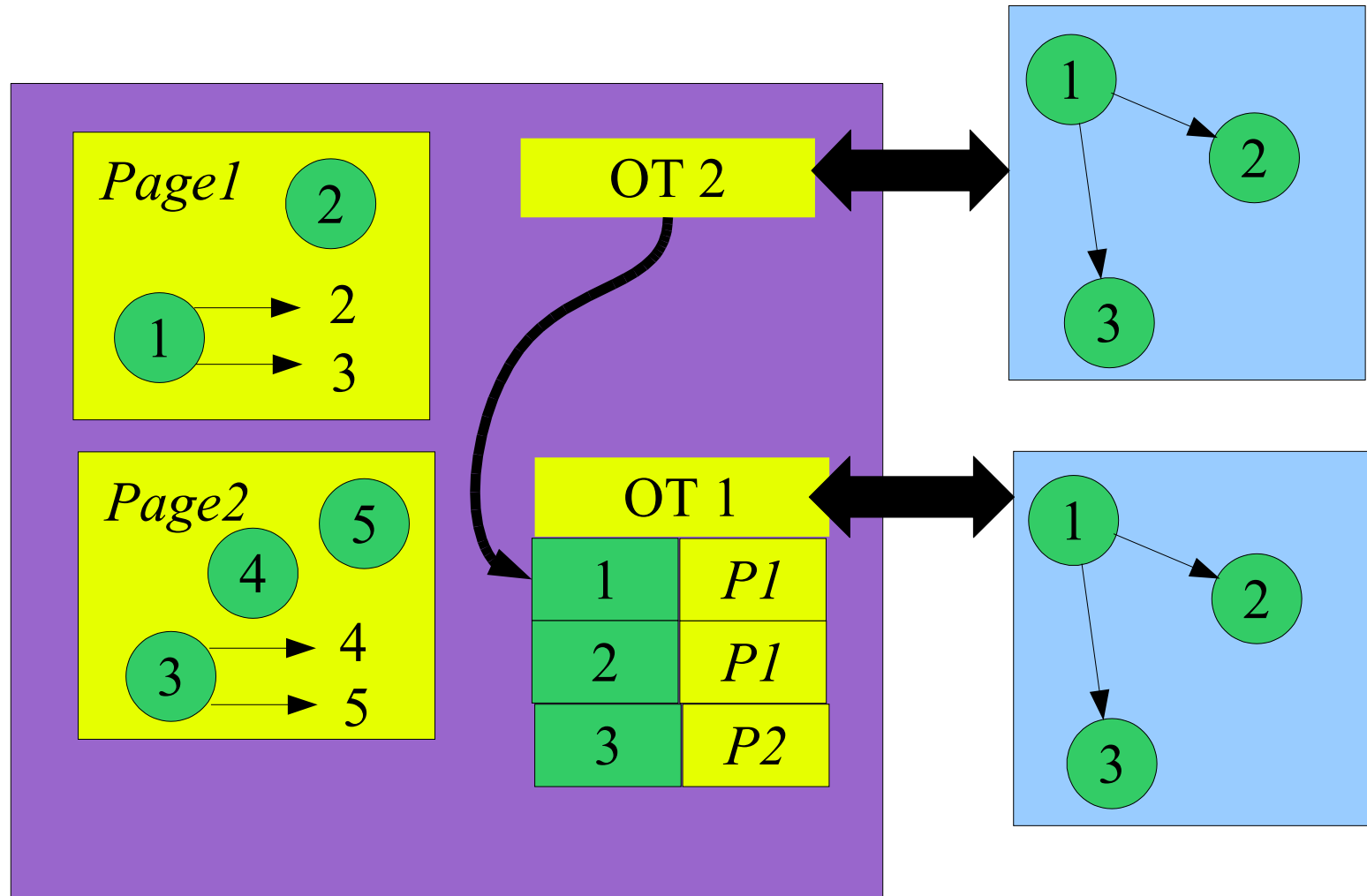
# Objects copied into VM object memory from SPC



