Checkpoint migration

Ricardo Koller

Memory precopy

- Precopy Phase Trace
- Precopy Phase 0

copy the base

- Precopy Phase 1

...

- Precopy Phase n

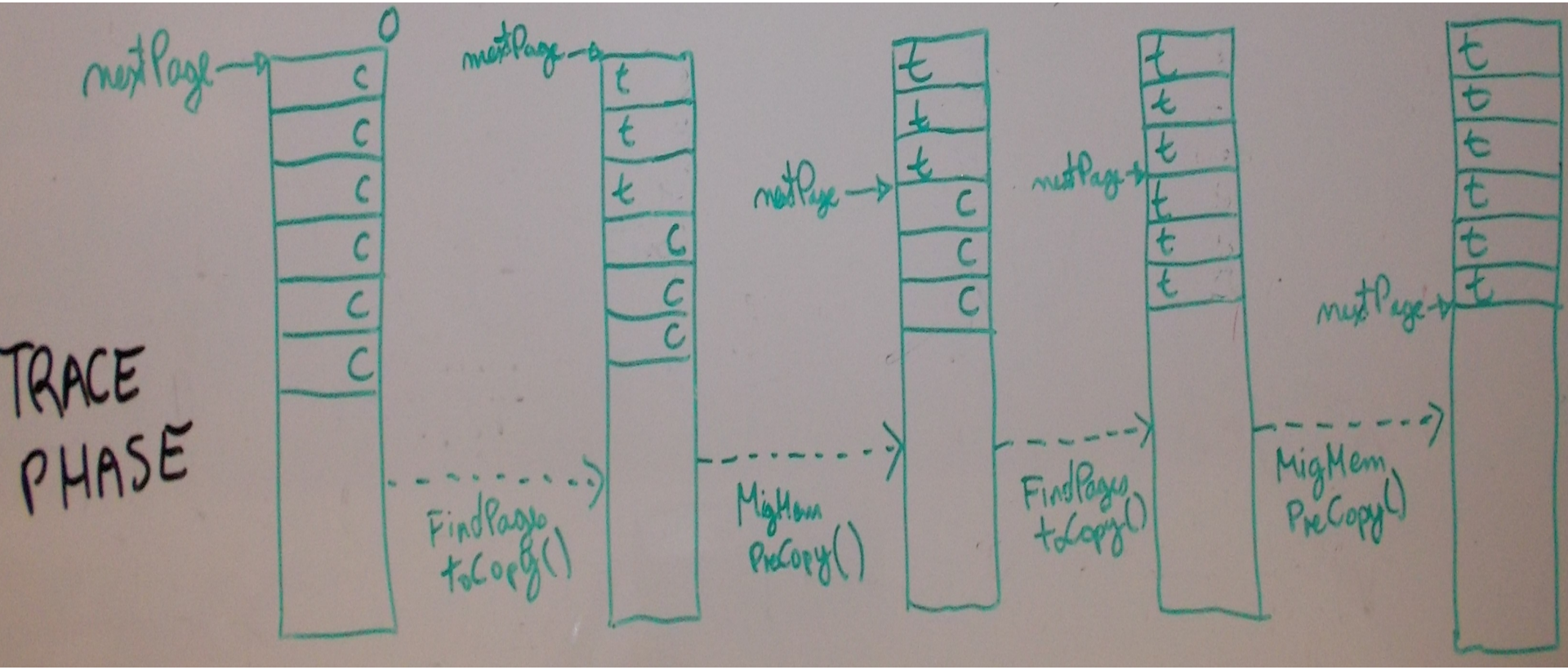
copy the deltas

```
for every phase
        for every MAX_TRACES_PER_STALL set of pages
                MigrateFindPagesToCopy // trace pages, mark pages for send
                MigratePreCopy // send pages
```

One of this per VMX:

