Automated Compile-Time and Run-Time Techniques to Increase Usable Memory in MMU-Less Embedded Systems

Lan S. Bai, Lei Yang, Robert P. Dick
Northwestern University
Dept. of EECS

Outline

MEMMU: Memory Expansion on Embedded Systems without MMUs

- The need for MEMMU
- Design of MEMMU
- Optimizing MEMMU for performance
- Automating MEMMU
- Experimental results

Motivation

- Memory is tightly constrained in many embedded systems
  - MICAz: 4 KB RAM; TelosB: 10 KB RAM
  - Increasing RAM increases cost, power, size
- Many low-power, inexpensive embedded systems do not have MMU
- Memory requirement keeps growing
  - Computation intensive applications (signal processing, routing, encryption…)

Related work

- Software virtual memory management for MMU-less embedded systems
  - Choudhuri and Givargis, 2005
- Hardware-based code and data compression for embedded systems
  - Lekatsas, Henkel, and Wolf, 2000;
  - Tremaine et al., 2001

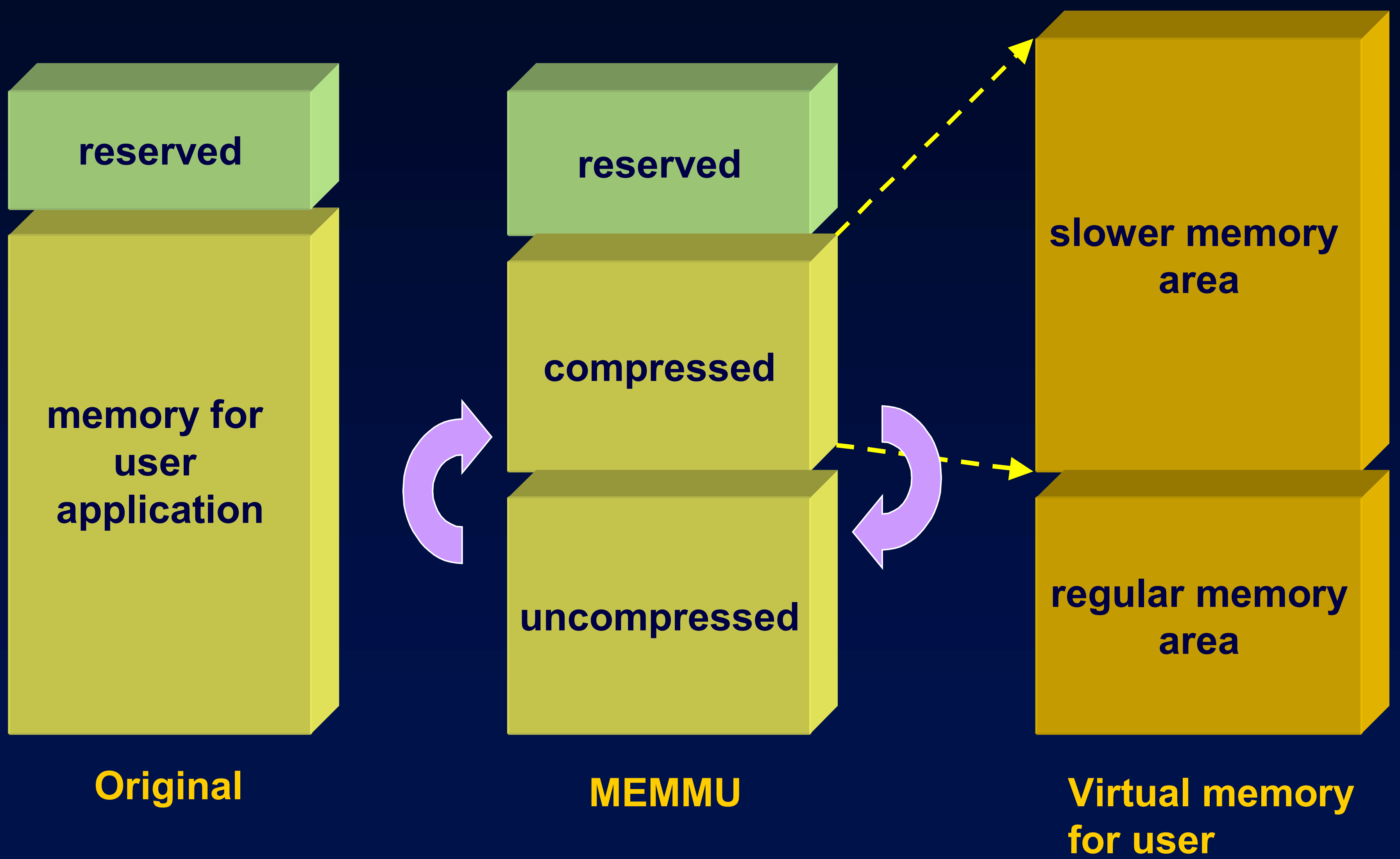
Related work

- Software-based online memory compression
  - Biswas, Simpson, and Barua, 2004
  - Yang et al., 2005, 2006
- Compression for reducing communication in sensor networks
  - Pradhan, Kusuma, and Ramchandran, 2002
  - Pereira et al., 2003
- Software-based memory compression algorithms
  - LZO, RIZZO, WKdm, PBPM

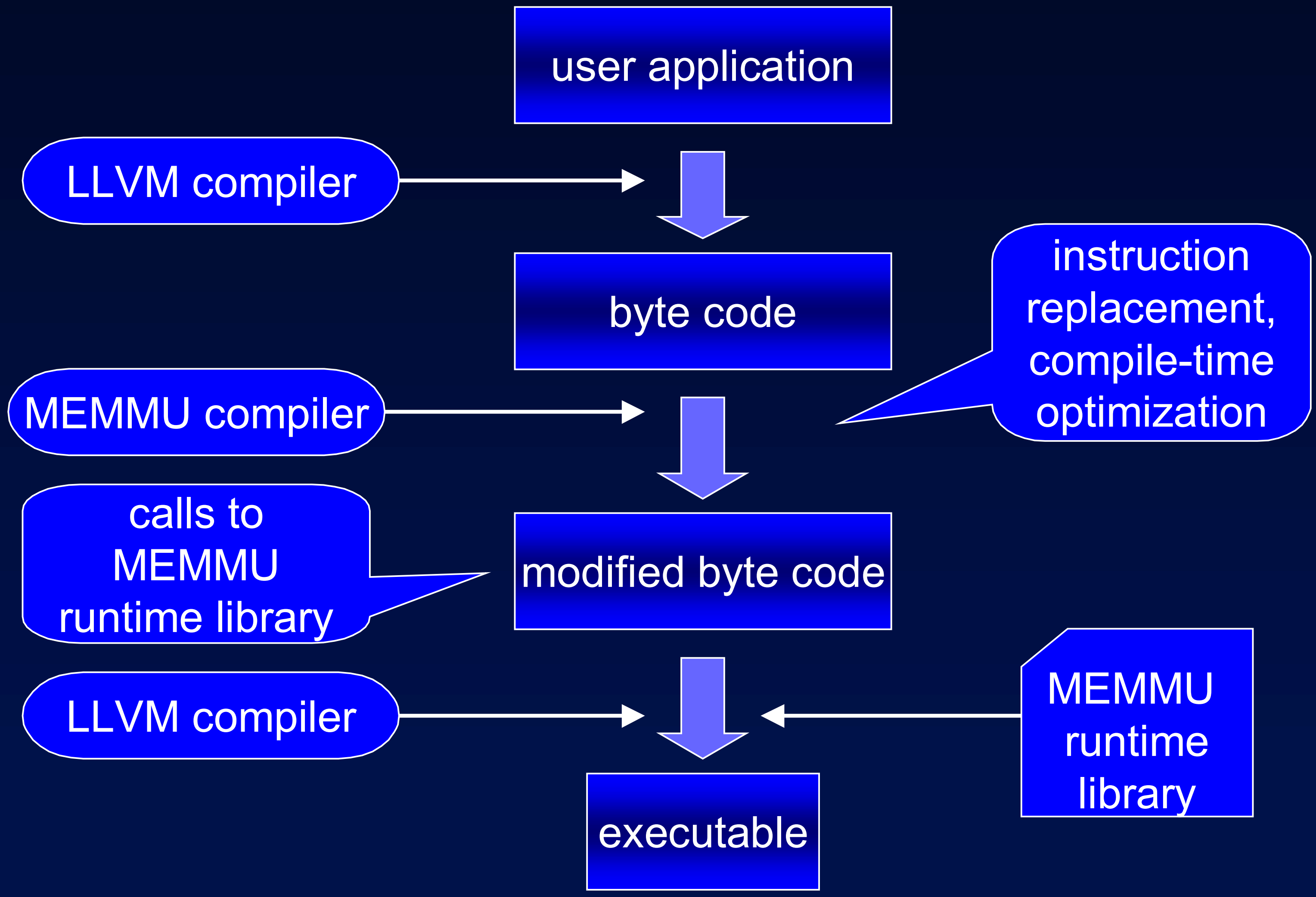
Features of MEMMU

- No change to hardware
- Requires no MMU support
- Automated technique, few or no change to application code by users
- Optimized to minimize performance overhead

Overview of MEMMU



Overview of MEMMU (cont.)



