# Translation, Invalidation, and Large Pages

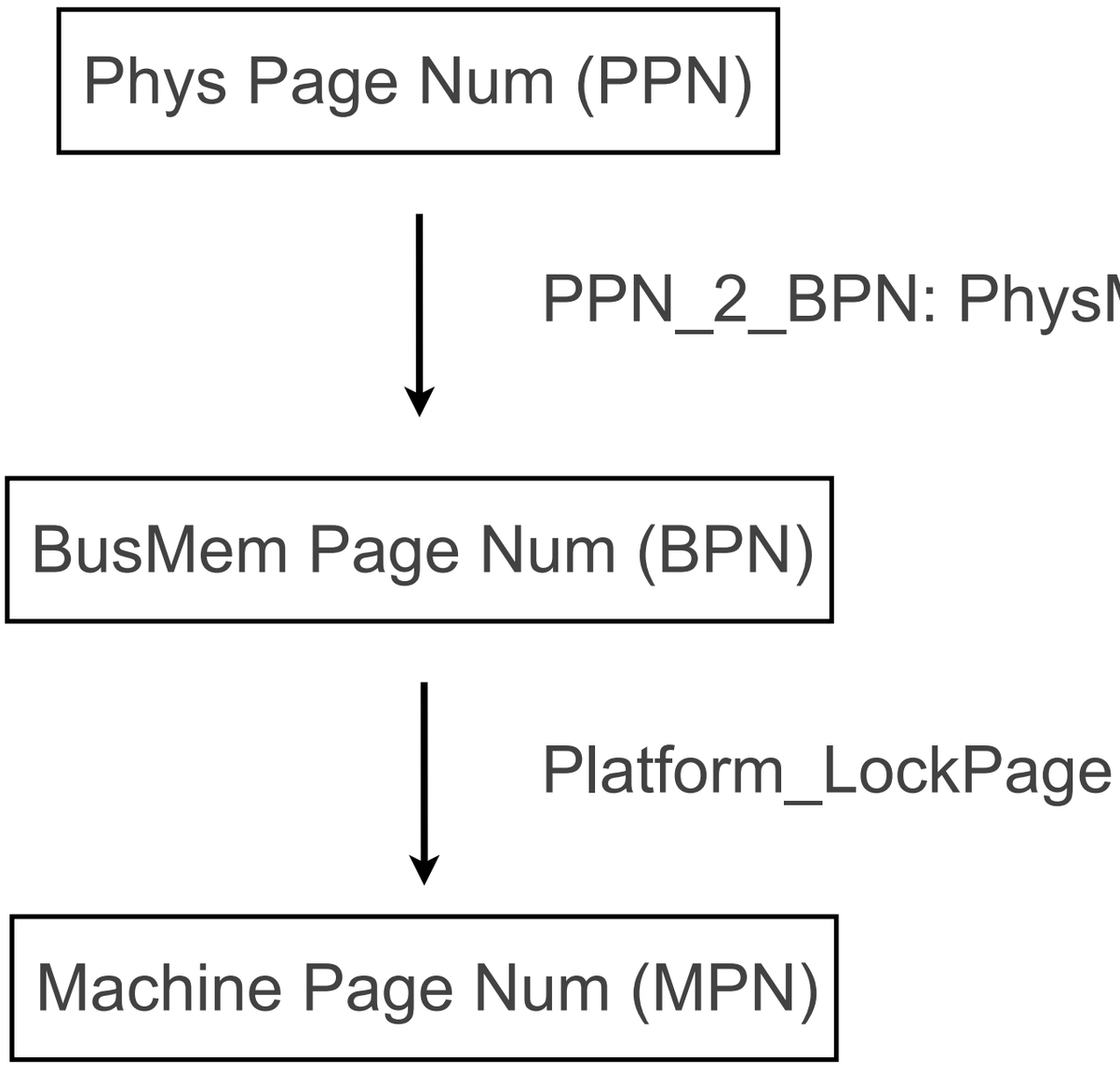
*Alex Garthwaite*

*Monitor Group*

*December 20th, 2012*

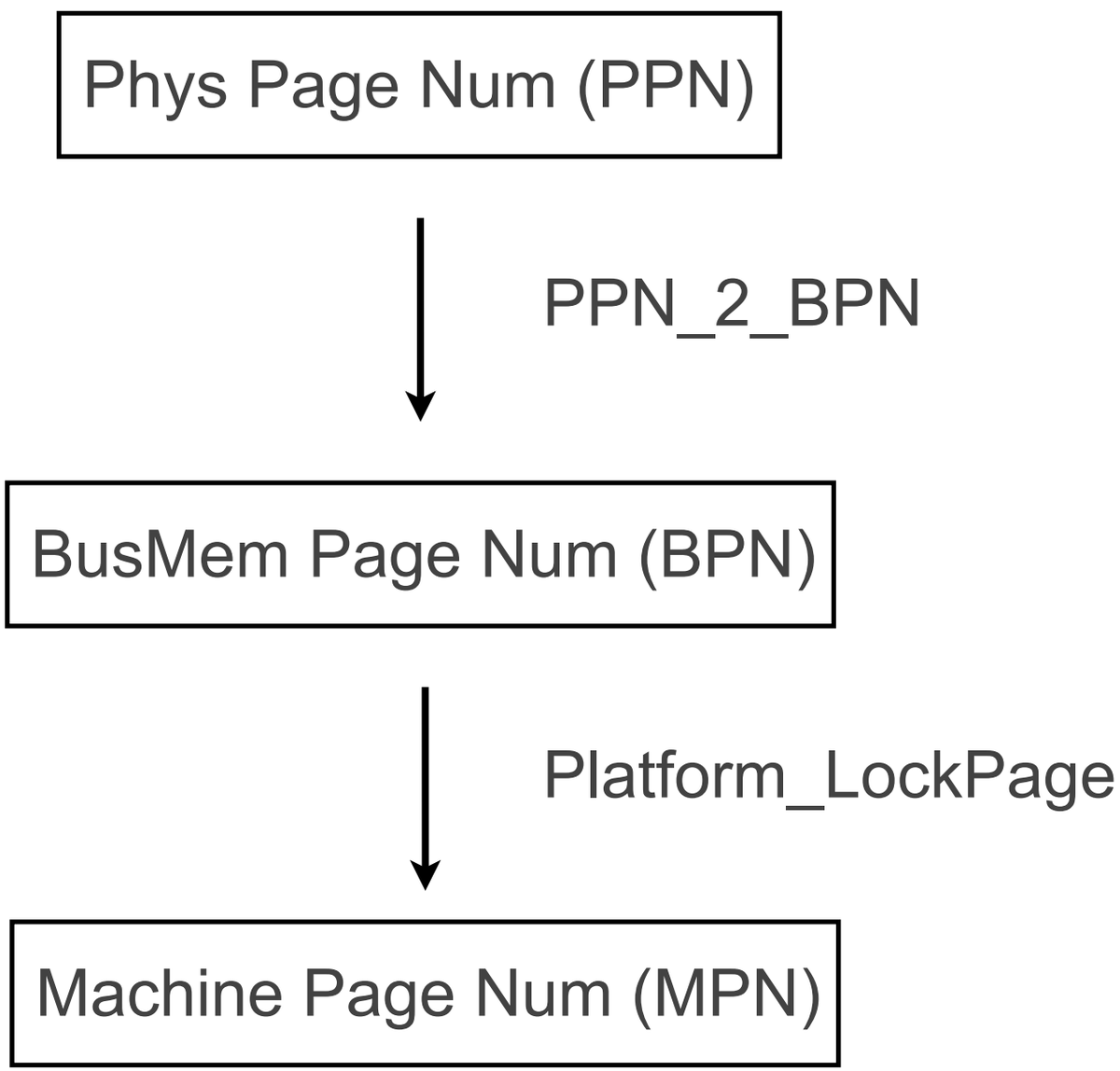
# Outline: mapping BPNs to MPNs

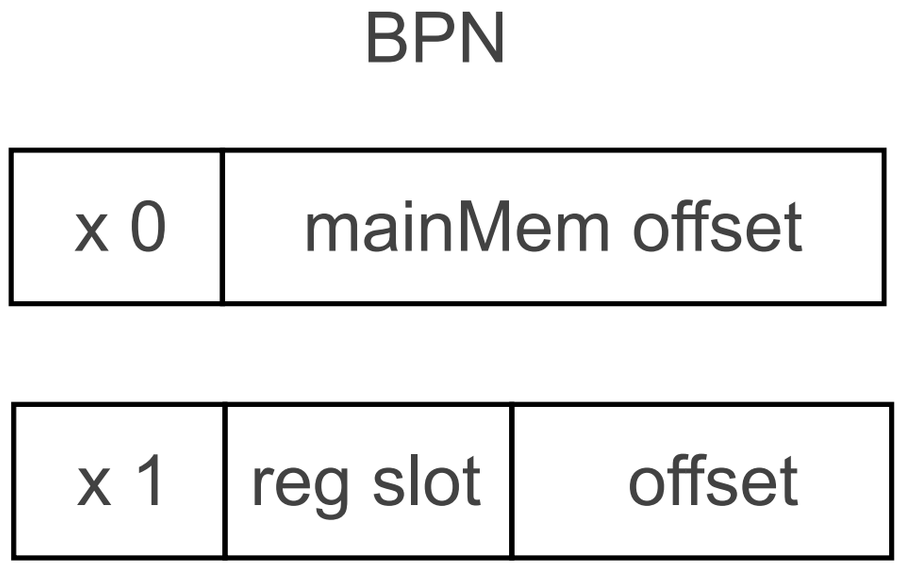
- **Translation**
- **Invalidation of mappings**
- **Large page heuristics**



# Outline: mapping BPNs to MPNs

- **Translation**
- **Invalidation of mappings**
- **Large page heuristics**



# Translation from BPNs to MPNs

- **BusMemFrames**
  - one per 4KB page
  - tracks properties, trace information, (for sw-mmu) MPN
  - mapped, dirty flags
- **GPhys page-tables**
  - cache MPNs when mapped
- **BusMem2MRegionInfo**
  - one per 2MB page
  - backedLarge, mappedLarge, dirtiedLarge, sampledLarge count
  - backoff mechanism
- **MainMem BPNs**
  - only ones released (swapped), shared, backed large
  - only kind vmkernel understands
  - only kind backed with large pages