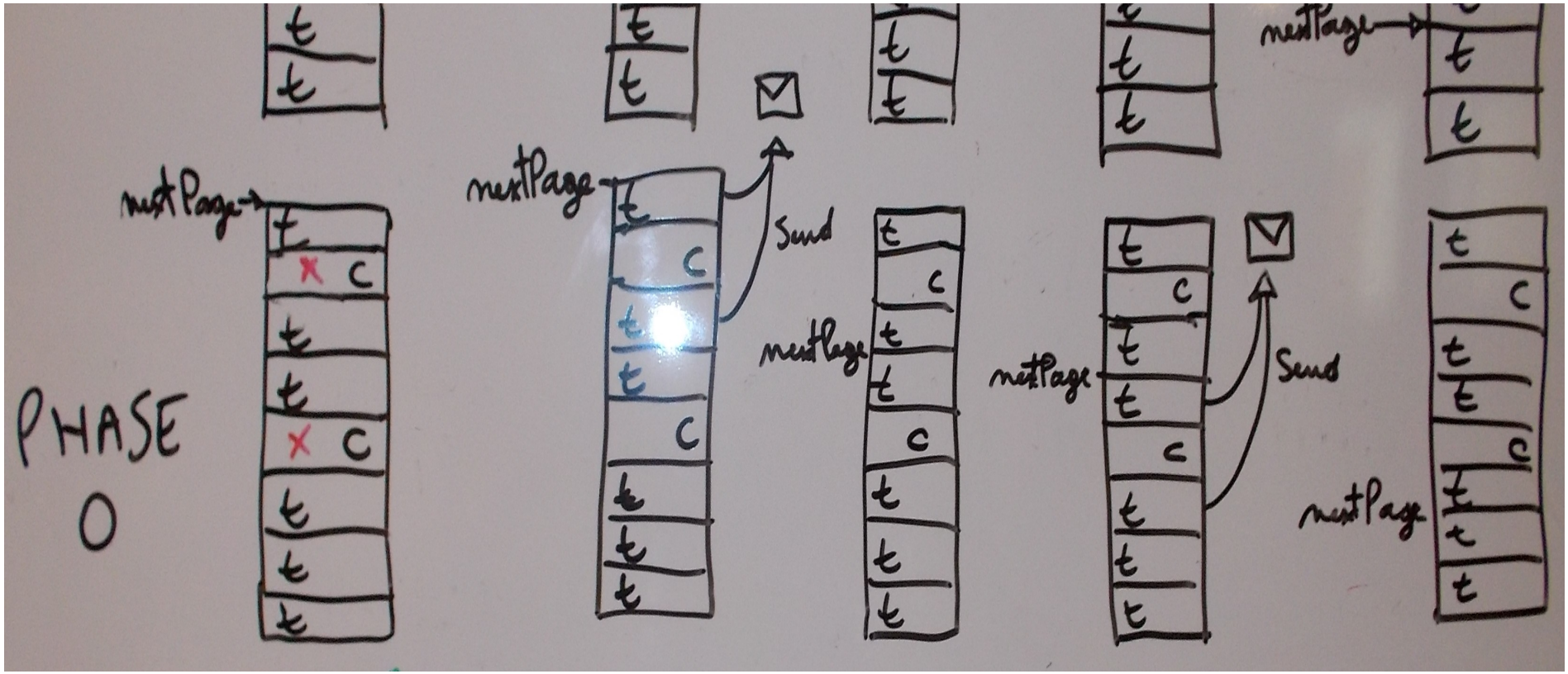
```
        struct MigratePreCopyInfo {
          uint32    phase;          // current phase
          uint32    nextPage;     // current page in the phase
          MPN64   bitmapMPN; // changed pages
        }
```

