Virtual Memory Simulation Theorem

Mark A. Hillebrand, Wolfgang J. Paul

{mah,wjp}@cs.uni-sb.de

Saarland University, Saarbruecken, Germa

This work
was partially
supported by

Overview

Theorem: the parallel user programs of a
see a sequentially consistent virtual share
(Correctness of main frame hardware & p

Context

A (practical) approach for the complete fo
verification of real computer systems:

1. Specify (precisely)
2. Construct (completely)
3. Mathematical correctness proof
4. Check correctness proof by compute
5. Automate approach (partially; recall

Example: Processor design

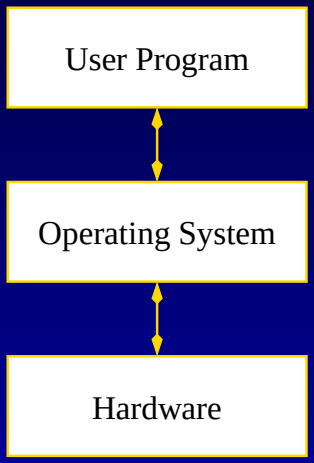
1.–3. [MP00]
Computer Architecture: Complexity and Correctness
Springer

4. [BJKLP03]
Functional verification of the VAMP
Charme ’03

5. PhD Thesis Project S. Tverdyshev
(Khabarovsk State Technical Univers

Why Memory Management

Layers of computer systems (all using loc
computation and communication):



! In memory management hardware a
are coupled extremely tightly.

DLX Configuration

A processor configuration of the DLX is $c = (R, M)$:

- $R : \{\text{register names}\} \rightarrow \{0,1\}^{32}$
  where register names $PC, GPR(r), s$
- $M : \{\text{memory addresses}\} \rightarrow \{0,1\}^{8}$
  where memory addresses $\in \{0,1\}^{32}$

Standard definition is an abstraction:
real hardware usually has no 2^{32} bytes ma

DLX_V Configuration

A *virtual* processor configuration of DLX
$c = (R, M, r)$:

- $R : \{\text{register names}\} \rightarrow \{0,1\}^{32}$
  where register names: PC, $GPR(r)$,
- $M : \{\text{virtual memory addresses}\} \rightarrow$
  where virtual memory addresses $\in \{$
- $r : \{\text{virtual memory addresses}\} \rightarrow$
  where the rights R (read) and W (writ
  identical for each page ($4K$).

! DLX_V is a basis for user programs.

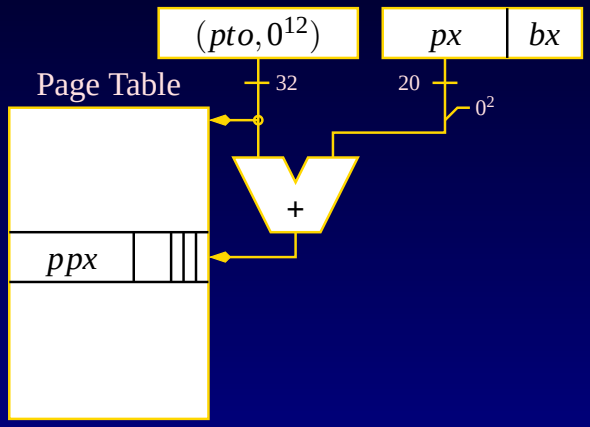
DLX_S Configuration

A real specification machine configuratio
is a triple $c_S = (R_S, PM, SM)$:

- $R_S \setminus R$:
  - $mode$ system mode (0) or user m
  - pto Page table origin
  - ptl Page table length (only for
- PM physical memory
- SM swap memory

! DLX_S is hardware specification.