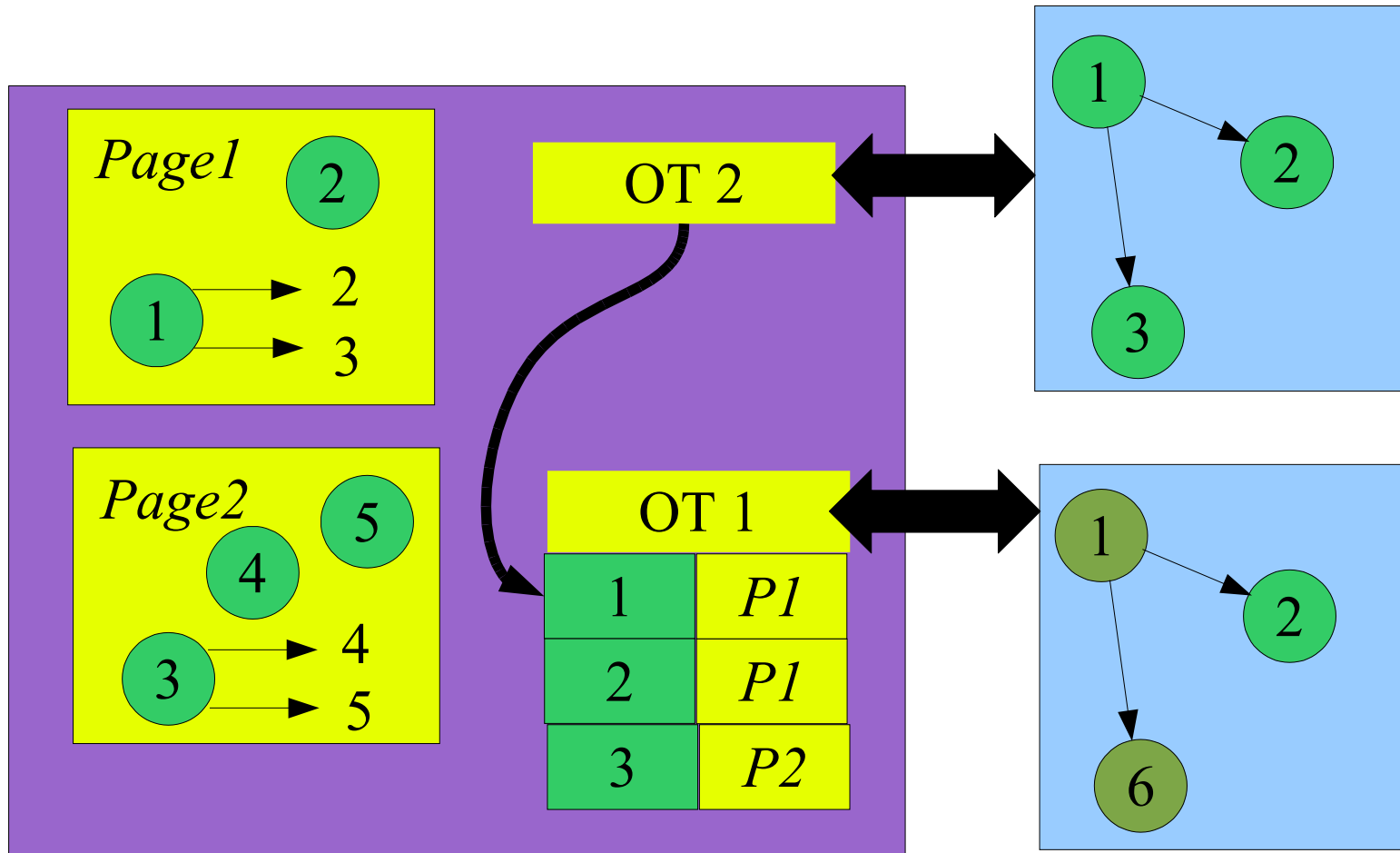
# Objects faulted into VM on reference



