

**ORACLE®**

# Brian Hirano

Virtual Operating System Group  
Server Technologies

# Oracle on Itanium®

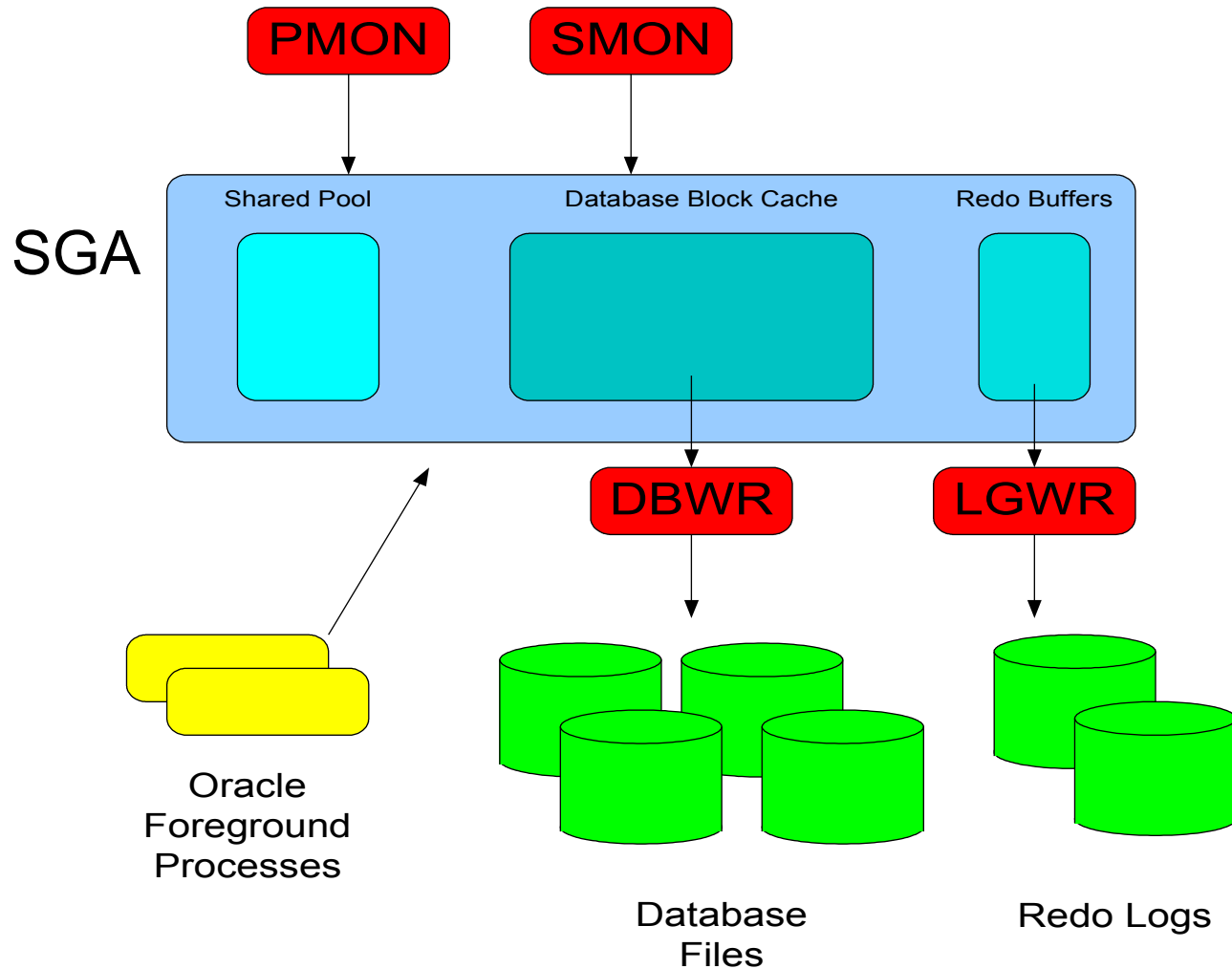
# Overview

- Oracle Database
- Oracle relationship with hardware vendors
- Intel-Oracle Relationship
- Initial Oracle Benchmark efforts on Itanium
- Current state of performance efforts
- Observations and ruminations