Page-Table Lookup



Let $c = (R_S, PM$

- Virtual addr
  $va = (px, b$

px: page in

bx: byte in

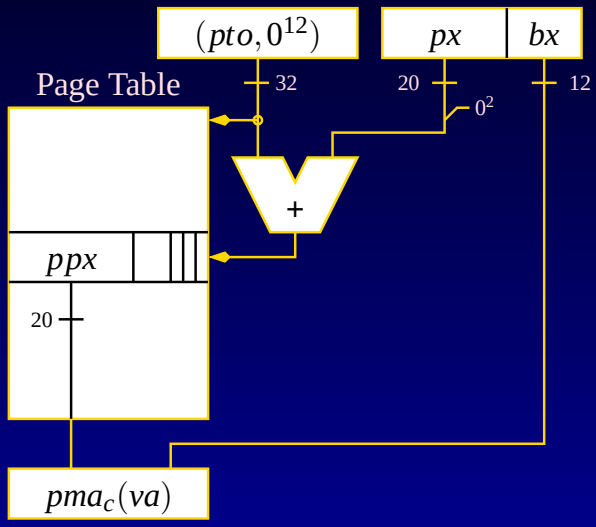
- $PT_c(px) = PM_4(\langle pto \rangle + 4 \cdot \langle px \rangle)$

$=$

$ppx_c(va)$: physical page index

$v_c(va)$: valid bit ($\leftrightarrow$ page in PM)

Address Translation



Let $c = (R_S, PM$

- Virtual addr
  $va = (px, b$

px: page in

bx: byte in

- $pma_c(va) = (ppx_c(va), bx)$
  pma_c: physical memory address
- To access swap memory, we also def
  sma_c: swap memory address (e.g. sb

Instruction Execution

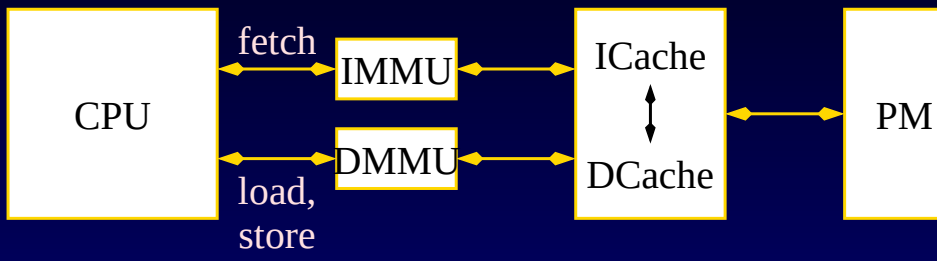
DLX_V uses virtual addresses:

- Fetch: $va = DPC$ (delayed PC)
- Effective address of load/store: $va = ea = GPR(RS1) + imm$ (regi

Hardware DLX_S for $mode = 1$ (user):

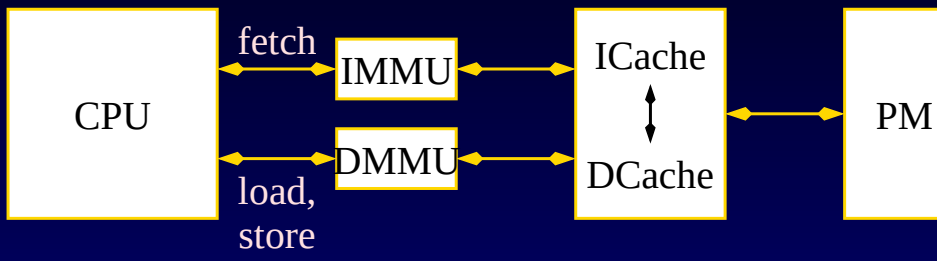
- If $v_c(va)$, use translated addresses pn instead of va.
- Otherwise, exception. (hardware supplies parameters for pa handler)

Hardware Implementation



- Build 2 hardware boxes MMU (mem
  management unit for fetch and load/s
  between CPU and caches
- Show it translates
- Done