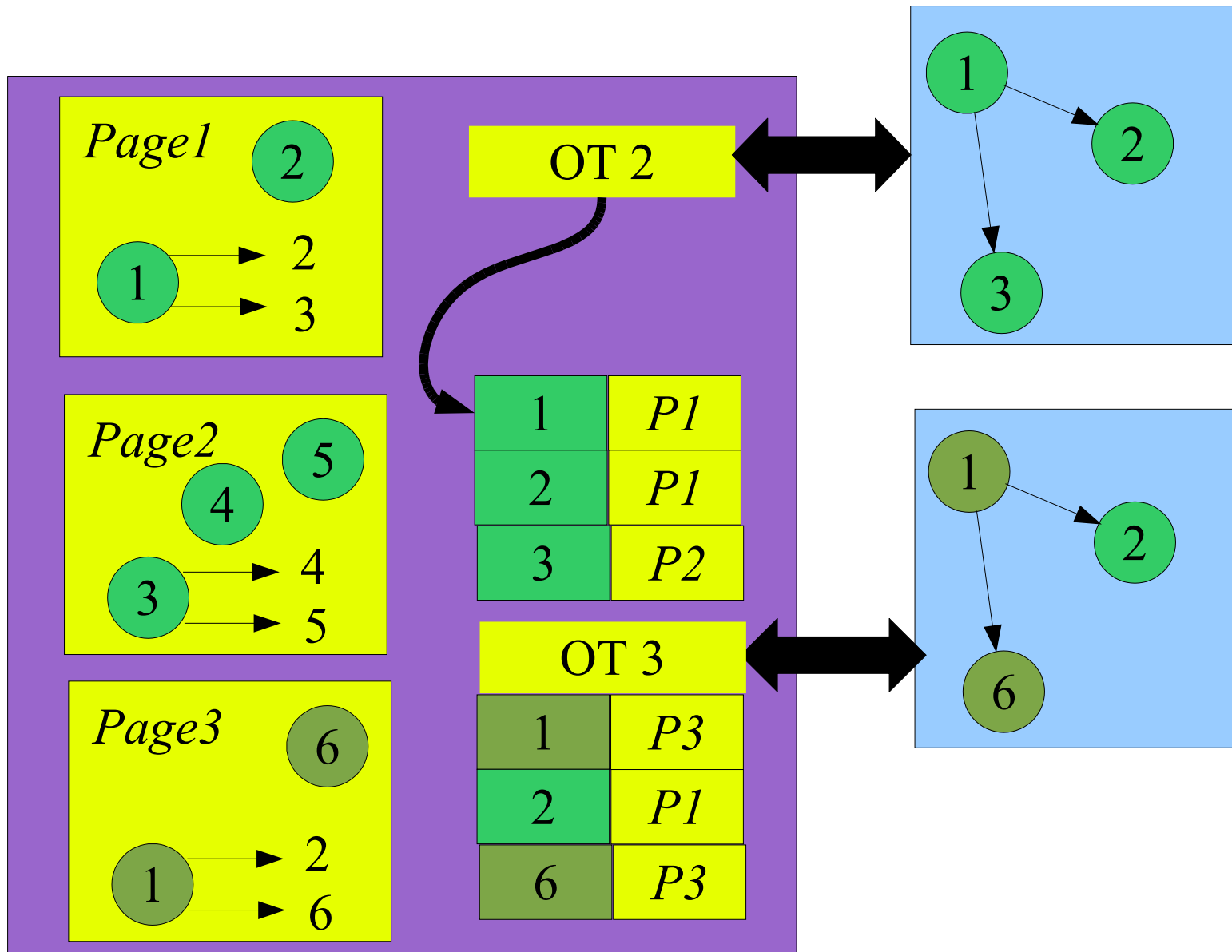
# Adding a second VM



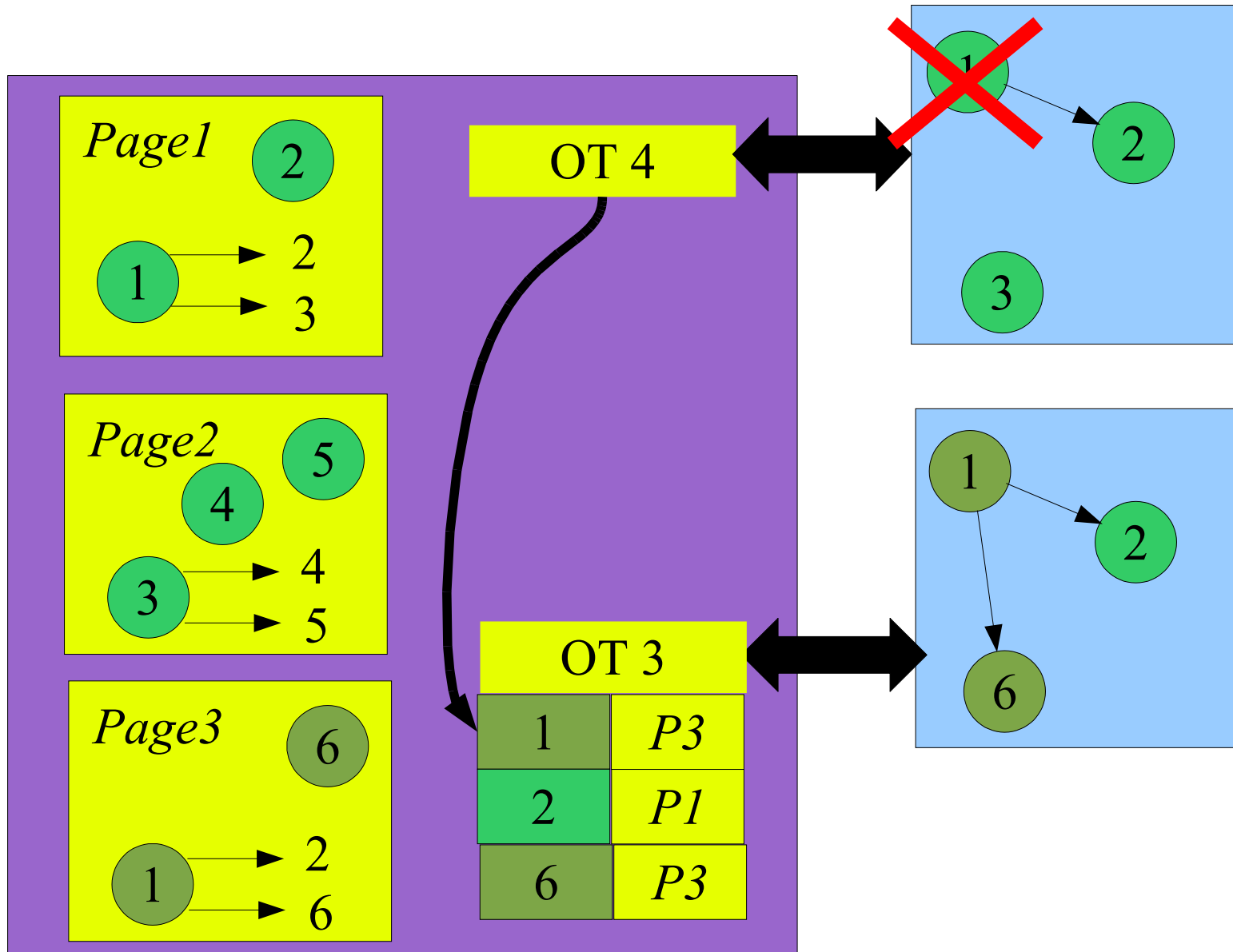