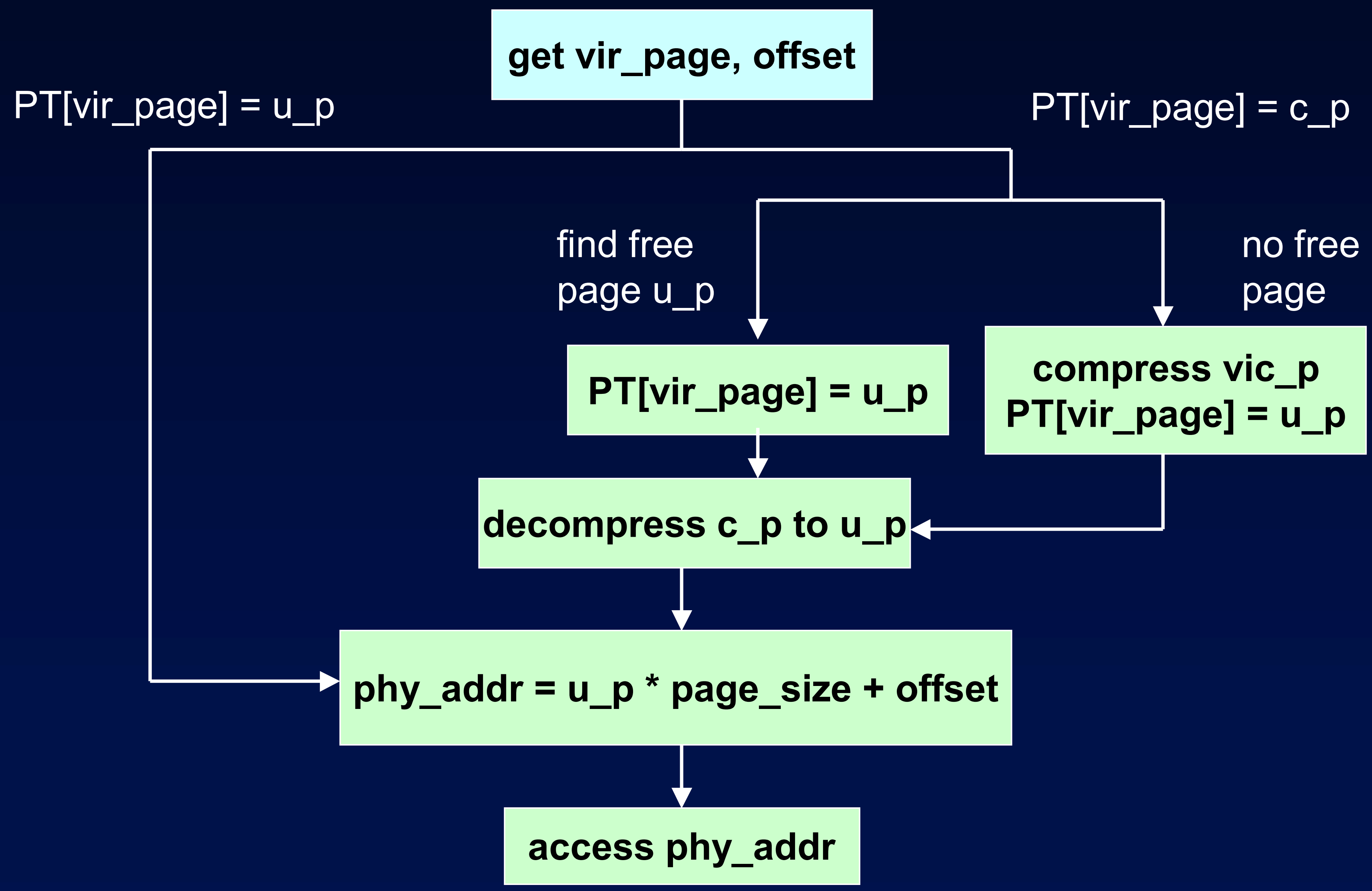
Challenges and sub-problems

- Goal
  - Maximize the increase in usable memory
  - Minimize performance and energy penalties
- Sub-problems
  - Select data to compress
  - Schedule compression and decompression
  - Organize compressed and uncompressed memory regions
  - Efficient compression algorithm

MEMMU design
Handle-based data access

- Page table maps virtual pages to physical pages in compressed and uncompressed regions
- Compressed pages need to be decompressed and moved to uncompressed region before access
- A victim page in uncompressed region is compressed and moved to compressed region when needed