Precopy trace phase



1. mark all pages as changed
2. trace all pages
3. do not send any page to dest

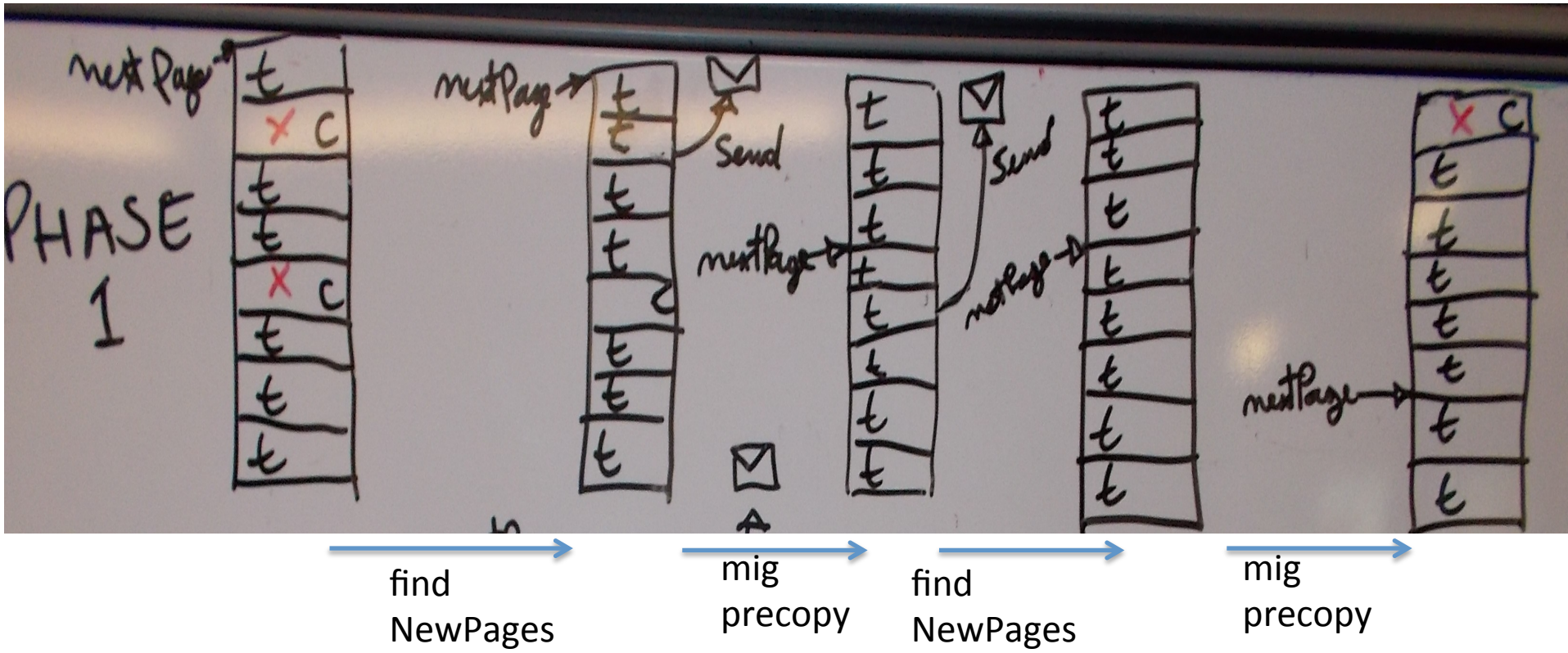
Precopy phase 0



find NewPages → mig precopy → find NewPages → mig precopy

1. send all pages that are still traced and unchanged
2. do not trace any new page