# Oracle Database

- Oracle Instance
  - Set of background processes
  - Processes communicate through shared memory
  - Foreground “shadow” processes
- Oracle Files
  - Database Files
  - Redo Logs
  - Control Files

# Oracle Instance



# Oracle and Hardware Vendors

- Operating Systems:
  - Large memory support: shared and private
  - I/O: synchronous, asynchronous, list (if available)
  - Scheduling with minimal preemption
  - Efficient timed-wait interfaces
- Compilers
  - Profile-guided optimizations
  - Efficient addressing modes
  - Intrinsic/asms for Oracle-specific annotations

# Oracle and Hardware Vendors

- Hardware systems
  - Balanced systems: good memory and I/O bandwidth
  - Support lots of memory
  - NUMA for scalability
- Microprocessors
  - Large caches to hold code and shared read-only data
  - Support for large address spaces
  - Low cache latency
- Make sure Oracle workloads used during design of all components

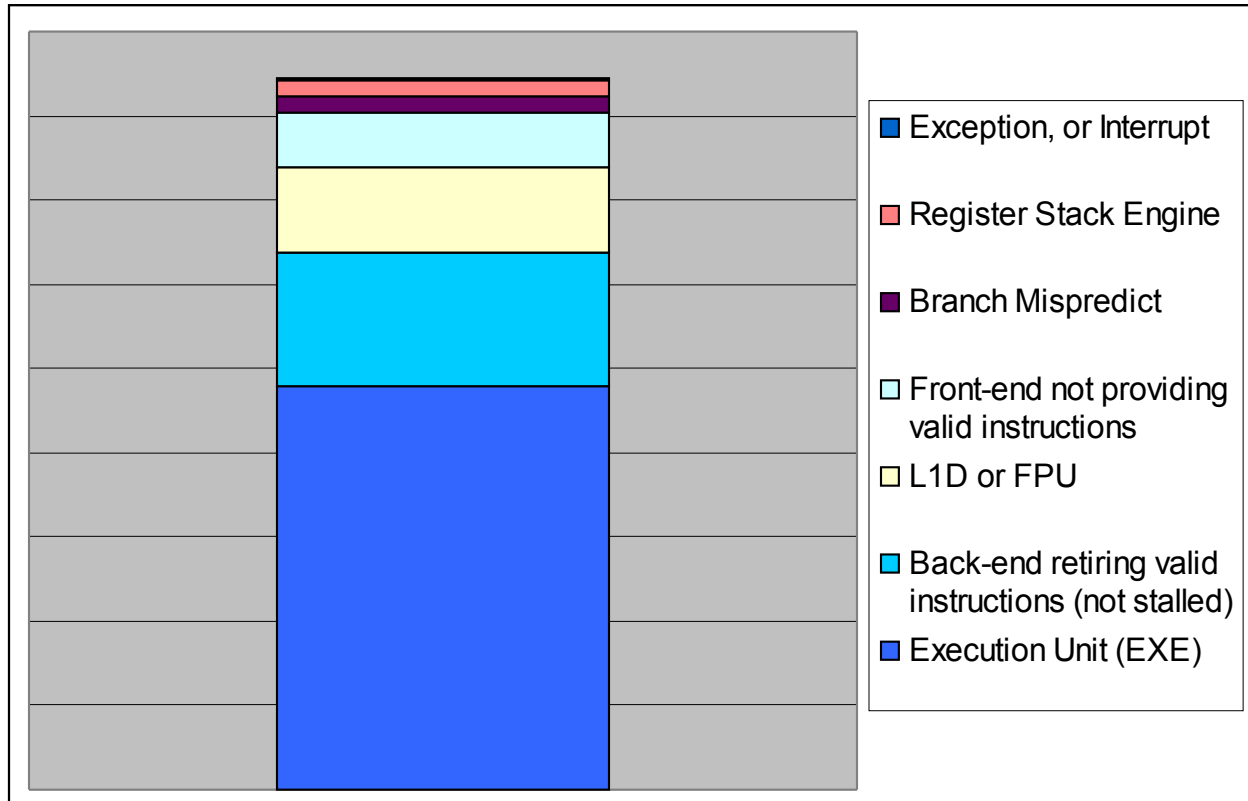
# Oracle and Itanium®

- Received Itanium documentation
  - Large page support
  - PA-RISC-like TLB support (for single-address space system)
  - Protection Keys
  - Acquire/Release Semantics
- 2000: Intel approached Oracle with Itanium® development environments