# Making Changes in one VM



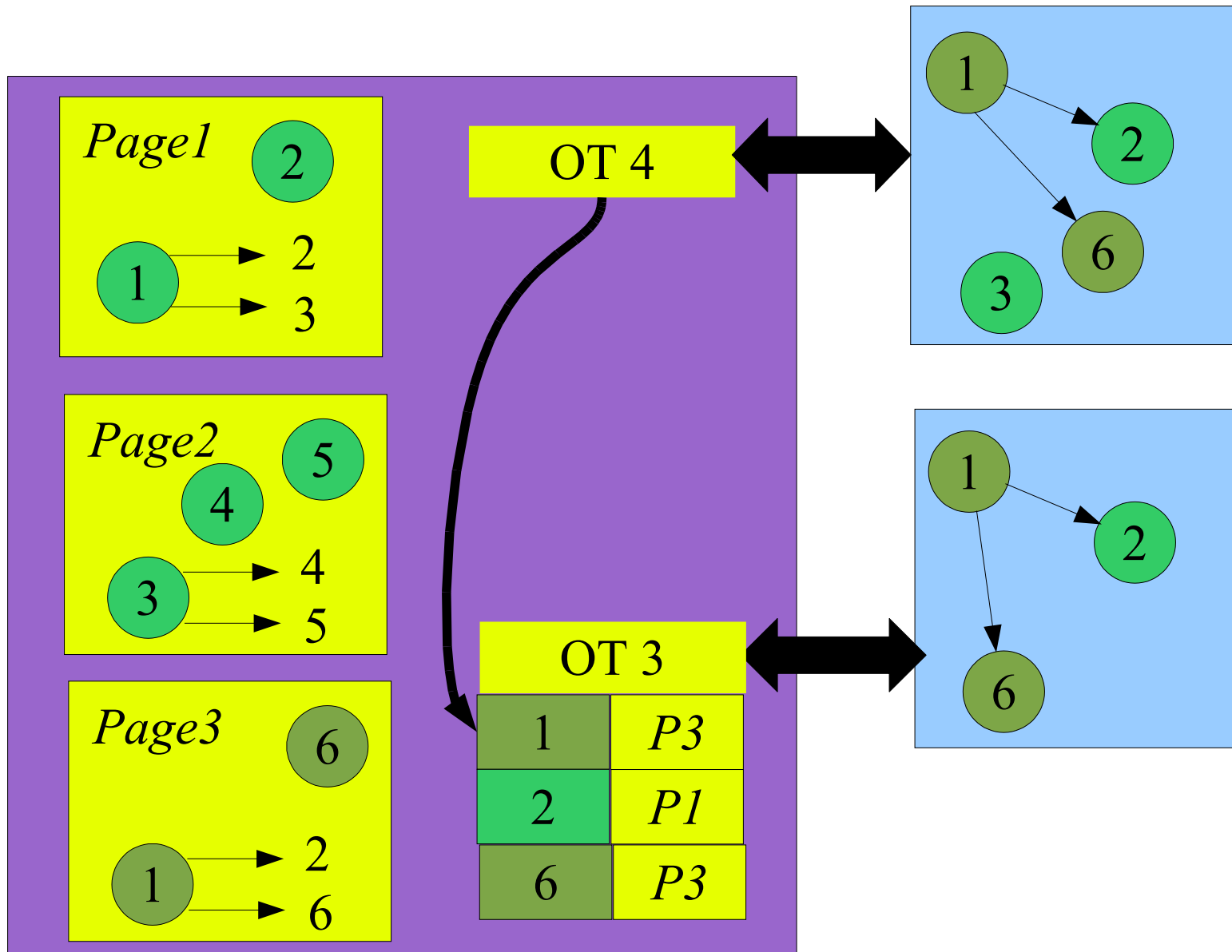
# Commit in vm2



# Abort in