Precopy phase 1

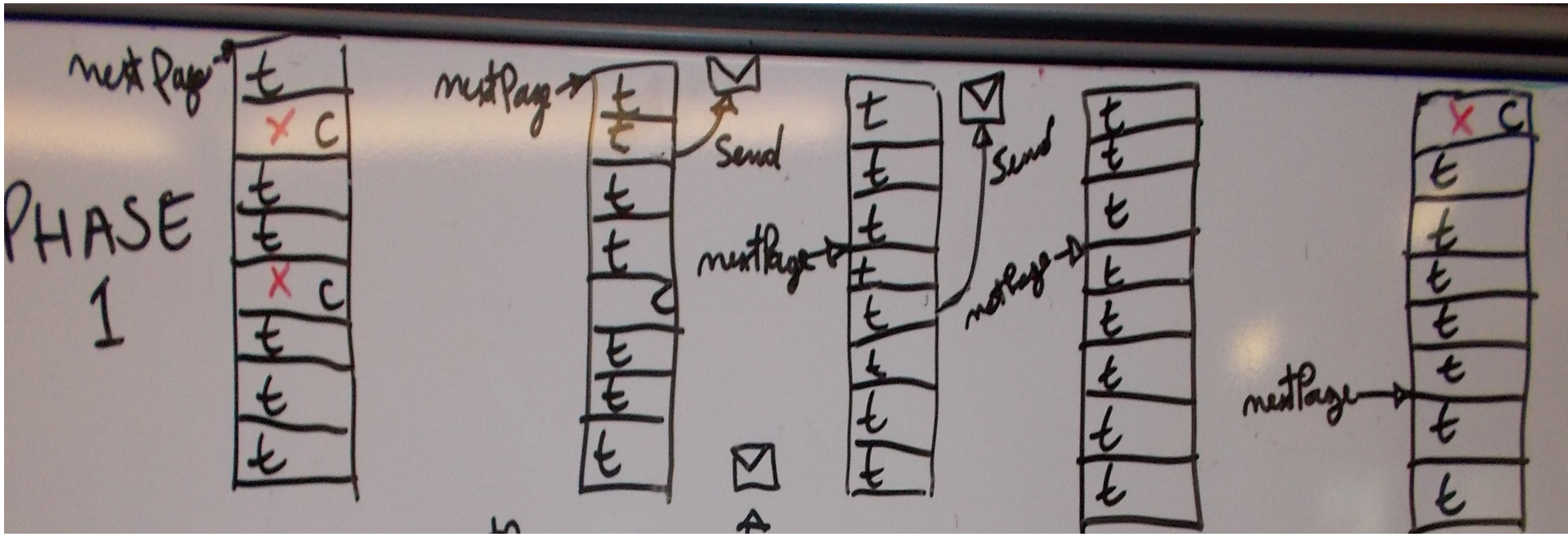


1. install missing traces (pages that changed)
2. send any changed page

Precopy from monitor POV

```
MigratePreCopyNext
    if (MigratePreCopyIterIsDone)
            VMKCall_MigrateMemPreCopyIterDone
                VMotion_MemPreCopyIterDone
    return
    MigratePreCopy
        while (vmkCanTakeMore && migPreCopyInfo.nextPage < maxPages)
            MigrateFindPagesToCopy
                BusMem_InstallTraceVectored
            VMKCall_MigrateMemPreCopy(&migPreCopyInfo.nextPage)
                VMotion_MemPreCopy(&migPreCopyInfo.nextPage)
                    // add some pages to the stream sender queue
                    Action_Post(vmmLeader, MONACTION_MIG_PRECOPY_NEXT);
```

Precopy phase 1



find NewPages → mig precopy → find NewPages → mig precopy

If converged?
PreCopyIterDone:
ask the VMX to stun the VM
and start checkpoint migrate

Precopy convergence

```
MigratePreCopyIterIsDone
    VMKCall_MigrateMemPreCopyIterDone(&anotherIteration)
        VMotion_MemPreCopyIterDone
            *anotherIteration = FALSE;
            S: Stopping pre-copy: only 1984 pages left to send, which can be sent within
                                                the switchover time goal of 0.500 seconds
            VMotion_PreCopyDone // wait a couple of slides
    if (anotherIteration)
        ...
    else
        MigrateStopPreCopyIter();
            MigrateUpdateUserlevel(MIGRATE_VMMMSG_SUSPEND_SRC, VMK_OK);
```

VMX: MigrateESX_UserRPCHandler: MIGRATE_VMMMSG_SUSPEND_SRC

VMX stun and save

1. stun the VM
2. save and send the checkpoint state

```
/*
 * checkpoint.c --
 *
 *   /----------\  Stun()  /---------------\
 *   | CPT_NONE |--------->| CPT_SAVE_SYNC |
 *   \----------/          \---------------/
 *        ^                        |   * VMM enters cont. processing loop.
 *        | UnstunCB()             |   * VMM masks deferrable actions.
 *        |                        |   * VMM and VMX devices wait for IO.
 *        |                        v
 *        |                 /-------------\
 *        |                 | CPT_QUIESCE |
 *        |                 \-------------/
 *        |                        |   * VMM waits for all actions to drain.
 *        |                        |   * Monitor actions are restricted.
 *        |                        v
 *        |                   /----------\
 *        |                   | CPT_SAVE |
 *        |                   \----------/
 *        |                        |  StunCB()
 *        |                        |   * VMX saves device state (optional).
 *        |                        |
 *        |                        v  Unstun()
 *        |               /--------------------\
 *        |               | CPT_CONTINUE_SYNC |
 *        |               \--------------------/
 *        |                        |   * VMM unrestricts montior actions.
 *        |                        |   * VMM and VMX device conitnue.
 *        |                        |
 *        \------------------------/
```