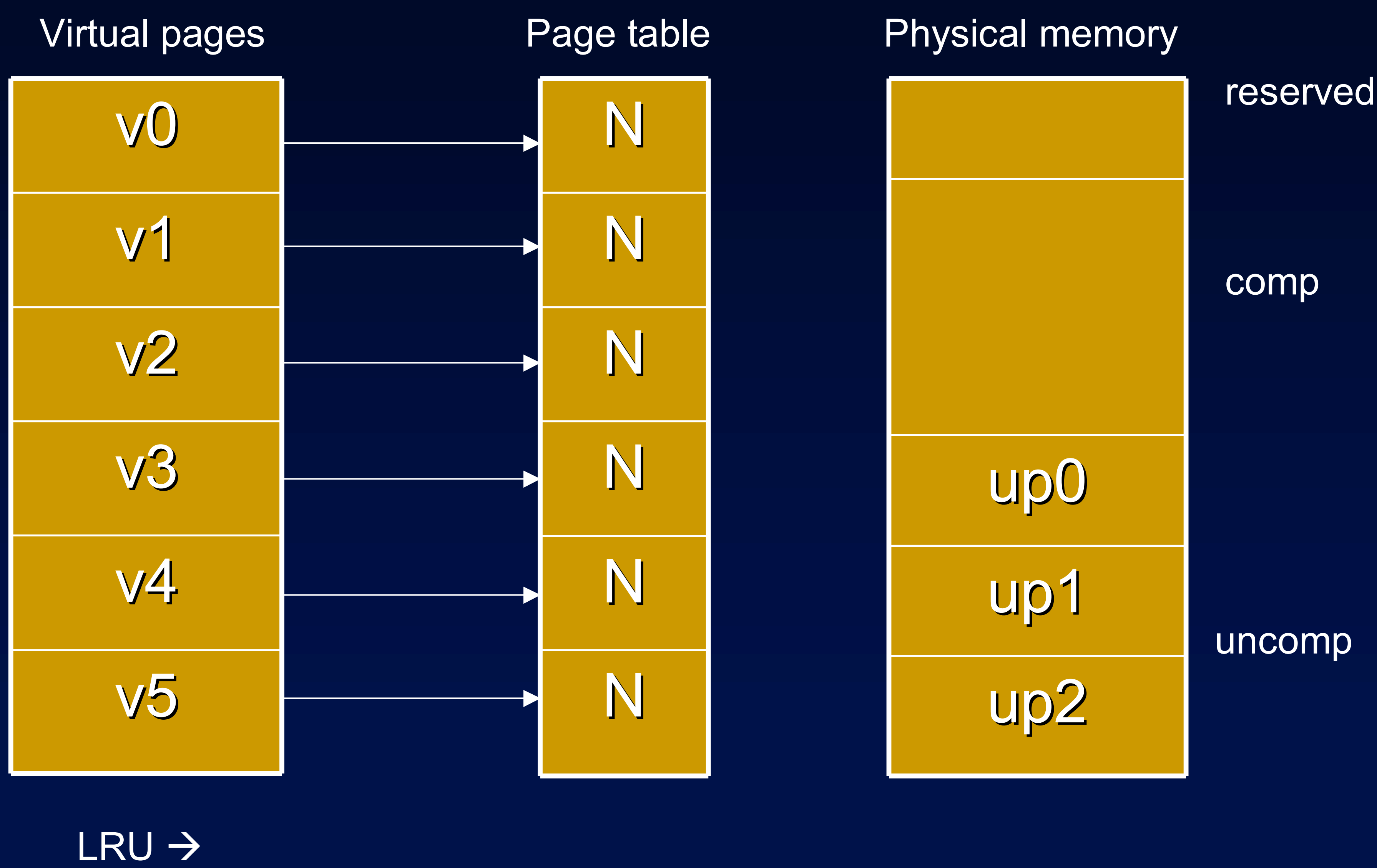
Example of access vir_addr



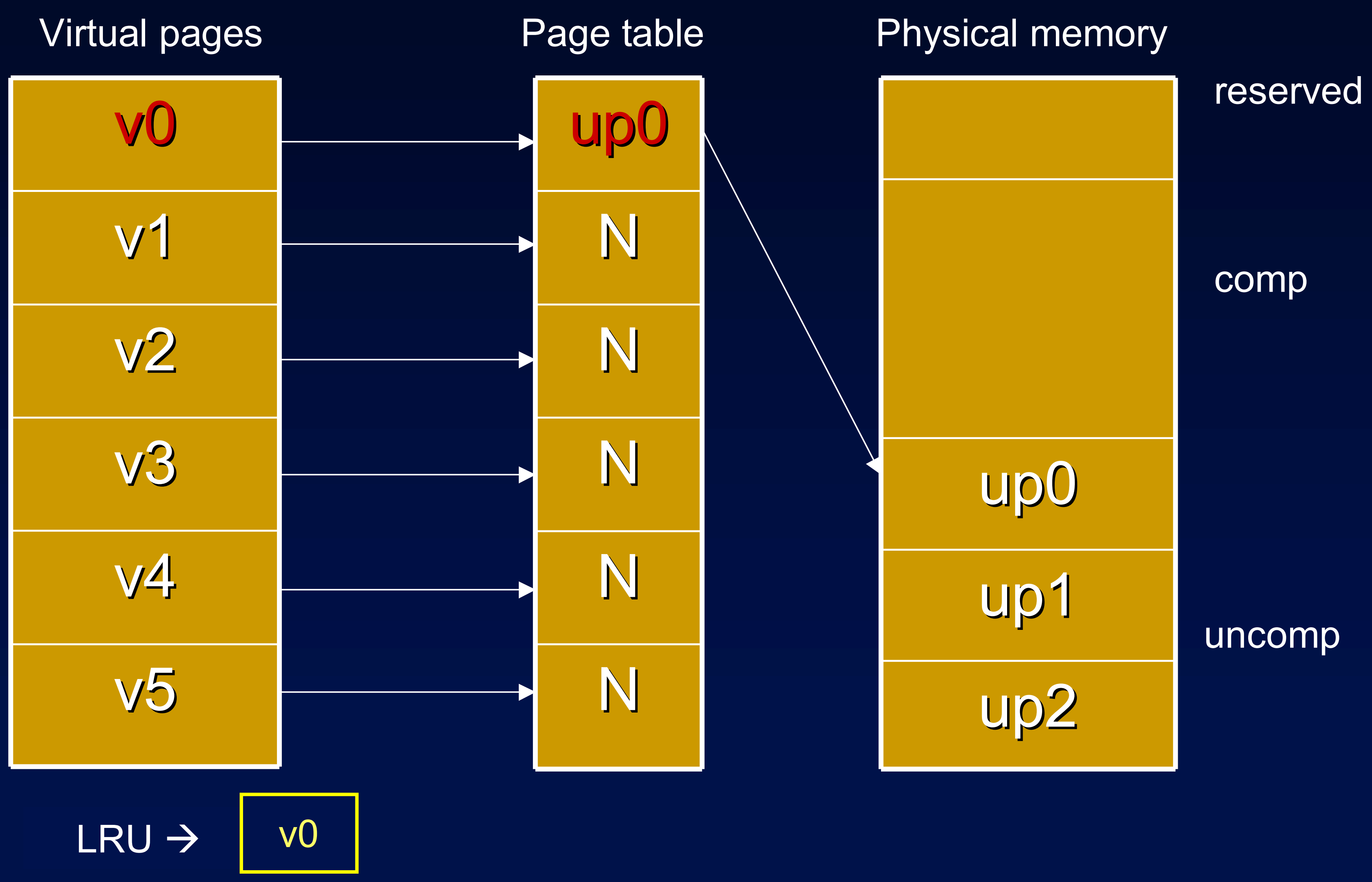
MEMMU design
Page replacement

- LRU page replacement policy
  - Minimize page migration times
  - Compress LRU page when no free page in uncompressed region
  - LRU list maintains page reference history