vm1

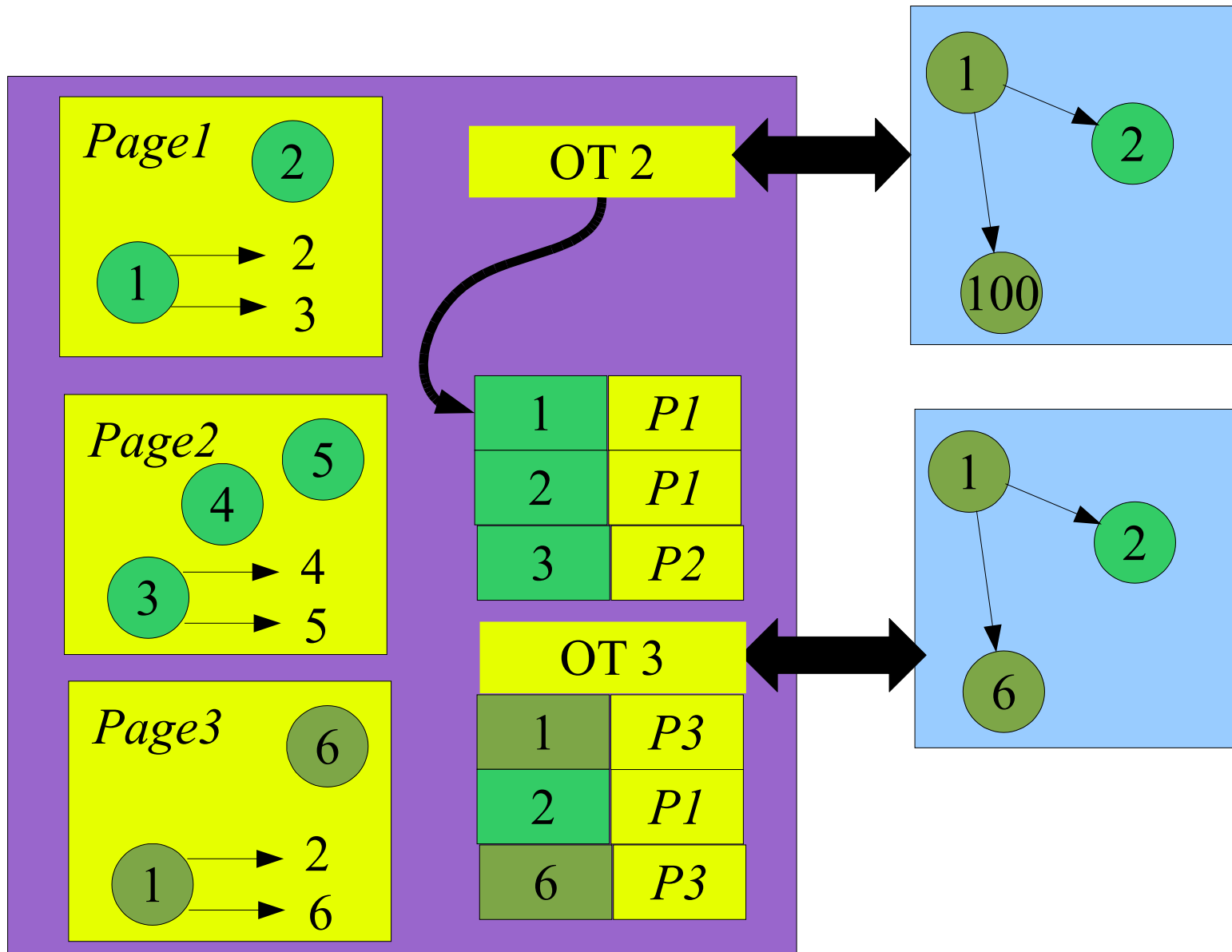


# Abort in vm1



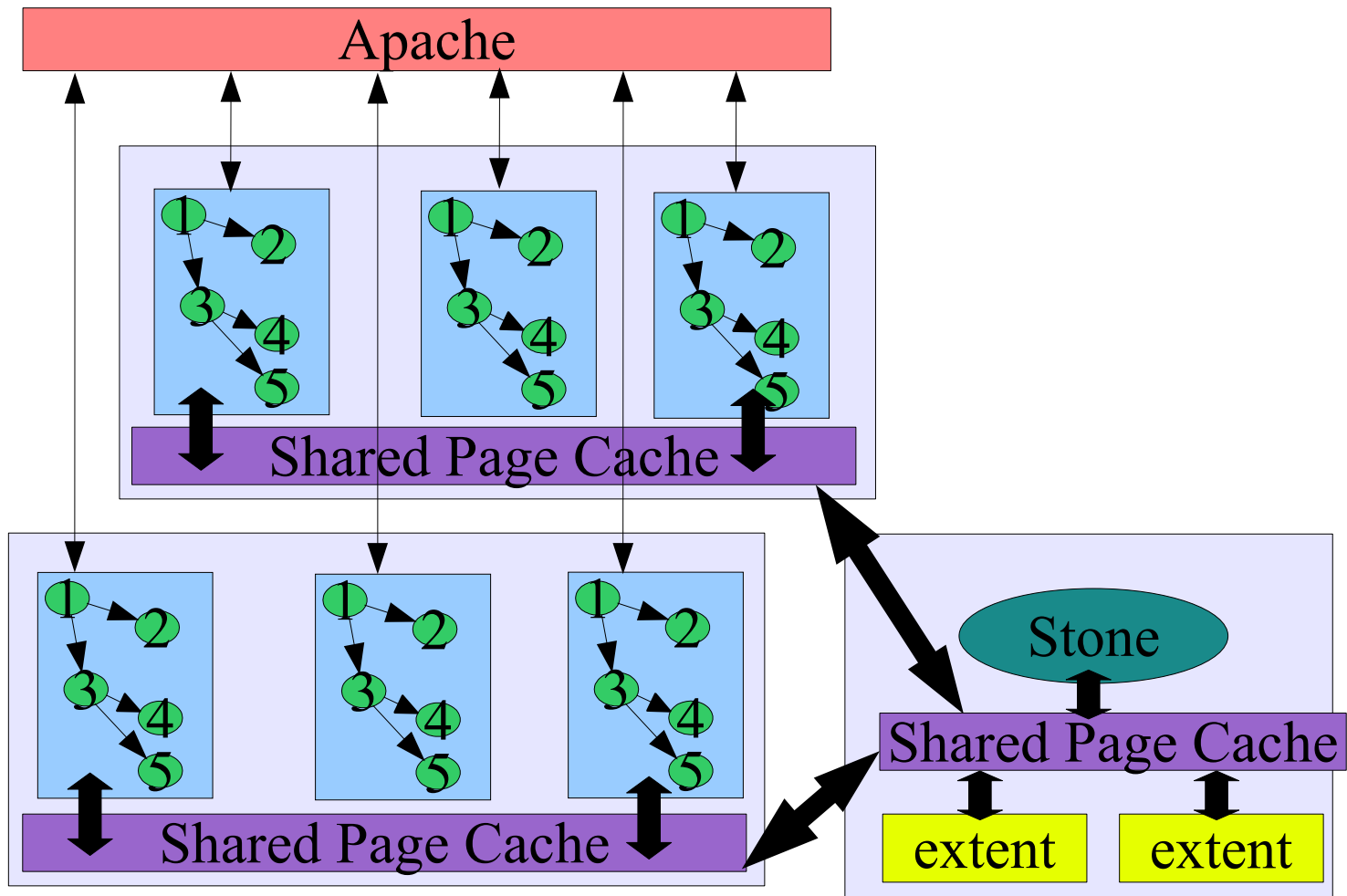
- ▶ A conflict occurs if two vms attempt to change the same object at the same time