    - only ESX uses large pages
    - but Philip Langdale's hosted patches exist
- **Platform**
  - pframes as canonical source of state about mappings: exposedToVMM, MPN

# Translation from BPNs to MPNs

- **Pinning mappings**
  - per-vcpu
    - clients: monTLB, MMU, VHV
  - global: pinCounts
    - BusMemPinFrame, BusMemAdjustPinCount
    - local shared buffer of <BPN, delta> values
    - flushed to platform's pframe pin-counts
- **Shared pages**
  - cannot be pinned (even for reads)
    - subsequent writes may need to break mapping

# Translation from BPNs to MPNs

- **Issues/Challenges**
  - scaling with VCPUs: busmem lock contention
  - scaling with large(r) page sizes
  - concurrency with respect to other translations, invalidations, pinning
  - concurrency with activity in vmx and vmkernel
  - platform constrained in its ability to allocate/break large pages

# Translation from BPNs to MPNs

**BusMem_TranslateBusFrame(bpn, frame, flags)**

```
MPN mpn = INVALID_MPN;
if (mainMem(bpn) && large-page needed) {
   mpn = BusMem_TranslateLargeBusFrame(bpn, frame, flags);
}
if (mpn == INVALID_MPN) {
   mpn = BusMemTranslateSmallBusMemFrame(bpn, frame, flags);
}
return mpn;
```

# Translation from BPNs to MPNs

**BusMemTranslateSmallBusFrame(bpn, frame, flags)**

```
mpn = BusMem_GetMPN(bpn);
if (mpn != INVALID_MPN || set dirty || is sampled || breaking COW) {
   do {
      if (breaking COW) {
         BusMemLock();
         zap BPN, break COW in platform, BusMem_ResetValidatorsForBPN(BPN)
         BusMemUnlock();
      }
      open transaction(bpn);
      mpn = BusMemLockGuestPage(bpn, guestFlags, &platformFlags);
      BusMemLock();
      validate transaction and mapping (in GPhys),
         mpn = INVALID_MPN on failure
      BusMemUnlock();
   } while (mpn == INVALID_MPN);
}
BusMemPinFrame(bpn, frame);    // only one unpinned MPN allowed
return mpn;
```