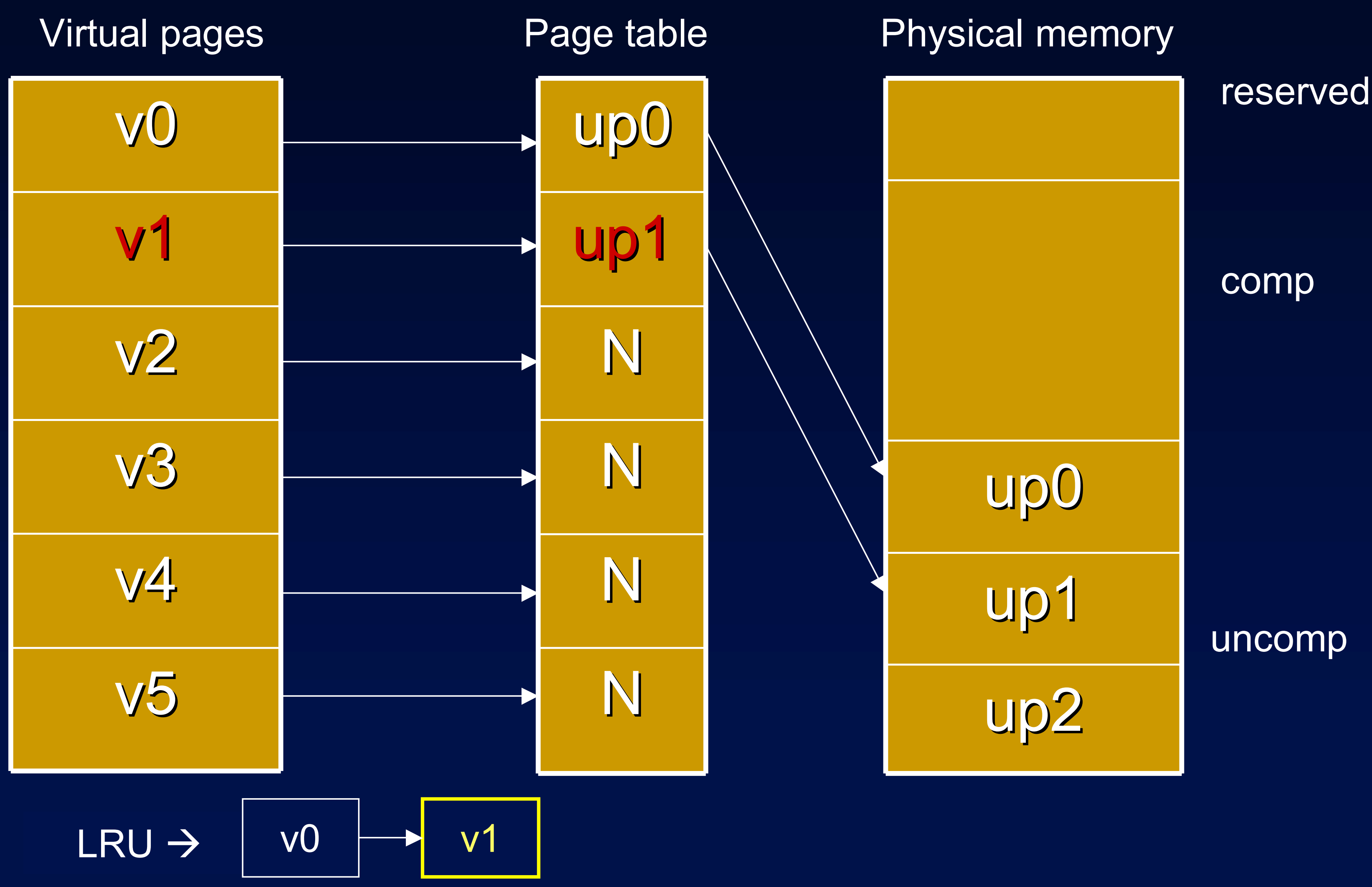
Initial state



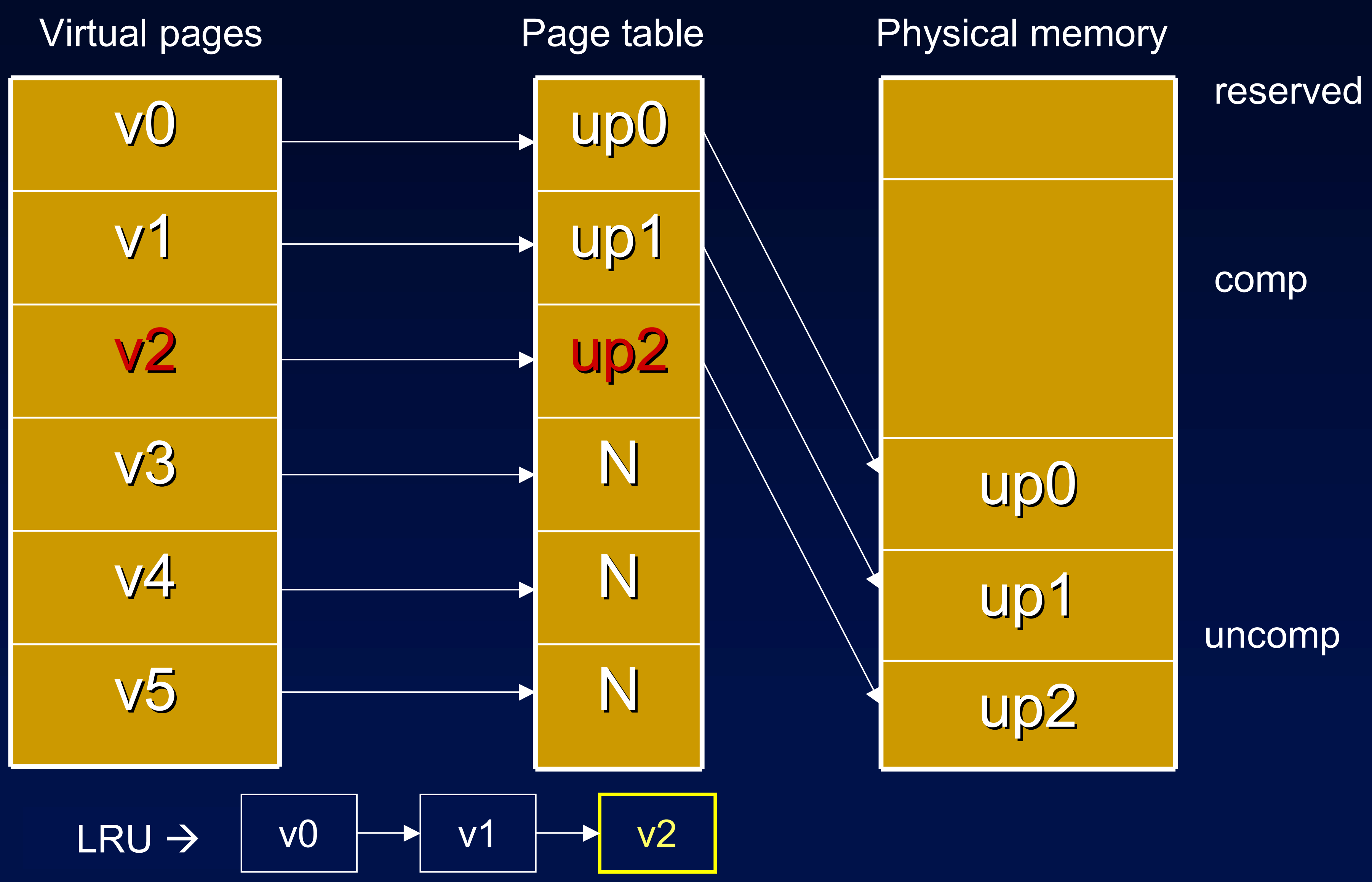
Write to page v0



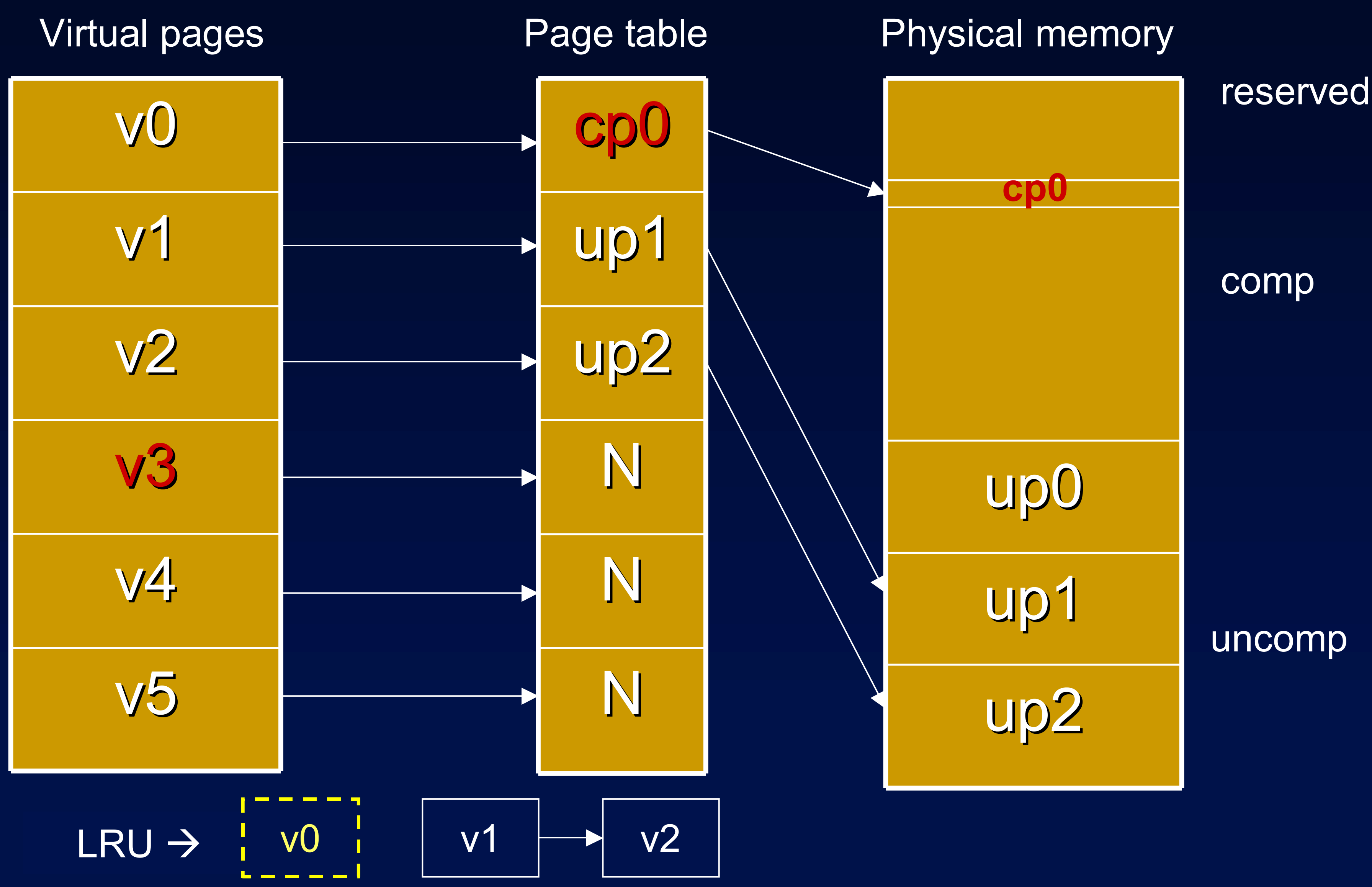
Write to page v1



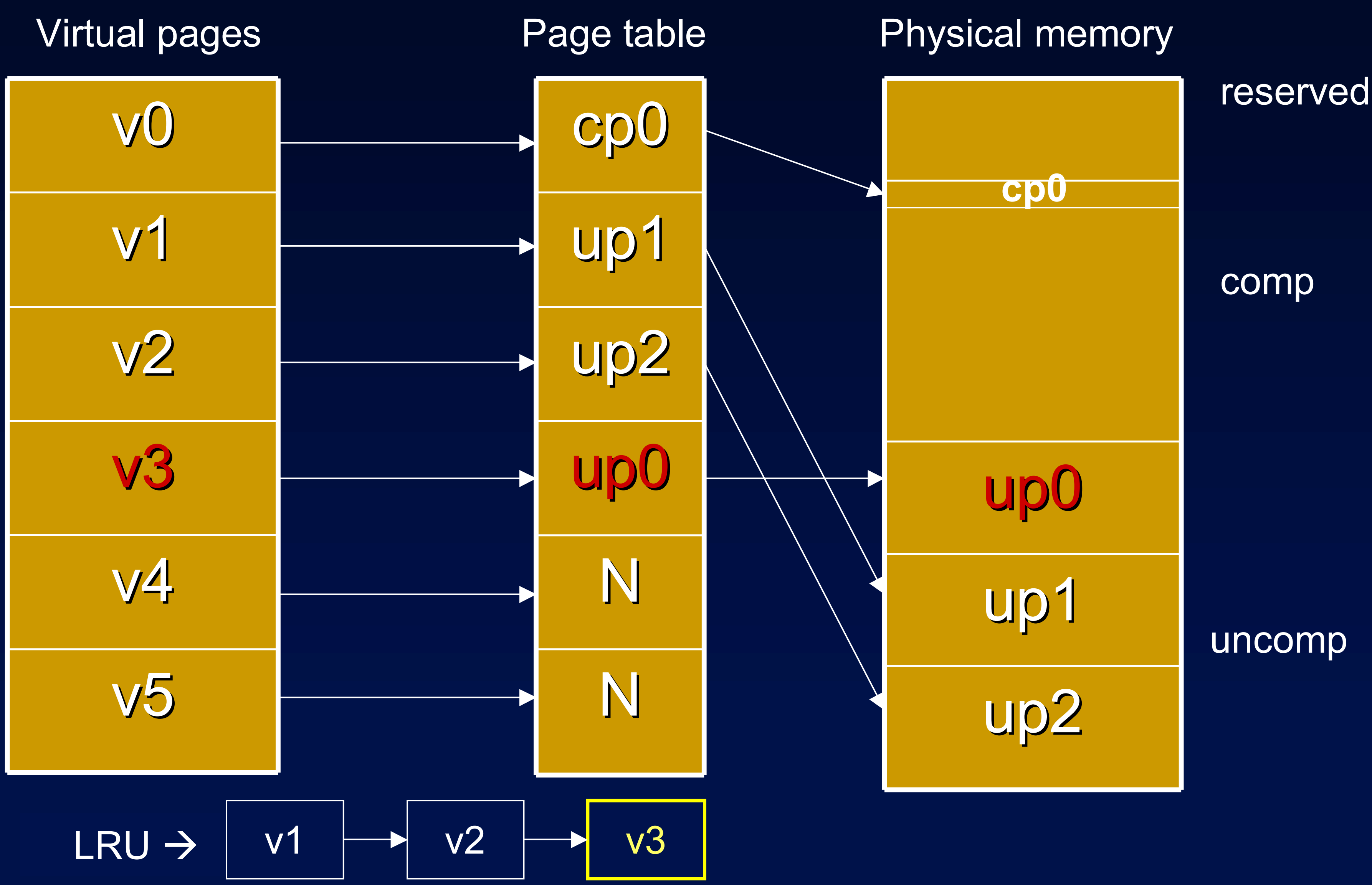
Write to page v2



Write to page v3 – compress up0



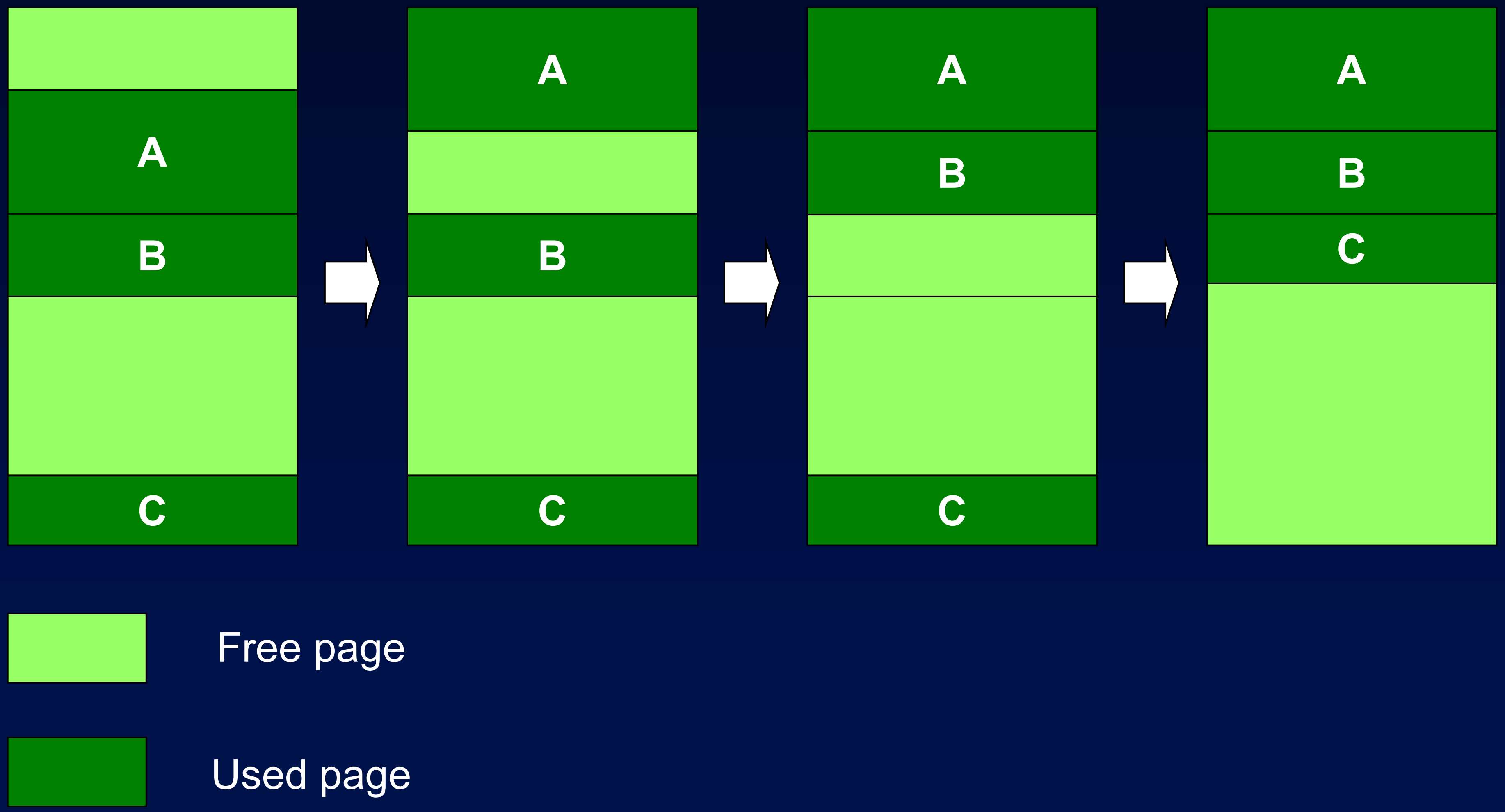
Write to page v3 – map v3 to up0



MEMMU design
Prevent fragmentation

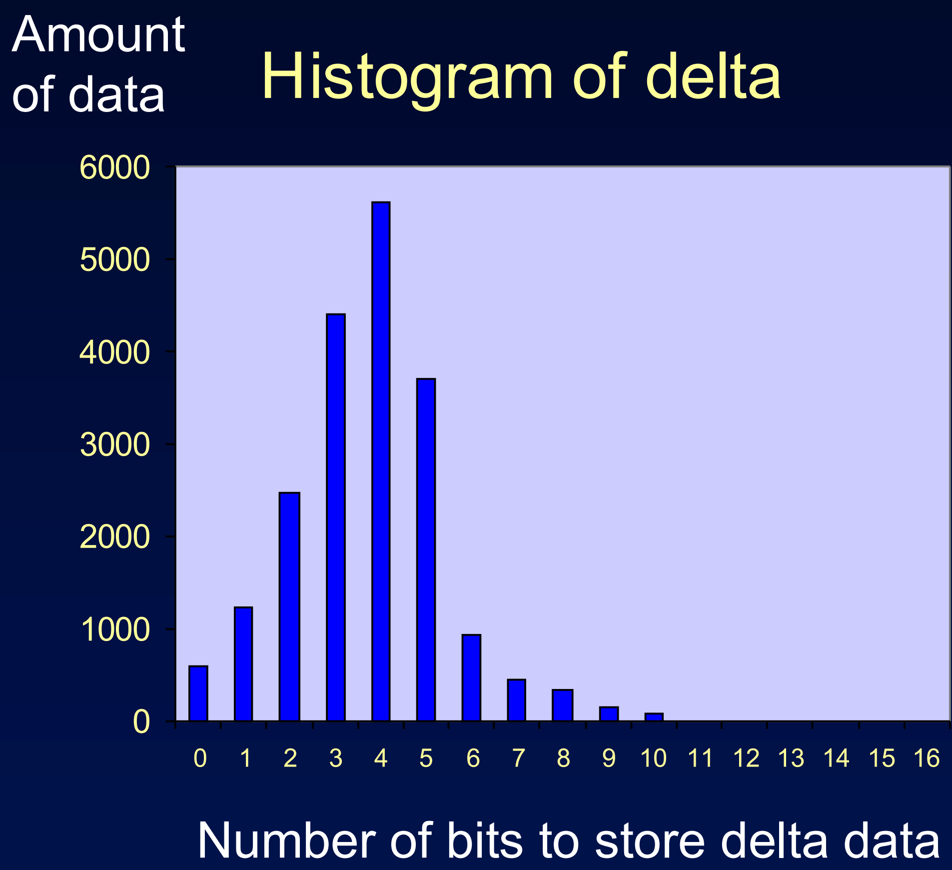
- Heap memory management for compressed region
- Fragmentation occurs in compressed region
  - Reduce the memory expansion proportion
  - Complicates memory expansion prediction
  - May stop running application
- Adjacent free block merging when free a compressed page
- Coalesce when no remaining blocks large enough

Memory coalescing example



Delta compression algorithm

- Compression algorithm affects memory expansion proportion and performance
- Sensor data changes smoothly
- Delta data require fewer bits to store
- Average compression ratio on sensor data: 50%



Handle check optimization

- Handle access = handle check + address translation + (data migration) + LRU update
- Performance overhead high, proportional to number of total memory access