# Potential conflict

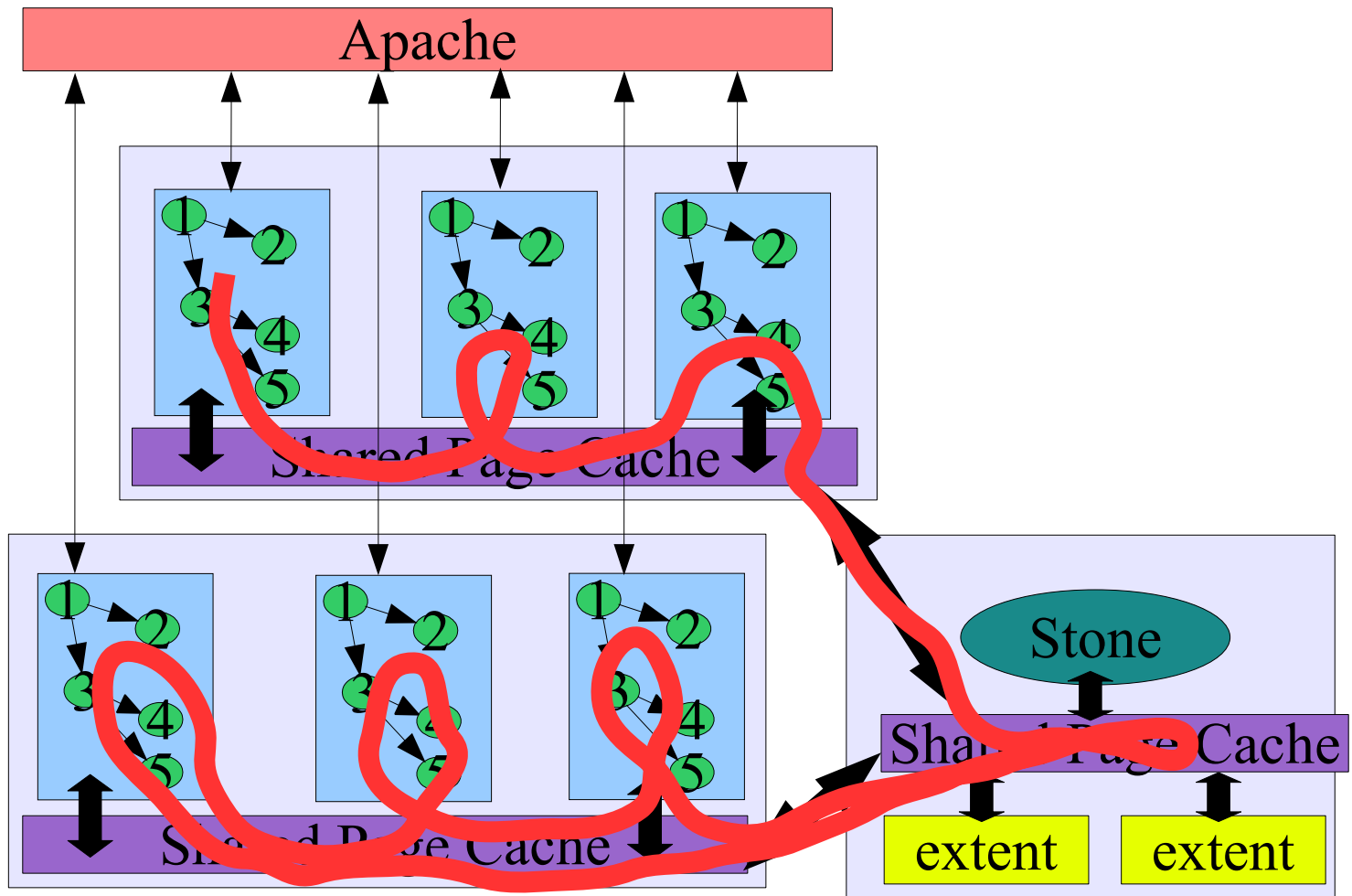


- ▶ abort when HTTP request received
- ▶ commit when HTTP response is ready to send
  - On conflict simply retry HTTP request