# Oracle Itanium®

- TPC-C Benchmark Oracle on Itanium®
  - Linux
  - HP-UX
  - Windows
- Oracle works on codepath reduction
- Oracle works with HP and Intel on compilers, operating systems, and system performance

# Oracle CPI on Itanium<sup>®</sup> 2 Linux



Source: Intel



# Distribution of Data Stalls

Address Range	Memory Type	Events	Total Clocks	Avg Latency	% Clocks
0x00000000xxx	VHPT tables	656	153,642	234	1.01%
0x00000044xxx	Fixed SGA	584	119,706	205	0.79%
0x2xxx	Library Space	2,534	621,692	245	4.11%
0x400000xxx	Text/ROData	3,143	762,872	243	5.04%
0x600000xxx	RW Data	14,250	3,459,832	243	22.85%
0x60000fff8xxx	RSE backing store	5,391	1,422,722	264	9.40%
0x60000fffffxxx	User Stack	3,966	961,267	242	6.35%
0x8000xxxx	Oracle SGA	21,926	4,910,574	224	32.43%
0xa000xxxx	Kernel:Drivers	186	45,873	247	0.30%
0xc000xxxx	PCI Space	289	248,457	860	1.64%
0xe000xxxx	Kernel Space	11,023	2,436,747	221	16.09%
<b>Total</b>		<b>63,948</b>	<b>15,143,384</b>	<b>237</b>	

38.59%

Stack Depth                   188,176  
 RSE Depth                     7,864

# Characterizing Oracle TPC-C Behavior

- Extremely “flat” profile
  - Improving one function contributes very little to overall throughput
  - TPC-C improvements extremely difficult
- Large code footprint (exceeds McKinley/Madison L2)
- Data stalls predominate
- Little contended data

# What Tools are Available?

- Oracle internal tools for counting instructions: ITM/ITA
- From Intel
  - VTune
  - EMON
  - Home-grown Data EAR (event address range) tools
- Capabilities of the Itanium PMU extremely helpful to this effort