- Reduce overhead by removing unnecessary handle check, address translation and LRU update

Frequent references optimization

- Some small data structures are referenced frequently
  - E.g., coefficient kernel in image convolution
- Put them in reserved region
  - MEMMU puts all scalars into reserved region
  - Reduce handle check, address translation, LRU update related to small data

Run-time handle check optimization

- A sequence of data references access the same page, only need to check the page table once
- Use conditional to prevent unnecessary handle checks
- May increase overhead

```
if cur_page != pre_page {
        check_handle (cur_page);
        pre_page = cur_page;
} else {
}
```

Loop transformation and compile-time elimination of inner-loop checks

- Transform a loop to nested loop
- Within inner loop, only access one page

```
for i in {0…N} do
  A[i] = x
end for
```

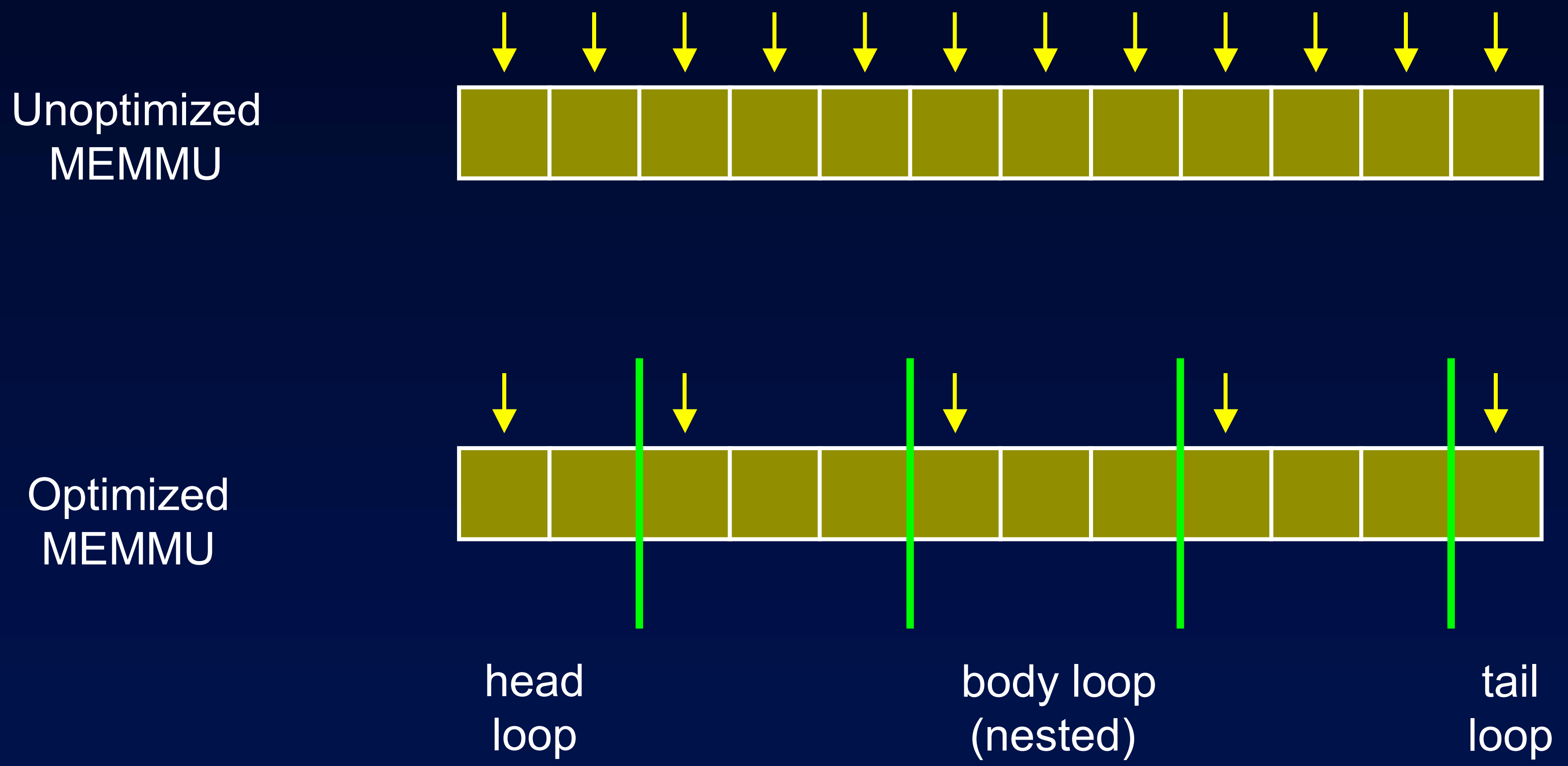
```
for i in {0…N} do
  cur_p = (A + i) / PSZIE
  check_handle(cur_p)
  write_handle(A + i, x)
end for
```

```
pnum = N / PSIZE
for i in {0…pnum} do
  check_handle((A + i) / PSIZE))
  for j in {0…psize} do
    write_handle(A + i × PSIZE + j, x)
  end for
end for
```