# GLASS a Scalable architecture

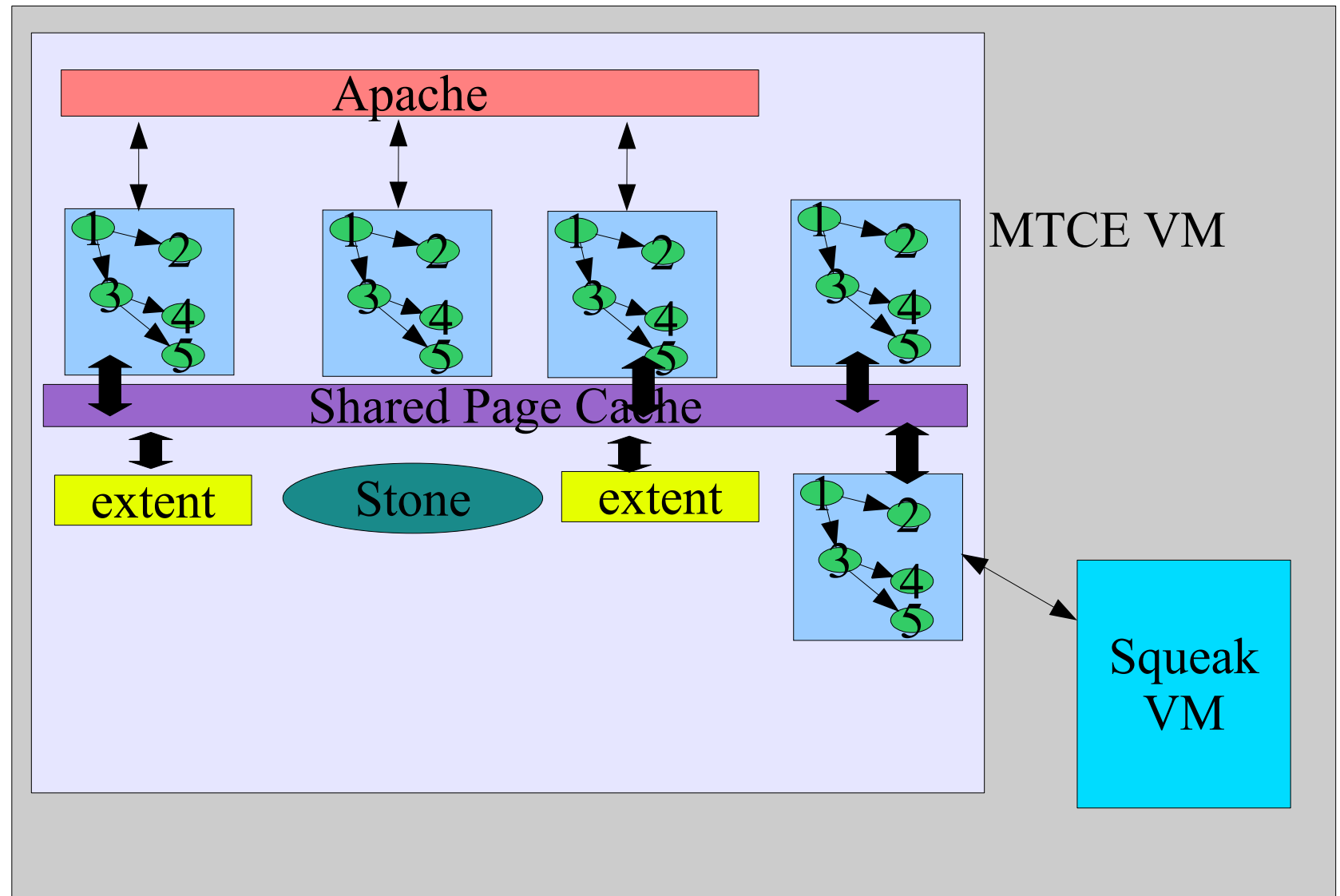


# GLASS a Scalable architecture



- ▶ Transparent persistence
- ▶ Transparent scalability
- ▶ Transparent development?

# GLASS Appliance



- ▶ auto commit
- ▶ object log
- ▶ debugging
  - halt
  - breakpoints
- ▶ Transcript

## ▶ Seaside2.9

- experimenting with reducing/eliminating redundant session state

## ▶ GemStone 3.0

- non-Tranlogged objects
- Native methods
- Improved Exception Handling
- Foreign Function Interface

## ▶ ...and Beyond?

- sharding for GemStone