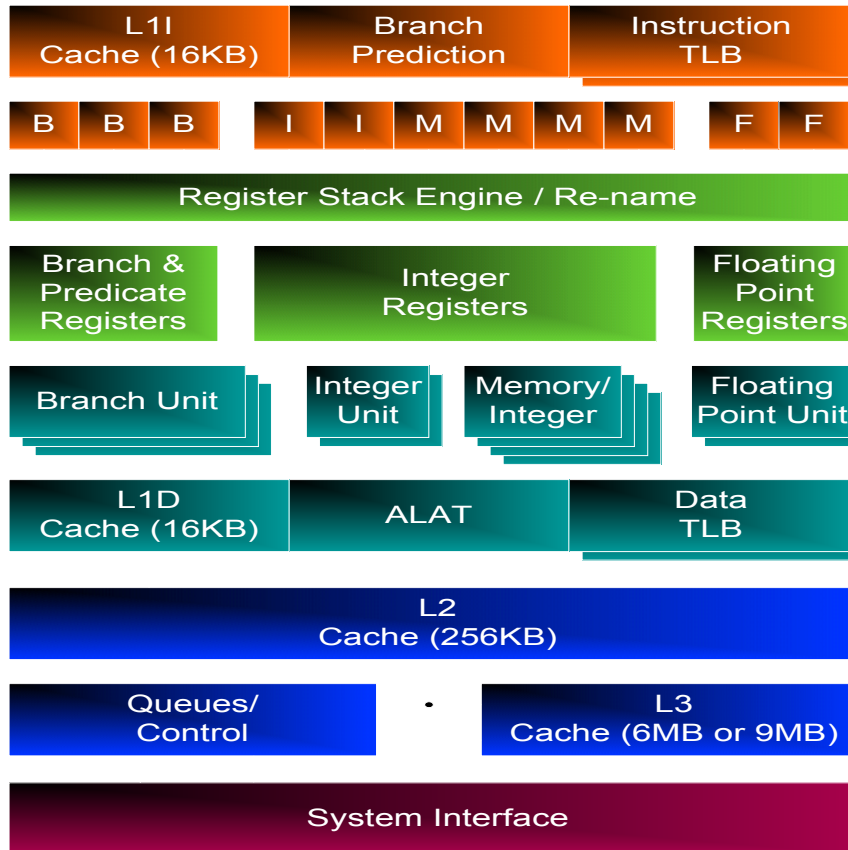
# Oracle and Itanium® Performance

- Work with Intel on compilers for Oracle on IA-64
  - Compilers: Mark Davis presentation, Gelato Fall 2005.
- Encourage OS improvements for Oracle on IA-64 and implement operating system-dependent routines to use new OS interfaces
  - Ken Chen's presentation today at 1:30pm
- Improve runtime of important C runtime library functions
- Begin to examine data latency

# Examples of Oracle's work on Itanium®

- Use of non-temporal stores for string functions (e.g., memcpy(), memset())
- Implement Oracle traceback functions for diagnostic output
- Reduce unnecessary memory fences (OzQ pressure)
- Data EAR Data
- Using Data EAR for prefetching

# Itanium<sup>®</sup> 2 Block Diagram



Source: Intel

# Memory Fences in Oracle

- Oracle Store Ordering
  - Certain stores need to be ordered for multiprocessor consistency
  - Certain stores need to occur in program order; stores completed in order after a process dies
  - MP synchronization: operations usually ordered using Oracle-implemented mutexes (Oracle latches)
- Oracle Memory ordering macros
  - Styled after SPARC fences
  - Also has “compiler barrier” semantics: compiler treats annotation as

# Oracle and Memory Fences

- Memory fences should be uncommon since normal concurrent operations
- Oracle told memory fences cause OzQ to slow down
- Find and remove unnecessary memory fences

# Eliminating Unnecessary Memory Fences

```
{
kssrc *rcv;
/* log operation */
rcv = ksugsr(ksugcp());

/* get recovery area */
rcv->kssrcdst = parent;
rcv->kssrcobj = so;
/* note kssrcobj is always set
   last, cleared first */
SL_WRITE_BARRIER();

/* rest of the function */
}
```

```
{
volatile kssrc *rcv;
/* log operation */
rcv = ksugsr(ksugcp());

/* get recovery area */
rcv->kssrcdst = parent;
rcv->kssrcobj = so;

/* no need for write ordering;
   volatile guarantees program
   ordered stores.

   */
/* rest of the function */
}
```

# Data EAR for Buffer Cache Function

```
40000000002c6e90:      [MMI]  adds r69=16,r25;;kcbhb->hb_kcbhb1t
                               (reuse r25)
40000000002c6e96:      ld8 r23=[r25]  kcbhb1t.hb_kcbhb1t.kggnxt
40000000002c6e9c:      adds r88=8,r69;; & latch level
[0.02%] 40000000002c6ea0:      [MII]  ld4.s r21=[r88]  ld latch level level
40000000002c6ea6:      shladd r70=r24,4,r23;; corr. ksglk entry
40000000002c6eac:      sxt4 r20=r21  sign-extend latch level
40000000002c6eb0:      [MMI]  ld8 r22=[r70];; read ksglk->kggnxt
[9.50%] 40000000002c6ed0:      [MII]  mov r118=r69
40000000002c6ed6:      adds r119=176,r9 ; ksupr->ksuprlki;
40000000002c6edc:      shladd r19=r20,1,r2 ;
                               &ks11vl[latch->ks11tlvl]
[10.26%] 40000000002c6ee0:      [MMF]  chk.s.m r21,40000000002ce4c0 <.b7_1638>
40000000002c6ee6:
```

# What do we do with Data EAR?

- Co-locate commonly accessed data: globals
- Reorder/regroup fields in a structure
- Can non-temporal stores help?
- Attempt to use data for prefetching

# Data Prefetching

- Data prefetching for TPC-C is hard
  - Published reports of 8% throughput increase due to data prefetching
- Data prefetching on IPF is harder
  - Id.a, Id.s, etc., is used extensively
- Data prefetching on IPF with prefetching compilers is even harder
- Over-prefetching has negative effects