N checks

N / PSIZE checks

Loop transformation and compile-time elimination of inner-loop checks



Handle check hoisting

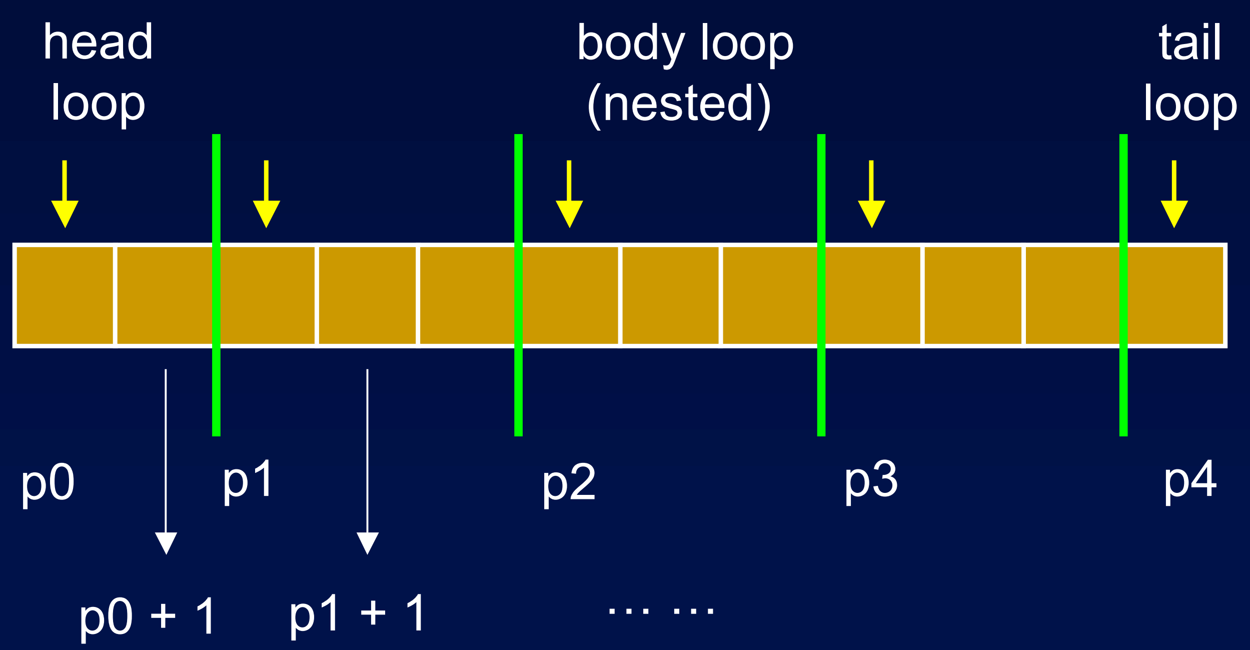
- Pages accessed inside a loop can reside in uncompressed region
- Replace multiple handle checks inside a loop by fewer checks outside the loop

```
for (i = 0; i < M; ++i)
    for (j = 0; j < N; ++j)
        p = (A + N * i + j) / PSIZE
        handle_check(p)
        access A[i][j]
```

```
for (i = 0; i < M; ++i)
    p = N / PSIZE
    pmin = A[i][j] / PSIZE
    handle_check pmin to pmin + p
        for (j = 0; j < N; ++j)
            access A[i][j]
```

Pointer dereferencing

- Explore dependencies among sequence of addresses
- Eliminate address translation



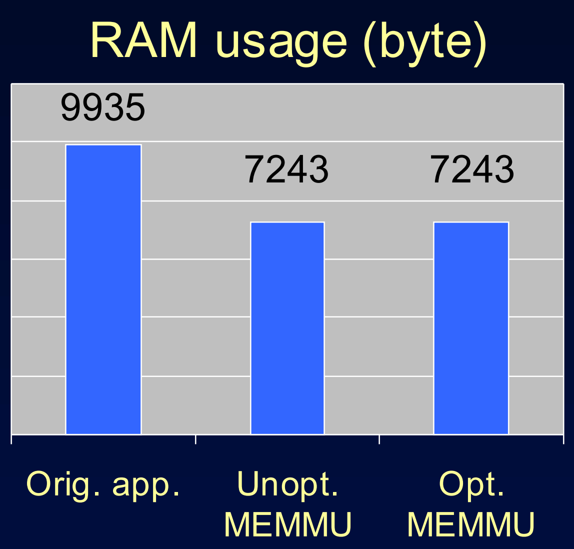
MEMMU evaluation

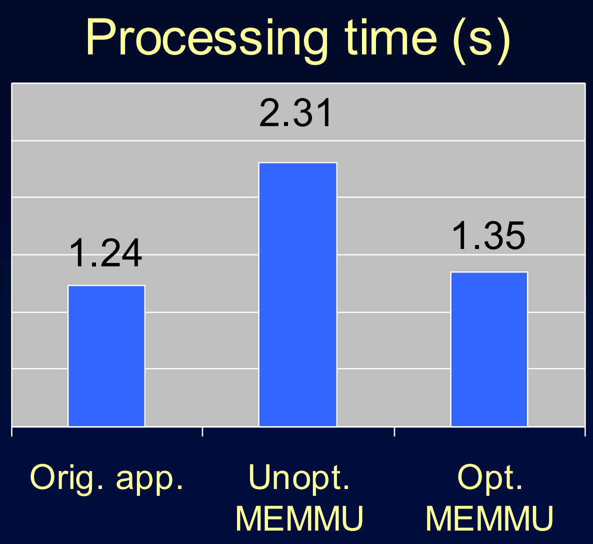
- TelosB wireless sensor node
  - MSP430, 10 KB RAM
- Power measurement
  - National Instrument 6034E data acquisition card
- Original, unopt. MEMMU, opt. MEMMU
- Metrics
  - Memory expansion proportion
  - Power
  - Execution time

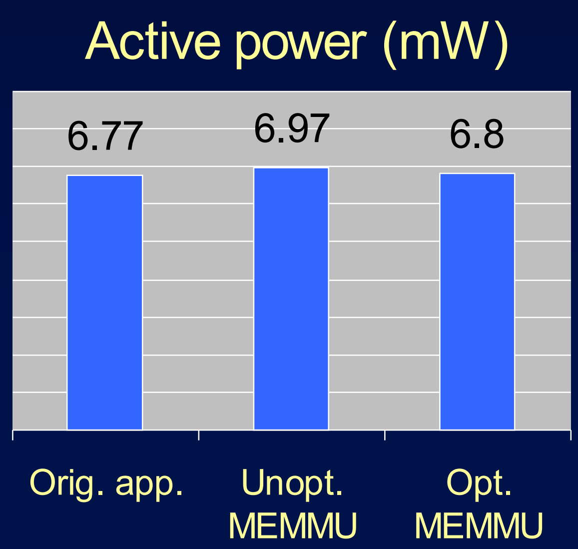
Benchmarks

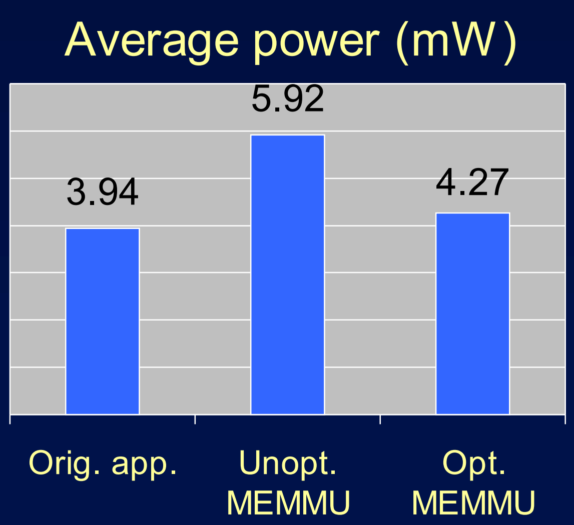
- Sound filtering
  - 1-D convolution useful in signal processing
- Image convolution
  - 2-D convolution useful in signal processing
- Light sampling
  - Periodically samples and transmits light level
- Covariance matrix computation
  - Matrix operation useful in PCA, feature extraction
- Audio signal correlation computation
  - Useful in automated location calibration