VMX stun and save

1. stun the VM

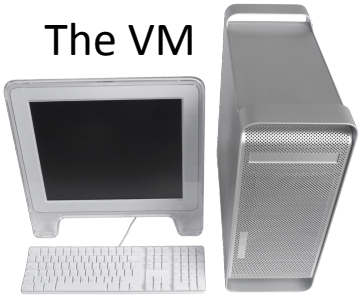
```
MigrateESX_UserRPCHandler: MIGRATE_VMMMSG_SUSPEND_SRC
    MigrateESXPreCopyToCheckpoint(&progress);
        MigrateDoSuspend()
            MigratePlatformQuiesceStart
                VMotion_AddResumeVMTimeout
            Checkpoint_Stun(MigrateStunCallback)
```

2. save and send the checkpoint state

```
MigrateStunCallback
    MigrateSave();
        CheckpointSave(CheckpointDefaultFilePath(), Migrate_Open)
        MigrateCompleteSave // end of checkpoint migration

Migrate_Open
    Dumper_SetAuxFunctions(dumper, MigrateDumperClose, ...read, ...write)
```

Checkpoint migration

The VM



CheckpointSave
for every CPT group (device)
MigrateDumperWrite
// send a cpt piece

// receive the cpt piece
// store it

checkpoint cache

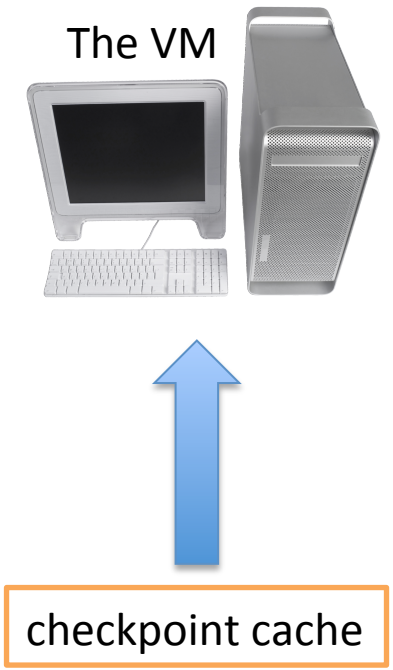
SOURCE

DESTINATION

Checkpoint migration

Later…

restore “The VM” on the dest



SOURCE

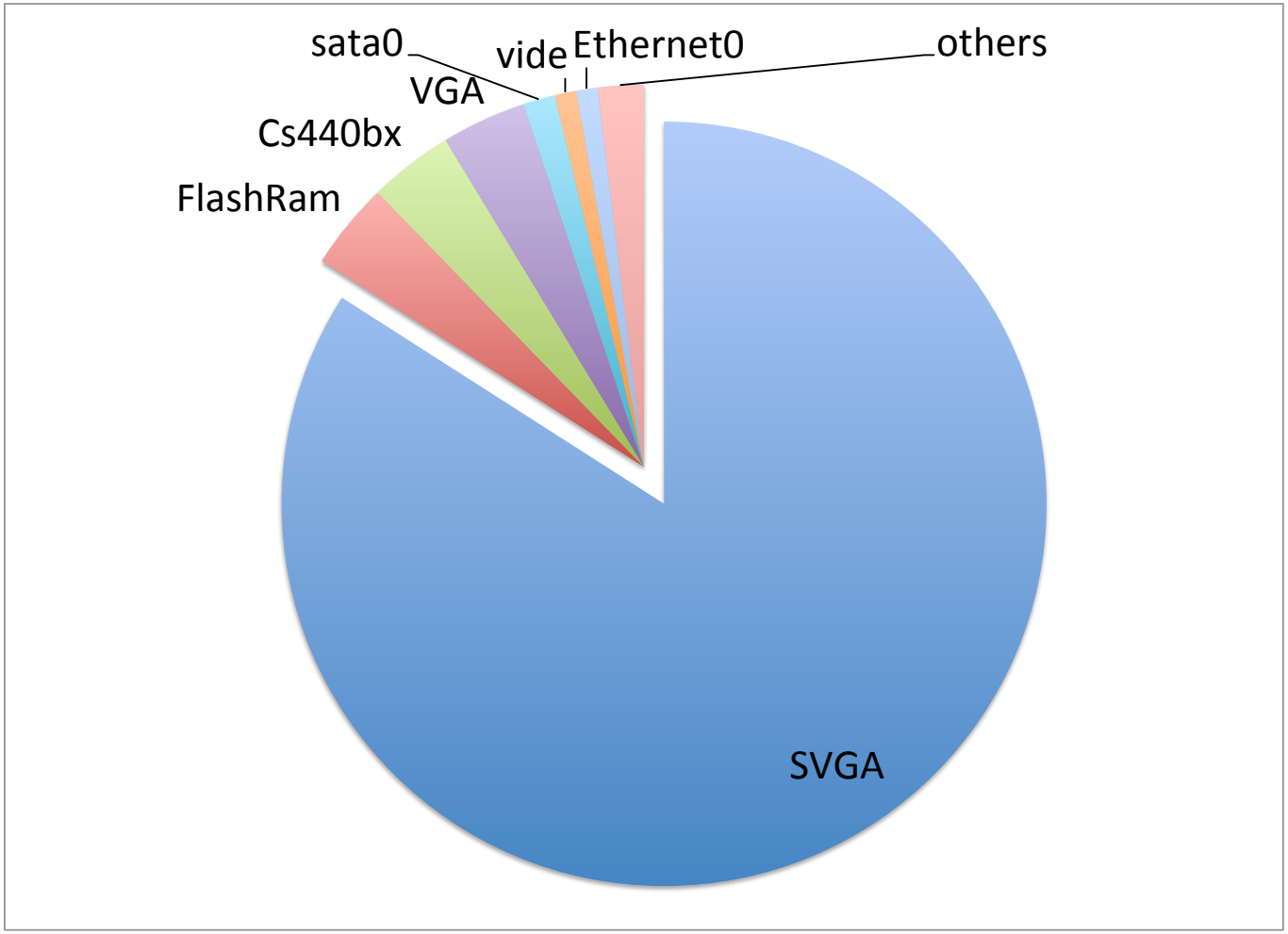
DESTINATION

CPT dump

0

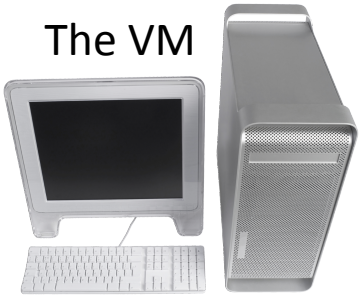
```
struct CptDumpHeader {
  uint32      id;
  uint32      version;
  uint32      numgroups;
};

struct CptGroupDesc {
  char        name[MAX_LENGTH];
  uint64      position;
  uint64      size;     // Size on disk
};

struct CptGroupDesc {
  char        name[MAX_LENGTH];
  uint64      position;
  uint64      size;     // Size on disk