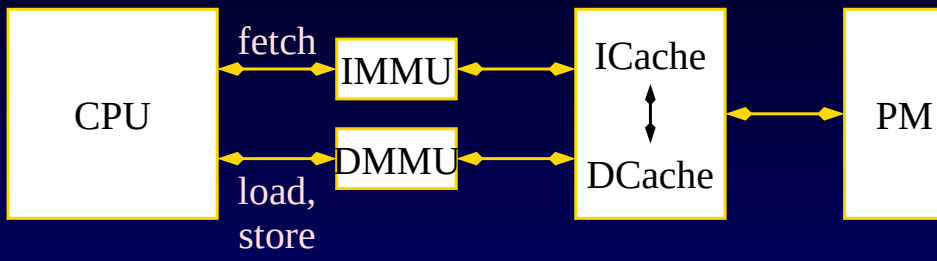
Hardware Implementation



- Build 2 hardware boxes MMU (mem
  management unit for fetch and load/s
  between CPU and caches
- Show it translates
- Done

No!

Hardware Implementation



- Build hardware boxes MMU & a few
- Identify software conditions
- Show MMU translates if software co
  met
- Show software meets conditions
- Almost done

We do not care about translation (purely t
we care about a simulation theorem.

Simulating DLX_V by DLX_S

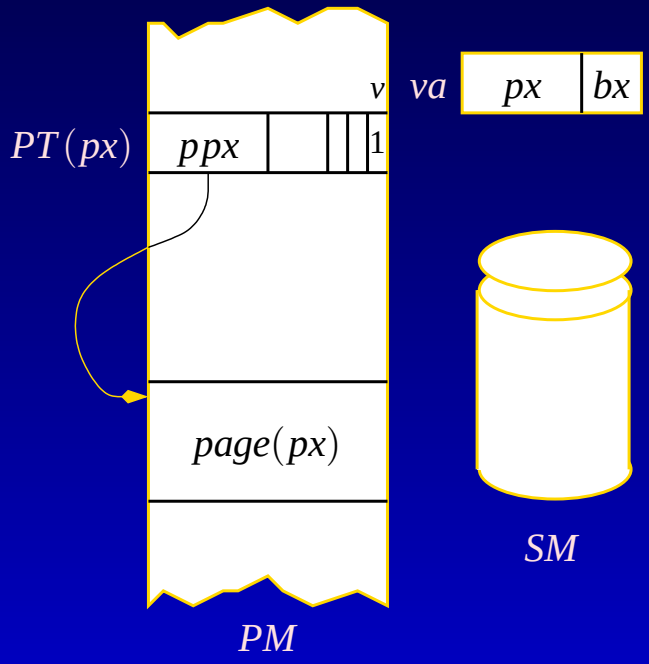
Let $c = (R_S, PM, SM)$ and $c_V = (R_V, P$
Define a projection: $c_V = \Pi(c)$

- Identical register contents: $R_V(r) =$
- Rights in page table:

$$\mathrm{R} \in r(va) \Leftrightarrow r_c(va) = 1$$
$$\mathrm{W} \in r(va) \Leftrightarrow w_c(va) = 1$$

Simulating DLX_V by DLX_S

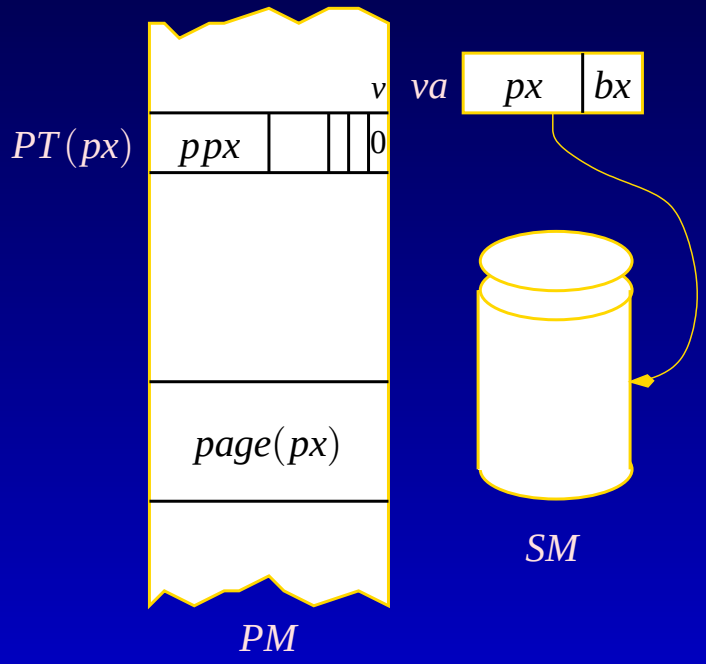
$$VM(va) = \begin{cases} PM(pma_c(va)) & \text{if } v_c(va) \\ SM(sma_c(va)) & \text{otherwise} \end{cases}$$