Experimental Results
Filtering benchmark



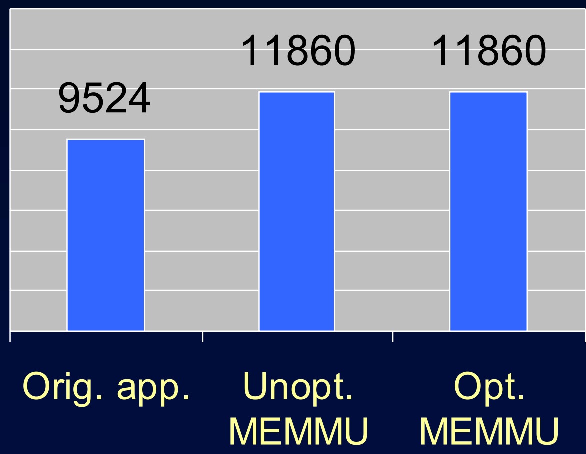




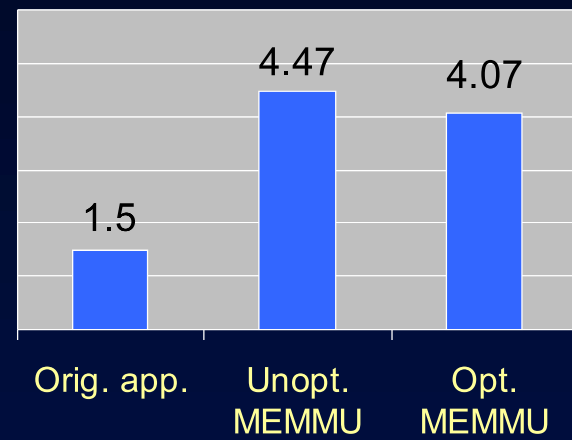


Experimental Results
Image convolution benchmark

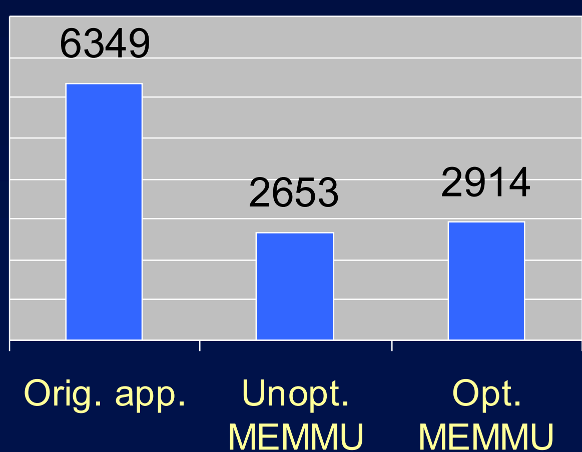
Image size (byte)



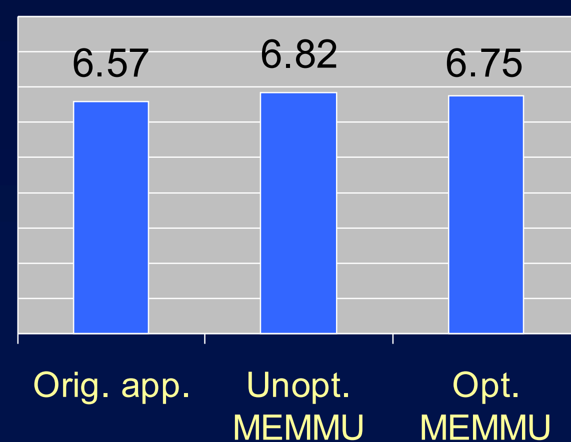
Processing time (s)



Processing rate (byte/s)

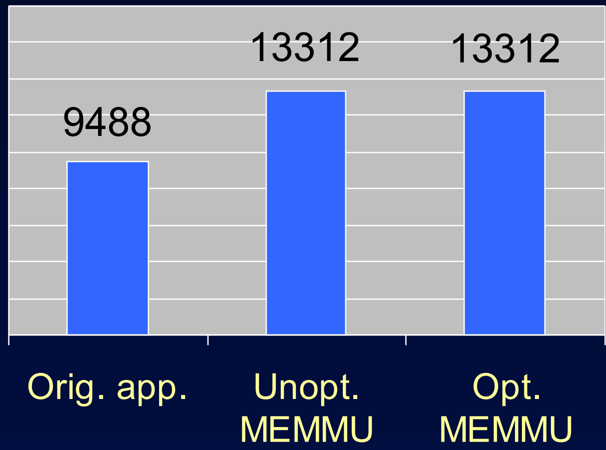


Power (mW)

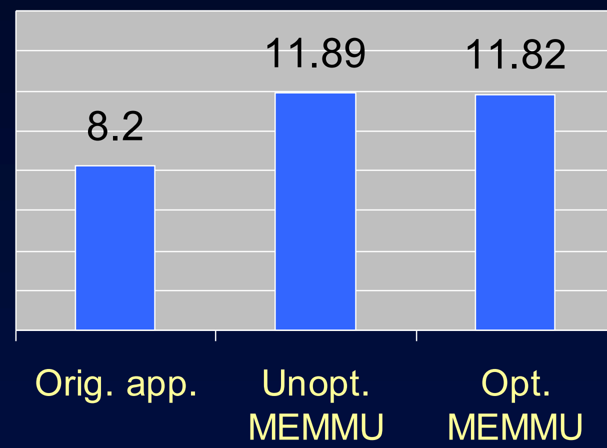


Experimental Results
Light sampling benchmark

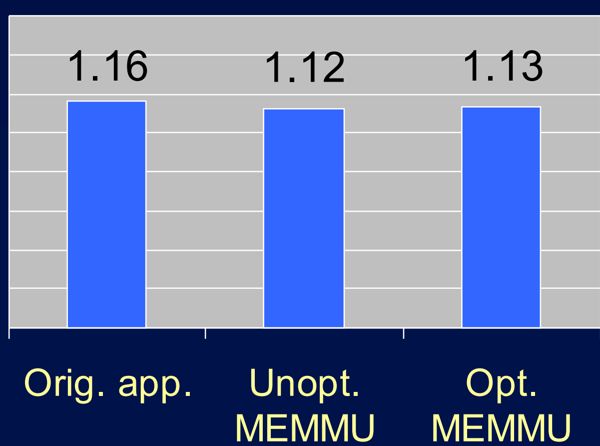
Buffer size (byte)



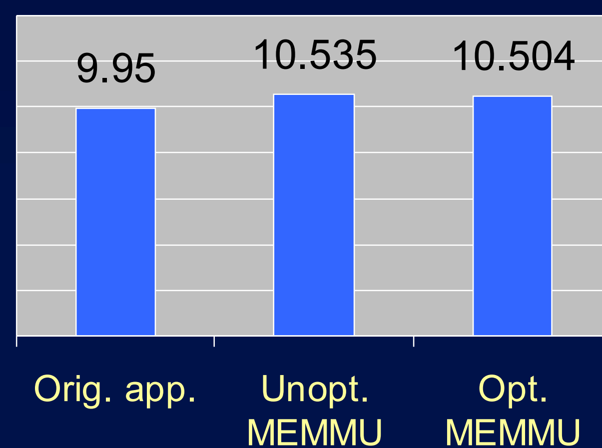
Processing time (s)



Processing rate (byte/s)

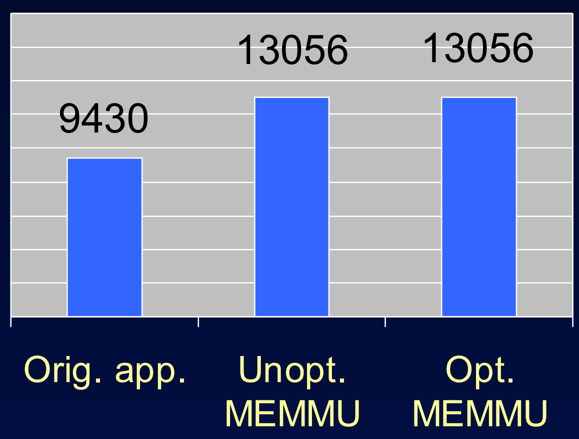


Power (mW)

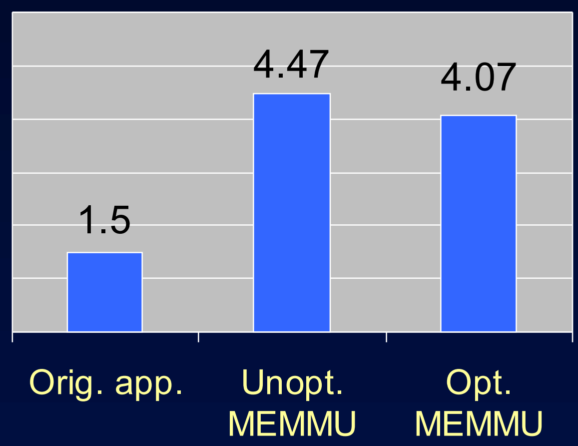


Experimental Results
Covariance matrix computation benchmark

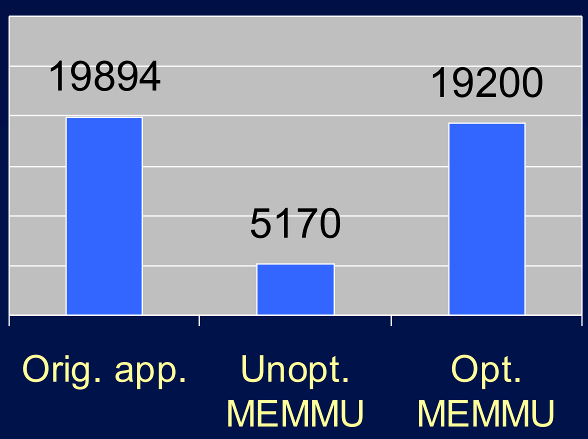
Buffer size (byte)



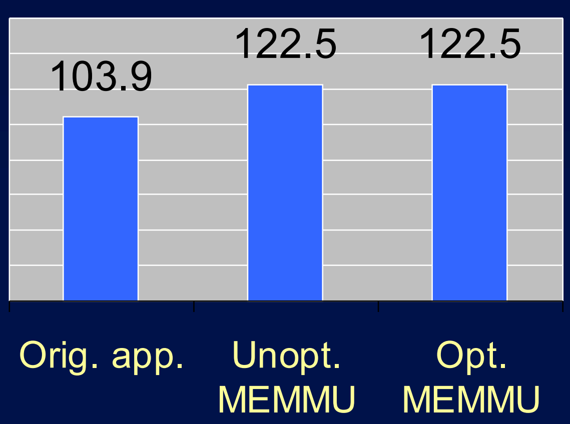
Processing time (s)