# Translation from BPNs to MPNs

**BusMem_TranslateLargeBusFrame(bpn, frame, flags)**

```
mpn = BusMem_TryGetMPN(bpn);
if (mappedLarge, !zapping, !sampledLarge, !needs dirtyLarge)
   return mpn;
mpn = INVALID_MPN;
if (!mappable large) return mpn;     // racy check
BusMemLock();
   if (mappable large) {
      if (backed large) {
         handle notifying platform, upgrading to mappedLarge
      } else {
         zap any existing small pages, allocate 2MB page via Platform_LockPage()
            which handles copying small-page contents to new large page
         if successful, populate 2MRegion frames
      }
   }
BusMemUnlock();
BusMemPinFrame(bpn, frame);
return mpn;
```

# Notes on Translation

- **unify two translation paths**
  - make large-page path transactional like small-page page:
    - preamble to handle zapping of 2M region, requesting VMMEM_GUEST_LARGE_PAGE
    - single (unified) Platform_LockPage call
    - postamble under busmem lock which validates transaction and populates 2M region if a large page is obtained
- **backed large v. mapped large constraints**
  - backed-large decisions should be in platform: ballooning, pinning, alloc issues
  - mapped-large could be either but some are definitely monitor-specific: tracing
    - currently sampling is only service that (if moved to platform) would restrict mapped-large decisions there
    - possibly a reason not to move (all of) sampling to platform
- **sampling backoff**
  - should evaluate
- **failure backoff**
  - currently, a simple exponential backoff that has caused scaling/perf problems in past
  - eliminate in favor of a bitmap/array that the platform populates with reasons why a large-page backing (or mapping) is not possible
- **validation transactions and resetting**
  - this and the use of zap-crosscalls are the main ways that concurrent translation is kept correct
  - BusMem_ResetAllValidators() and BusMem_ResetValidatorsForBPN(bpn)

# Invalidation

- **Three types of calls**
  - BusMemZapPageOnAllVcpus
  - BusMemZapPageRangeOnAllVcpus
  - BusMemZapPageListOnAllVcpus
- **Basic sequence**
  - preamble
    - clear mapped state for batch of pages
    - store off MPNs
    - mark 2MRegions where zaps happening
  - if needed, do pin-check crosscall to all VCPUs
    - not needed for !mapped or shared pages
  - add opportunistic cases (other VCPUs waiting to break sharing)
  - issue zap-crosscall
  - clear large-page state, call platform to break large pages
  - clear zapping flags for 2MRegions

# Invalidation

- **Three types of calls**
  - BusMemZapPageOnAllVcpus
  - BusMemZapPageRangeOnAllVcpus
  - BusMemZapPageListOnAllVcpus
- **Basic sequence**
  - preamble
    - clear mapped state for batch of pages
    - store off MPNs
    - mark 2MRegions where zaps happening
  - if needed, do pin-check crosscall to all VCPUs
    - not needed for !mapped or shared pages
  - add opportunistic cases (other VCPUs waiting to break sharing)
  - issue zap-crosscall
  - clear large-page state, call platform to break large pages
  - clear zapping flags for 2MRegions

**lpage transl., releases**

**expensive**

# Invalidation

- **Three types of calls**
  - BusMemZapPageOnAllVcpus
  - BusMemZapPageRangeOnAllVcpus
  - BusMemZapPageListOnAllVcpus
- **Basic sequence**
  - preamble
    - clear mapped state for batch of pages
    - store off MPNs
    - mark 2MRegions where zaps happening
  - if needed, do pin-check crosscall to all VCPUs
    - not needed for !mapped or shared pages
  - add opportunistic cases (other VCPUs waiting to break sharing)
  - issue zap-crosscall
  - clear large-page state, call platform to break large pages
  - clear zapping flags for 2MRegions
- **Improvements**
  - make pin-checks, zap handlers work on ranges of BPNs, not individual BPNs
  - improve crosscall performance
  - don’t break large pages in platform (allow release vmkcalls to do so)