PM cache for virtual memory, PT is cache
Handlers (almost!) work accordingly (sel
write back to SM, swap in from SM)

Simulating DLX_V by DLX_S

$$VM(va) = \begin{cases} PM(pma_c(va) & \text{if } v_c(va) \\ SM(sma_c(va) & \text{otherwise} \end{cases}$$



PM cache for virtual memory, PT is cache
Handlers (almost!) work accordingly (sel
write back to SM, swap in from SM)

Software Conditions

1. OS code and data structures (PT, $sbas$ free space) maintained in system ar $Sys \subseteq PM$
2. OS does not destroy its code & data
3. User program (UP) cannot modify Sy (impossibility of hacking)
4. Writes to code section are separated f code section by `sync` or (syncing) r

Standard `sync` empties pipelined or OoO order) machine before next instruction is

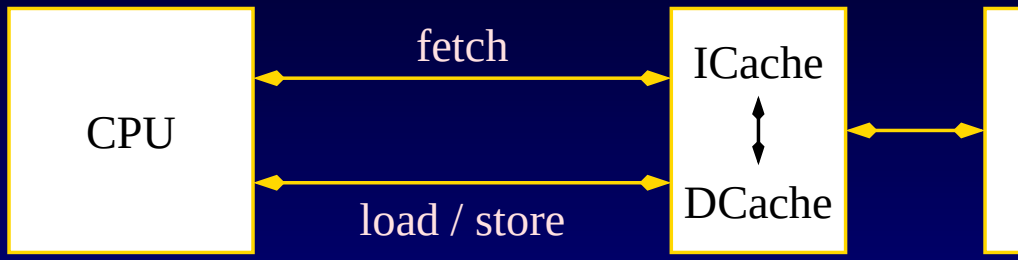
! Swap in code then user mode fetch
= self modification of code by OS & U