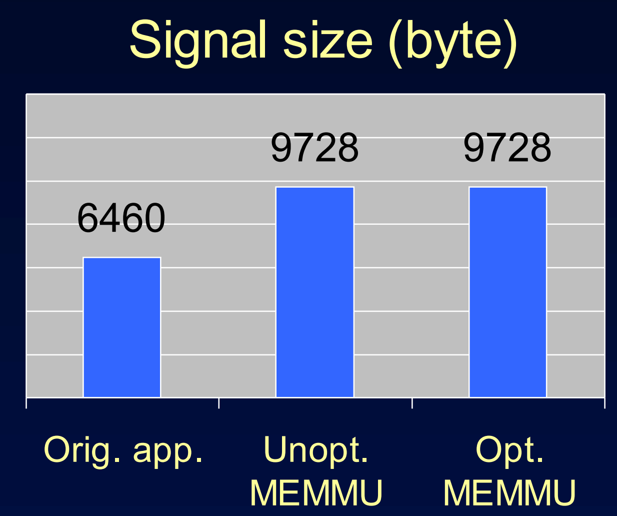
Processing rate (byte/s)

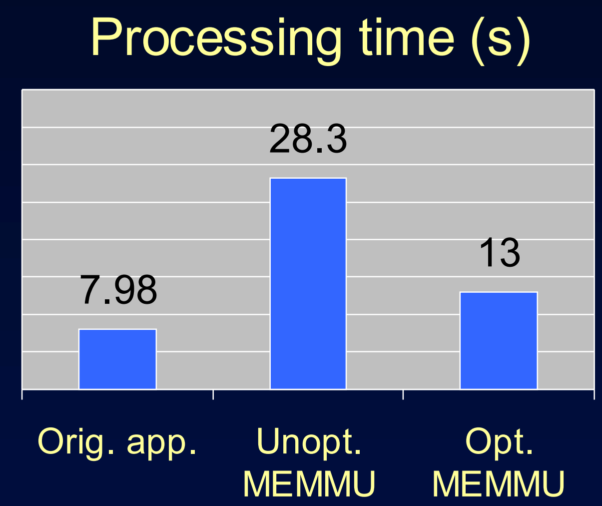


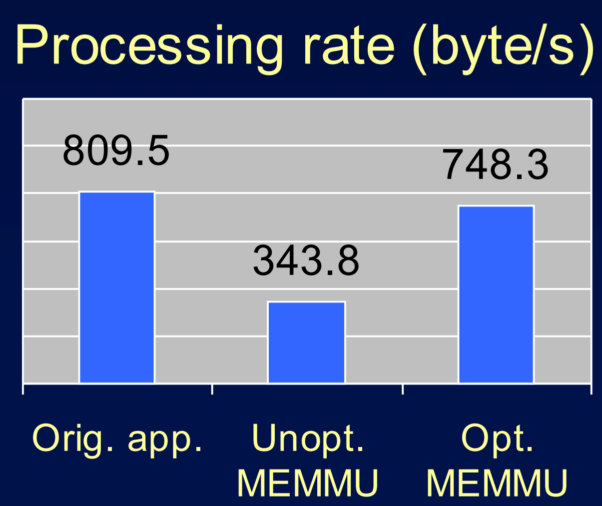
Power (mW)

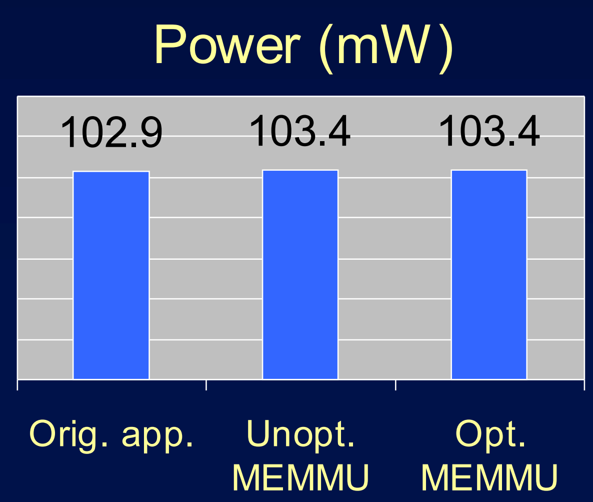


Experimental Results
Correlation computation benchmark









Conclusions

- MEMMU increases usable memory by 50%, usually with small performance and power penalties
- Optimization techniques largely reduces performance penalty
- One application may benefits more from one optimization method depending on its memory access pattern

Thank you for attending

Questions?

MEMMU run-time library

- Check_handle (vir_page)
  - Ensure or bring vir_page to uncompressed region
- Read_handle8 (vir_addr)
- Read_handle16 (vir_addr)
- Read_handle32 (vir_addr)
- Write_handle8 (vir_addr, data)
- Write_handle16 (vir_addr, data)
- Write_handle32 (vir_addr, data)
  - Translate to physical address in uncompressed region, and then read or write to vir_addr

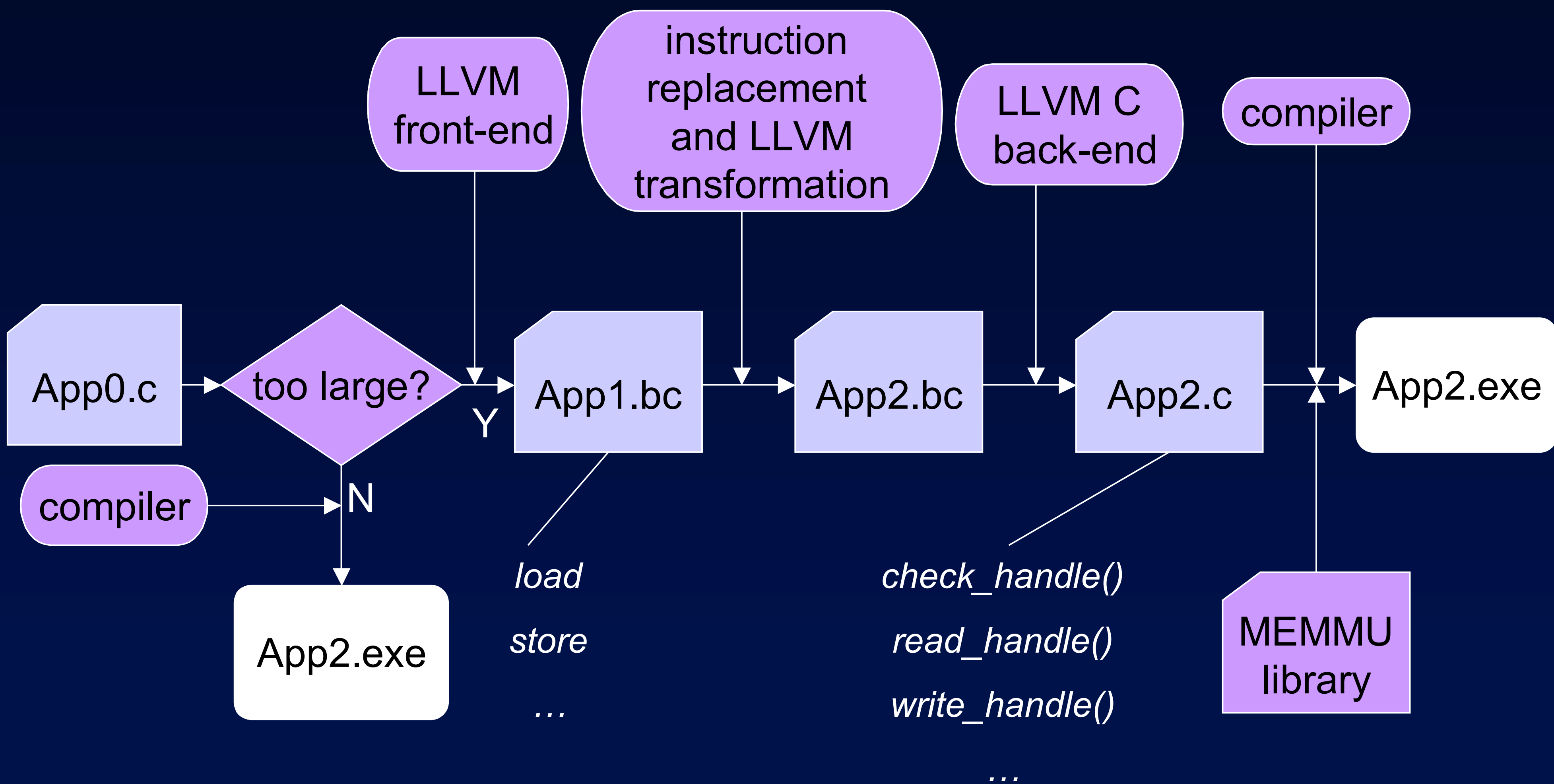
MEMMU design
Interrupt management

- Interrupt may access user memory
  - ADC interrupt
- Disable interrupt when accessing data in user memory?
  - Missing interrupt
- Allow interrupts at anytime ?
  - Inconsistent data state
  - Page table is updated but not finish data migration

MEMMU design
Interrupt management

- Ring buffer in reserved region
  - Data is written to ring butter when they arrive
  - Allow interrupt at any time
- Worst-case delay: coalesce + compression + decompression
- Ring buffer is needed only when sampling period is shorter then worst-case delay

Generate executable with MEMMU



- What if add MMU?
- How to predict memory usage?
- How to decide the sizes of each regions?
- How to deal with interrupts?
- How general is the compile-time opt?

MMU

- DTB (data translation buffer)
  - ITB 0.00131649175 0.00059400296
  - DTB 0.00313065017 0.00073307456
  - FPAdd 0.00217439653 0.00089220157