};
```

Checkpoint: 7.35MB (reg. linux VM)



Migrate dumper functions

The VM



```
CheckpointSave(Migrate_Open)
    for every CPT group (device)
        MigrateDumperWrite
            // send a cpt piece
```

SOURCE

```
Migrate_Open
    Dumper_SetAuxFunctions(dumper,
        MigrateDumperClose,
        MigrateDumperRead,
        MigrateDumperWrite,
        MigrateDumperSeek,
        MigrateDumperTruncate,
        MigrateDumperError);
```

Migrate dumper write

cpt dump

SOURCE:

```
CheckpointSave: for every "piece" of CPT dump
    MigrateDumperWrite(offset, src, numBytes)
        MigrateWrite → VMKernel_MigrateWriteCptData
            VMotion_WriteCptData
                Util_CopyIn(kdata, (void*)data, size, UTIL_USERWORLD_BUFFER);
                VMotionSend_WriteCptData(vi, offset, kdata, size);
                    VMotionSendRemoteCallLocked
```

DESTINATION:

```
VMotionRecv_WriteCptData
    Migrate_CptCacheWrite
        Util_CopyIn(mi->cptCache->data + offset, data, length, bufType);
            mi->cptCache->length = MAX(mi->cptCache->length, offset + length);
```

VMX stun and save

1. stun the VM

```
MigrateESX_UserRPCHandler: MIGRATE_VMMMSG_SUSPEND_SRC
    MigrateESXPreCopyToCheckpoint(&progress);
        MigrateDoSuspend()
            MigratePlatformQuiesceStart
                VMotion_AddResumeVMTimeout
            Checkpoint_Stun(MigrateStunCallback)
```