# Data Prefetching for Oracle function: `kpobii()`

- Function responsible for “dispatching” bind types (e.g., bind variables for SQL statements).
- Third highest function with respect to total time spent

# kpobii() prologue

```
STATICF
void kpobii(hst, xsc, ctx, fr)
hstdef  *hst;
kxsc    *xsc;
ctxdef  *ctx;
kksfr   *fr;
{
    ctxbbs  *curpg  = ctx->ctxibcs;
    ctxbb   *curbbc = curpg->elem_ctxbbs;
    kxsbctx *ictx   = &xsc->kxscbctx;
    ub4     bpos    = ictx->pos_kxsbctx;
    uacdef  *uac    = ictx->uac_kxsbctx;
    ctxmut  *ctxm   = ictx->ctxm_kxsbctx;

    kxsbb   *bb     = ictx->bb_kxsbctx;
    kgsmp   *gp     = ksmgpga;
```

# Switch Statement in kpobii()

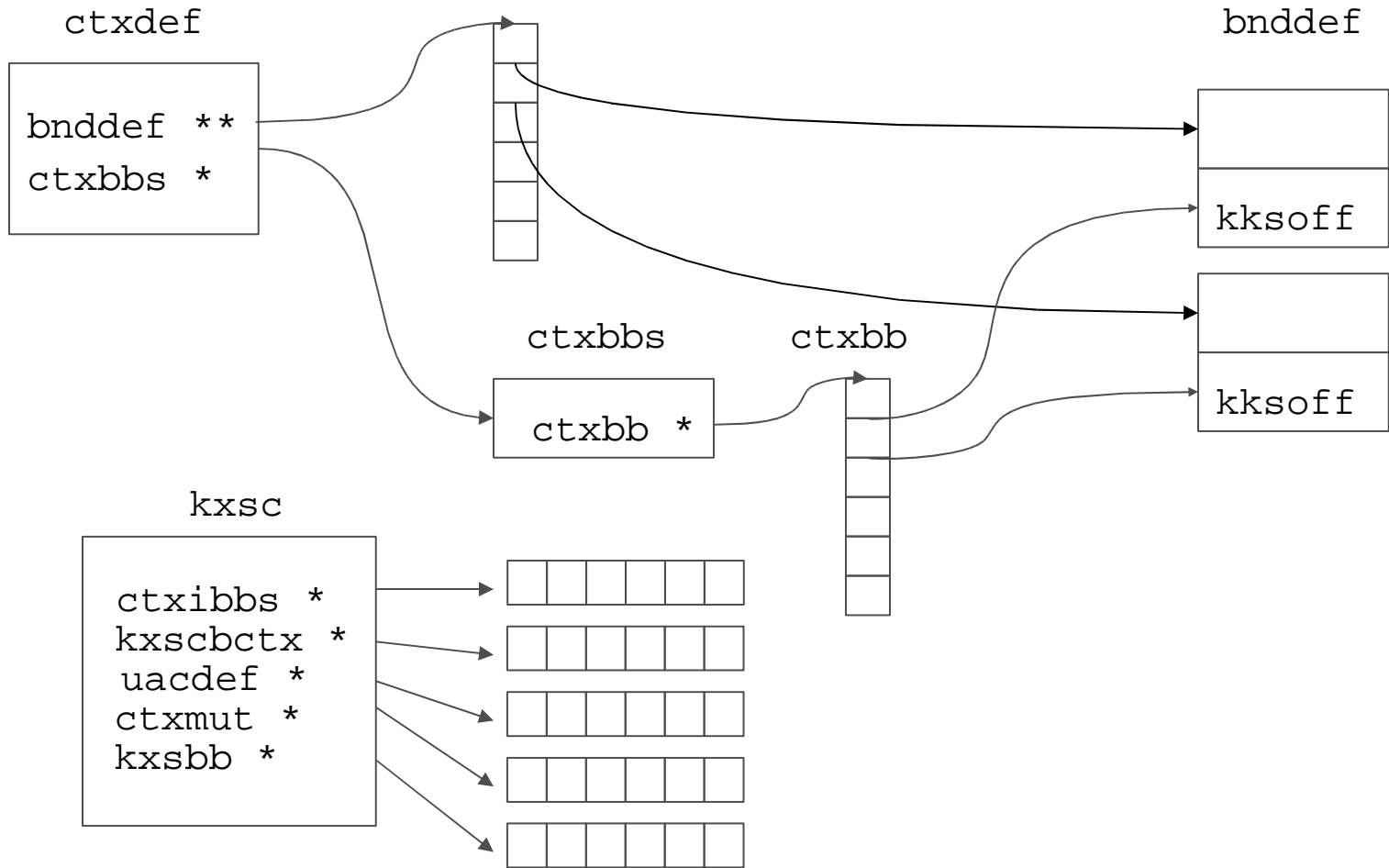
```
switch (curbbc->opc_ctxbb)
{
    case CTXBG_IBRSSINNNS:
    {
        /* The values need to be read only the first time */
        if (ctxm->ctxitn == 1)
        {
            opnmut    *opm;

            /* The bind buffers need not be populated */

            /* Assign the opndef appropriately */
            opm = (opnmut *)kkscpp(fr, curbbc->bb2_ctxbb.opnoff_ctxbb2);
            opm->opnbfpr = (ub1 *)KPRCGBFP0(uac);
            opm->opncvl = (b2)KPRCGVLN0(uac);
            opm->opnmflg = OPNFAE;
        }

        /* Make appropriate interpreter context changes */
        bpos++;
        bb++;
        uac++;
        curbbc++;
        break;
    }
}
```