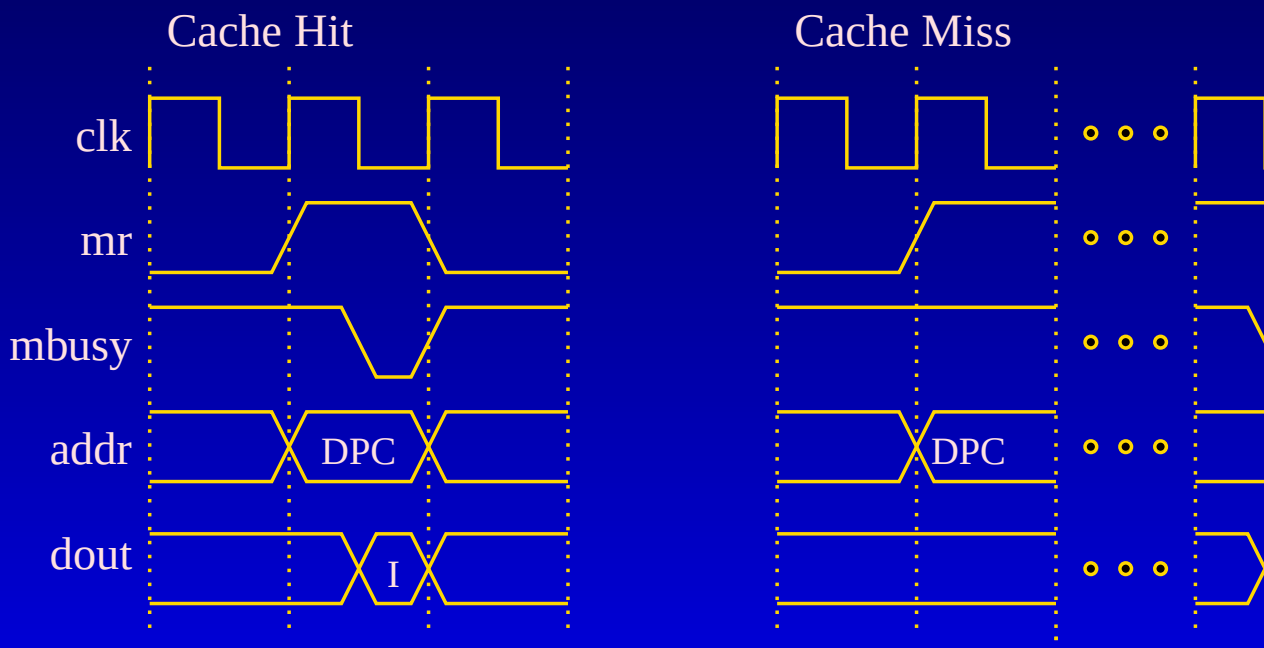
Guaranteeing Software Con

1. Operating system construction
2. Operating system construction
3. No pages of Sys allocated via PT to
4. UP alone not self modifying, handler `rfe`

Hardware I

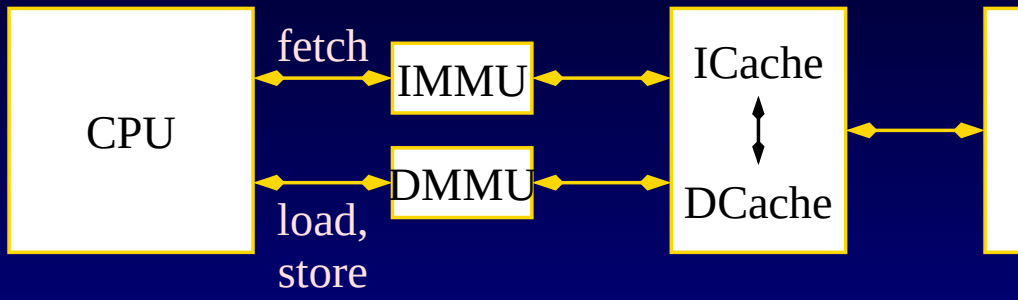
CPU – memory system – protocol





Hardware II

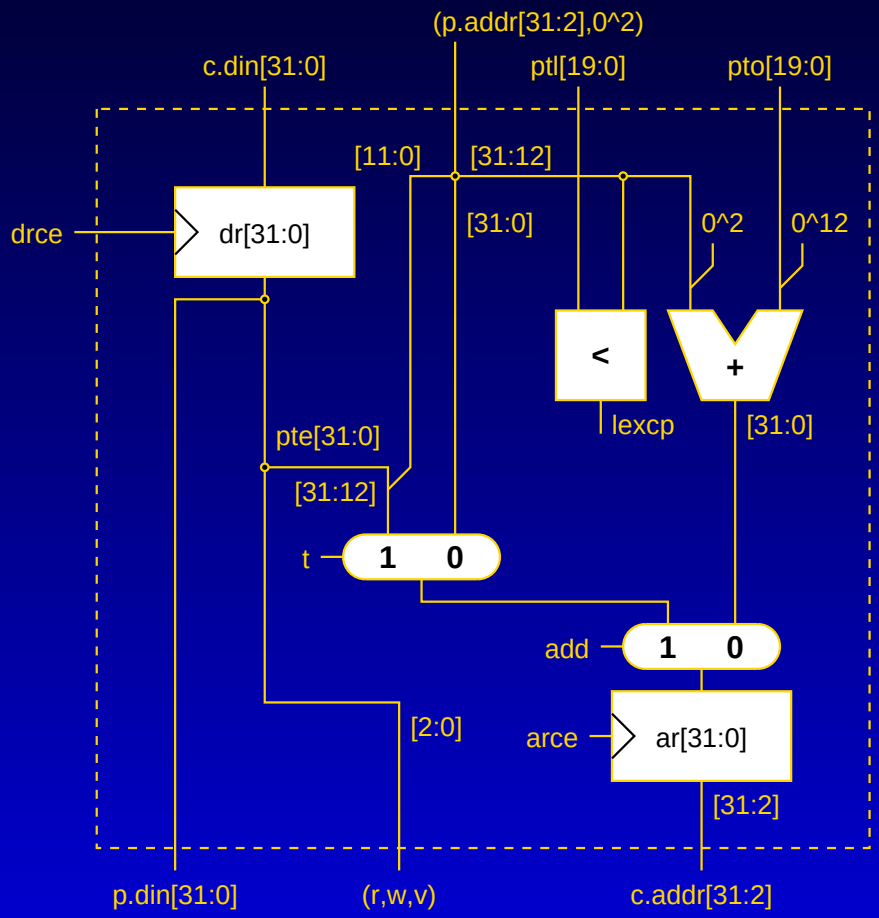
Inserting 2 MMUs:

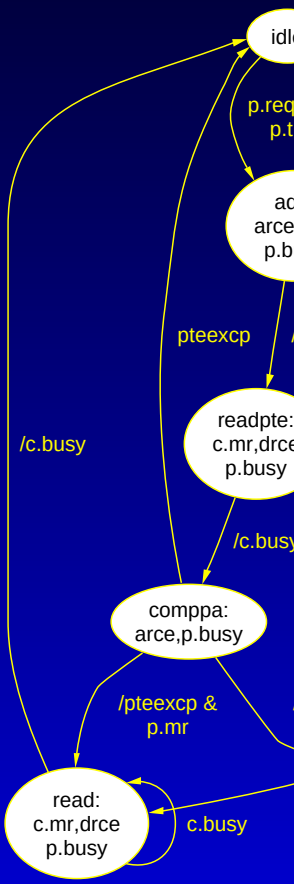


Must obey memory protocol at both sides

Primitive MMU

Primitive MMU controlled by finite state





MMU Correctness

Local translation lemma:
Let T and V denote the start and end of a
read request, no excp. Let $t \in \{T, \ldots, V$
Hypothesis: the following 4 groups of inp
change in cycles t (i.e. $X^t = X^T$):

G_0 : $va = p.addr^t, p.rd^t, p.wr^t, p.req^t$
G_1 : $pto^t, ptl^t, mode^t$
G_2 : PT^t
G_3 : $PM^t(pma^t(va)$

Claim: $p.din^V = PM^T(pma^T(va))$
Proof: plain hardware correctness

Guaranteeing Hypotheses G

- G_0 MMU keeps $p.busy$ active during tra
- G_1 Extra gates: normal `sync` before issu enough. If `rfe` or update to $\{mode,$ issue stage, *stop* translation of fetch o instruction.
- G_3 User program cannot modify Sys. Pr system code terminated (by `sync`)
- G_4 Fetch: correct by `sync`
  Load: assumes non-pipelined, in-ord unit, extra arguments otherwise

Global Hardware Correctne

Define scheduling functions $I(k, T) = i$:
I_i is in stage k during cycle T iff (...)

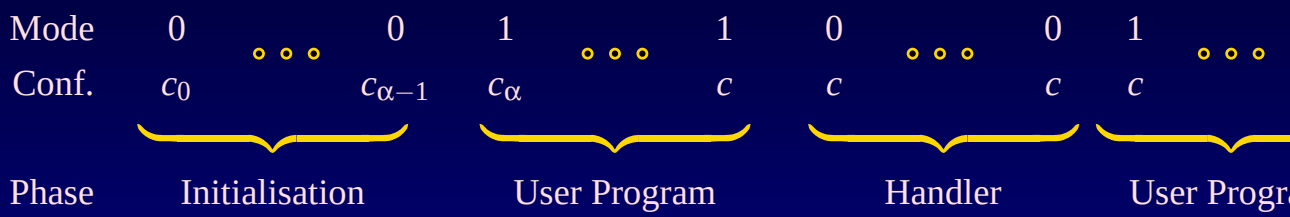
- Similar to tag computation
- Key concept for hardware verificatio