2. save and send the checkpoint state

```
MigrateStunCallback
    MigrateSave();
        CheckpointSave(CheckpointDefaultFilePath(), Migrate_Open)
        MigrateCompleteSave // end of checkpoint migration
```

Checkpoint finish

SOURCE:

```
MigrateCompleteSave
    MigratePlatformCheckpointFinished
        VMKernel_MigrateCheckpointFinished
            VMotion_CheckpointFinished
                MigrateState_Set(vi->mi, MIGRATE_PAGEIN);
                VMotionSend_RemoteCall(VMOTION_MSG_SAVE_END)
```

DESTINATION:

```
VMotionRecv_CheckpointFinished
    vi->checkpointFinished = TRUE;
    (void) MigrateState_Set(vi->mi, MIGRATE_CPTLOAD);
    if (vi->preCopyDone)
        VMotionQueue_Add(VMotionSend_RestorePhaseBegin)
```

Precopy convergence

```
MigratePreCopyIterIsDone
    VMKCall_MigrateMemPreCopyIterDone(&anotherIteration)
        VMotion_MemPreCopyIterDone
            *anotherIteration = FALSE;
            S: Stopping pre-copy: only 1984 pages left to send, which can be sent within
                                                the switchover time goal of 0.500 seconds
            VMotion_PreCopyDone // destination sets vi->preCopyDone to TRUE
    if (anotherIteration)
        ...
    else
        MigrateStopPreCopyIter();
            MigrateUpdateUserlevel(MIGRATE_VMMMSG_SUSPEND_SRC, VMK_OK);
```

VMX: MigrateESX_UserRPCHandler: MIGRATE_VMMMSG_SUSPEND_SRC

Checkpoint finish

SOURCE:

```
MigrateCompleteSave
    MigratePlatformCheckpointFinished
        VMKernel_MigrateCheckpointFinished
            VMotion_CheckpointFinished
                MigrateState_Set(vi->mi, MIGRATE_PAGEIN);
                VMotionSend_RemoteCall(VMOTION_MSG_SAVE_END)
```

DESTINATION:

```
VMotionRecv_CheckpointFinished
    vi->checkpointFinished = TRUE;
    (void) MigrateState_Set(vi->mi, MIGRATE_CPTLOAD);
    if (vi->preCopyDone)
        VMotionQueue_Add(VMotionSend_RestorePhaseBegin)
```

RestorePhaseBegin (on dest)

- Memprecopy and checkpoint migration are Done.
- Destination can continue getting state before unstun the VM

1. VMotionSendGetDVSState // why not on cpt dump?
2. VMotionSendGetDVFilterState // DV filter == DVS filter???
3. VMotionSendGetChangeMap // get changed pages
4. VMotionSend_GetSwapBitmap // only for ESX 3 source
5. VMotionSendGetLoadHistory // some CPU stats for DRS
6. VMotionSendChecksumMemory // debug purposes
7. VMotionSendStartPageIn // get me the missing pages

VMotionSendGetDVSState (just as an example)

waits for reply

VMotionSendGetDVSState

```
VMotionSendLock(vi);
VMotionSendRemoteCallLocked(vi, VMOTION_MSG_GET_DVS_STATE, &msg)
dvsStateSize = msg.replyDataLength;

dvsState = VMotion_Alloc(vi, dvsStateSize);
MigrateNet_Read(vi->sendSocket, dvsState, dvsStateSize, -1, &bytesRead);
VMotionSendUnlock(vi);

DVS_MigrationRestore(vmmLeader, dvsState, dvsStateSize)

VMotion_Free(vi, dvsState);
```