# Structure Relationships



# Prefetching Results

- Pipeline prefetches across four stages
- Started prefetches three function calls up from `kpobii()`
- Conditionally included prefetches within `kpobii()` loop for arrays accessed from `kxsc`
- Saw a 0.5% reduction in net time spent from `kpobii()` on a cached run.
- WAG target: 1% reduction

# Results: November 2002

- Oracle achieved 80,494 tpm-C running on Red Hat Linux Advanced Server
- Achieved 80,570 tpm-C on HP-UX
- HP Server rx5670
  - 4 Itanium® 2 1-GHz Processors with 3MB L3 Cache
- At the time, highest TPC-C number on a four processor system

# Current Work

- Oracle and Intel continue work on Itanium®
- Continue to investigate data stalls
- Work on scalability (e.g., NUMA)
- Investigate RSE traffic
  - Oracle has large functions
  - Oracle has functions with lots of arguments
  - Oracle has functions with many automatics
- Work on Montecito
- Continued codepath reduction

# Montecito



Source: Intel

# Microarchitectural Features to Mitigate Memory Latency

- Large caches
- Shared caches effective for caching code and shared data
- Effects of increased use of speculation?
- HT: threads overlap cache misses
- “Execute ahead” techniques
- Caveat: OLTP code behavior makes this difficult

# Need Better Tools

- Need performance measurement tools other than ITA
- Require program behavior tools
  - Correlate CPI to call graph
- Require detailed analysis tools
  - Associate data latency with C source lines
  - Associate data latency with C type
- Tools to visualize data access patterns and data dependencies
  - Pin and Pin-based tools

# Oracle and Protecting Memory

- Database block cache (buffer cache) important
- Stray pointers or memory corruptions may persist on disk
- Oracle parameter `_db_block_cache_protect`
  - Buffer cache in protected no-access
  - Read or modify buffer, buffer is changed to read or read-write, respectively
  - Re-protect buffer no-access after operation

# Problems with Oracle and Memory Protection

- Does not work for threads
- High overhead mode – not usable in production environments
- Page protection on a per-process basis
  - If buffer cache memory used for something else instance broadcast an “unprotect address range” message
  - If buffer cache gets new memory, instance broadcasts “protect address range” message

# Protection Keys

- Associate protection key with address range
- Thread of execution acquires protection key prior to operating within address range
- No “broadcast” necessary
- Buffer cache memory protected by a key.
- If buffer cache relinquishes memory, change protection key to default mode.
- If buffer cache acquires memory, associate memory with buffer cache protection key

# Protection Keys

- Use keys to protect different components or subsystems
- Allow lower access modes to untrusted code
- Fault resilient
- How many protection keys?