Hypothesis: $I(fetch, T) = i$, translation f
Claim: $IR.din^V = PM^i_S(pma^i_S(DPC^i_S))$

Formal proof: part of PhD thesis project
(Khabarovsk State Technical University)

Virtual Memory Theorem (S

Consider a computation of DLX_S:



Initialisation: $\Pi(c_S^\alpha) = c_V^0$

Simulation Step Theorem:

- **!** 2 page faults per instruction possible (fetch & load/store)

Virtual Memory Theorem I

Assume $\Pi(c_S^i) = c_V^j$.
Define:

- Same cycle or first cycle after handle

$$s_1(i) = \begin{cases} i & \text{if } \neg p \\ \min\{j > i, mode^j\} & \text{other} \end{cases}$$

- Cycle after pagefault-free user mode

$$s_2(i) = \begin{cases} i+1 & \text{if } \neg pff^i \wedge \\ s_1(s_1(i)) + 1 & \text{otherwise} \end{cases}$$

Claim: $\Pi(c_S^{s_2(i)}) = c_V^{j+1}$

Liveness

We must maintain in Sys:
$MRSI$ (most recently swapped-in page)

Page fault handler must not evict page M

Formal proof: PhD thesis project of T. In
(Saarbrücken)

Translation Look-aside Buff

1-level lookup: formally caches for PT-re

- Consistency of 4 caches:
  ICache, DCache, ITLB, DTLB
- Simply invalidate TLBs at mode swit
  sufficient by `sync` conditions

Translation Look-aside Buff

Multi-level lookup: TLB is *simplified* cac

- Normal cache entry:

c_ad	$v = 1$	tag	$PM(tag, c\_ad)$

- TLB entry:

c_ad	$v = 1$	tag	$pma^t(tag, c\_ad)$

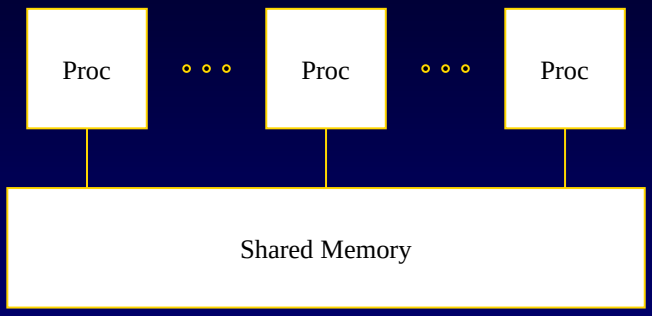
  t : time of last `sync` / `rfe`
  Invalidate at mode switch to 1
- No writeback or load of lines
- Only ‘cache’ reads and writes of valu by MMU

Formal verification trivial from verified c

Multiuser with Sharing

- Easy implementation and proof of pr
  properties using right bits $r(va)$ and

Main Frames I



- PM: sequentially consistent shared m
  cache coherence protocol)
- New software condition: before chan
  page table entry *all* processors sync
- Sync hardware: some AND trees and
  interfaced with CPU
  Considered alone almost completely
  meaningless.

Main Frames II

Theorem: user programs see sequentially
virtual shared memory
Proof: Phases OS - UPs OS UPs
Global serial schedule:

- in each phase from sequential consis
physical shared memory
- straight forward composition across
- remaining arguments not changed!

Summary

- Mathematical treatment of memory m
- Intricate combination of hardware an considered
- Formalization under way

Future Work

Formal verification of

- compilers,
- operating systems,
- applications,
- communication systems

in industrial context. . .

Future Work

Formal verification of

- compilers,
- operating systems,
- applications,
- communication systems

in industrial context. . .

. . . with a little help from