just reads

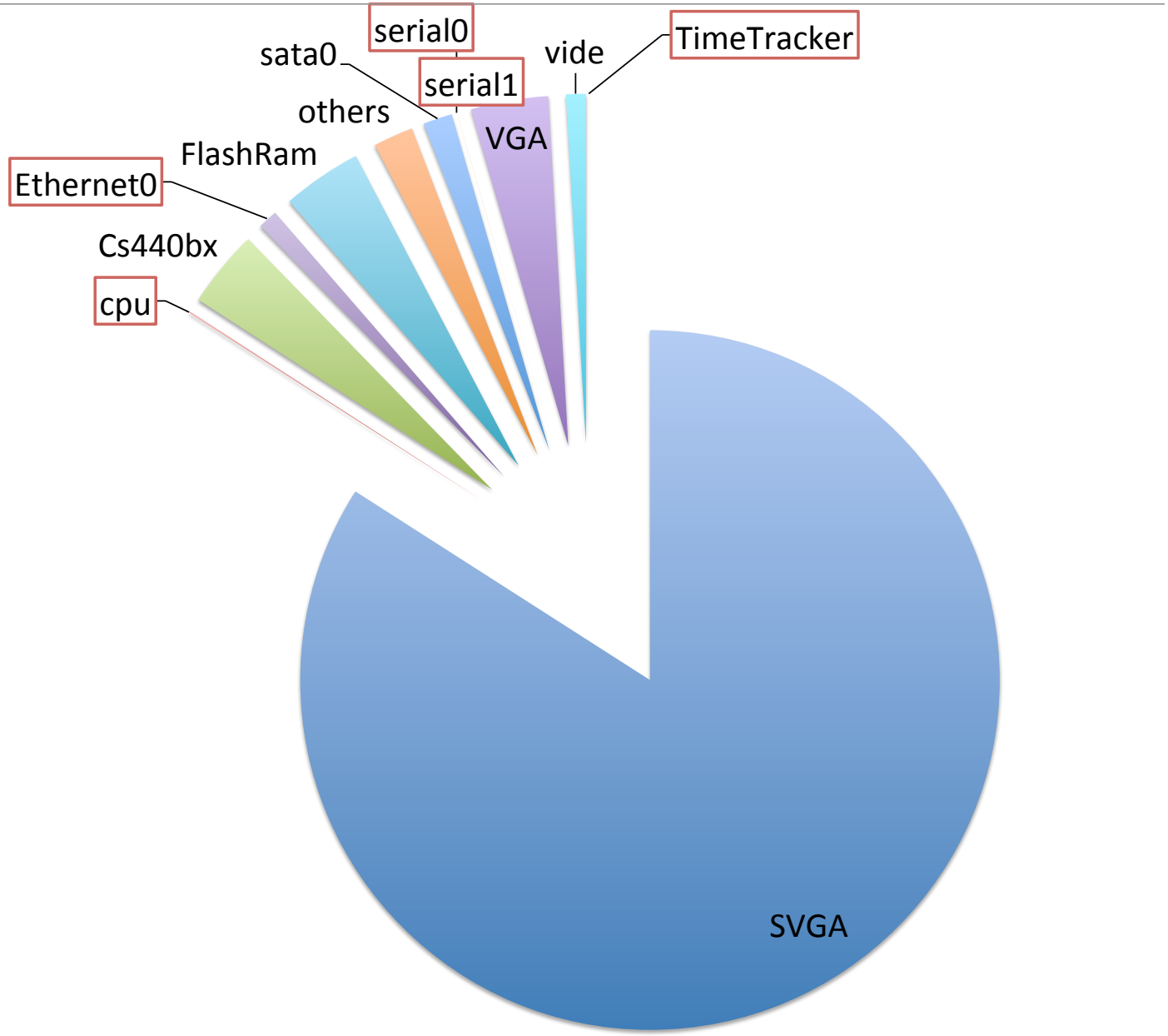
VMotionRecv_GetDVSState (source)

vmotionInt.h:

```
220   /*                    NAME,                  Function
237   VMOTION_MSG(          GET_DVS_STATE,    VMotionRecv_GetDVSState        ...)
```

VMotionRecv_GetDVSState

```
    dvsStateSize = DVS_MigrationDataSize(vmmLeader);
    dvsState = VMotion_Alloc(vi, dvsStateSize);
    DVS_MigrationCheckpoint(vmmLeader, dvsState, dvsStateSize);

    msgPair->replyData = dvsState;
    msgPair->reply.replyDataLength = dvsStateSize;
    msgPair->freeReplyData = TRUE;

    // msgPair because it has msg INcoming and msg OUTgoing
```


serial0
sata0
serial1
vide
TimeTracker
others
VGA
FlashRam
Ethernet0
Cs440bx
cpu
SVGA

Checkpoint: 1.26MB (small VM)