# Zap CrossCalls and Scaling

- **Source Files**
  - vmcore/vmm/main/crosscall.c, vmcore/vmm/private/crosscall.h, vmcore/vmm/private/crosscall_defs.h, vmcore/vmm/public/crosscall_ext.h
- **CrossCall types**
  - early-release: variants that allow invoker to continue on receipt of CC request
  - async: allows offline VCPUs to queue action
  - none: blocks invoker until all vcpus complete handlers (barrier)
- **EXECUTE_CB, EXECUTE_BUSMEM_CB, EXECUTE_LEAF_CB**
  - EXECUTE_BUSMEM_CB used by zap crosscall
  - these are all neither early-release nor async
  - these differ in the lock rank they execute at
    - would be nice to parameterize rank
- **Zapping can be relaxed**
  - make early-release: handler won't access/use MPNs
  - async modes
    - allow buffering of zap-request state for VCPUs out of vmm
    - allow offline VCPUs to remain offline
- **Need to study scaling of crosscalls for VCPU count**

# Large Pages in Platform

- **Driven by vmm**
  - broken when zapping with BusMemBreakLargePage
  - large-page translation call explicitly asks for large page
- **Limitations/Impact**
  - NUMA-migration sequence inefficient
    - move some subset of small pages
    - rely on subsequent vmm request to remap small contents to large page on new node
  - checkpointing, VMotion similar: moves pages as small
  - leads to fragmentation of large-page pool, remap/zap costs
- **Blocker**
  - ballooned state in platform: what can be backed large
- **VMMEM_PLATFORM_LARGE_PAGE flag returned by LockPage()**
- **If platform allocates large pages, can we eliminate/relax alwaysTryLPageAlloc policy?**

# Large Pages and Sharing

- **Trade-off in performance v. consolidation**
- **Current state**
  - VMs that have swap targets are denied large pages
  - ballooning, swapping breaks large pages, inducing (a bit later) page-sharing
  - (transient) performance hit when platform hits soft state or VMs hit max-limit
- **Lack of insight into large-page use**
  - Yuri/Kiran: categorize large pages based on sharing, access opportunities
  - categorize large pages based on coldness
- **Heuristics**
  - prevent large pages
    - if too much sharing (disabled)
    - if too much swapped (with increasing threshold)
    - if no large pages available, pinning/in-use
  - break large pages for sharing if swap target exists
- **Need spectrum rather than hard cliff**
  - track amount of allocation (new, shared, swapped) to back large
    - as soft state approached, increase threshold on number of allocs allowed
  - track sharing opportunity, coldness
    - as soft state approached, increasingly break coldest, highest-sharing LPages

# Not-Exposed Pool(s)

- **Decouple vmm and platform for releasing memory**
  - fits with model of vmm zapping/forgetting info., platform doing work async
- **Swapping**
  - allows “cooling” of selected pages
  - point to preclean dirty pages to the swap file, spread out swap-related I/Os
- **Largely defeated by large-page allocations**
- **Should fit in with idea of sliding scale for allocating large pages**

# Single-Level Release

- **Most pages only in vmm page-tables**
  - shared MPNs
  - hw-mmu: simpler, GPhys
  - sw-mmu: per-vcpu, various caches: harder
- **Modify update-approach for vmm page-tables**
  - use atomics to allow vmkernel updates
  - mark PTEs that are cached or pinned elsewhere
  - alternative: helper thread in vmm
- **Vmkernel (platform) handles invalidating PTEs**
  - flushing TLBs for running VCPUs
- **Advantages**
  - allow vmkernel to reclaim memory at will, little interaction
  - simplifies making MPN mappings not exposed (page-sharing, numa-migration)
    - leaves offline VCPUs offline, ignores VCPUs in vmx/platform
  - allows platform to block allocating VCPUs that have targets
    - not block ones with reservations
  - allows minFreePct to be reduced, possibly eliminated
- **Consider only doing for hw-mmu**
  - but then have complexity of two approaches

# Recap

- **Translation**
  - unify large-/small-page translation paths, reduce locking
  - use VmMemPlatformFlags completely
- **Invalidation**
  - switch to invalidating ranges of BPNs
  - stop breaking large pages during zap
  - measure/improve zap crosscall
- **Implement not-exposed pool**
  - reduce interactions with VMM
  - preclean dirty pages
- **Large Pages and Platform**
  - balance page-sharing and large page allocation
  - hosted support for large pages
  - prototyping 1GB pages to understand EPT costs
- **Enable vmx discontiguous mappings, bound pinning from vmx**
- **Support range between large-page backing and page-sharing**
- **Single-level release**
  - eliminate p2mUpdate, reduce minFreePct, enable alloc-prioritization, block VMs