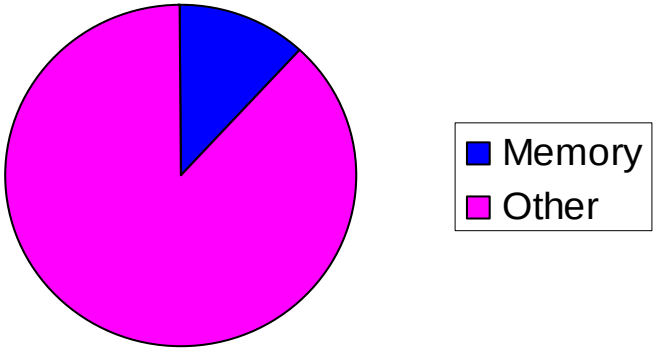
Outline for Today

- Objective
  - Power-aware memory
- Announcements

Memory System Power Consumption

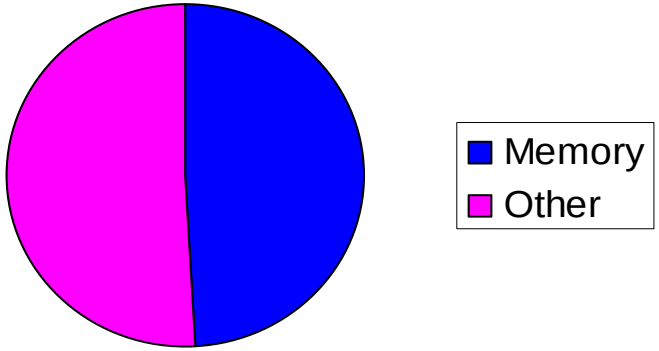
Laptop Power Budget

9 Watt Processor



Handheld Power Budget

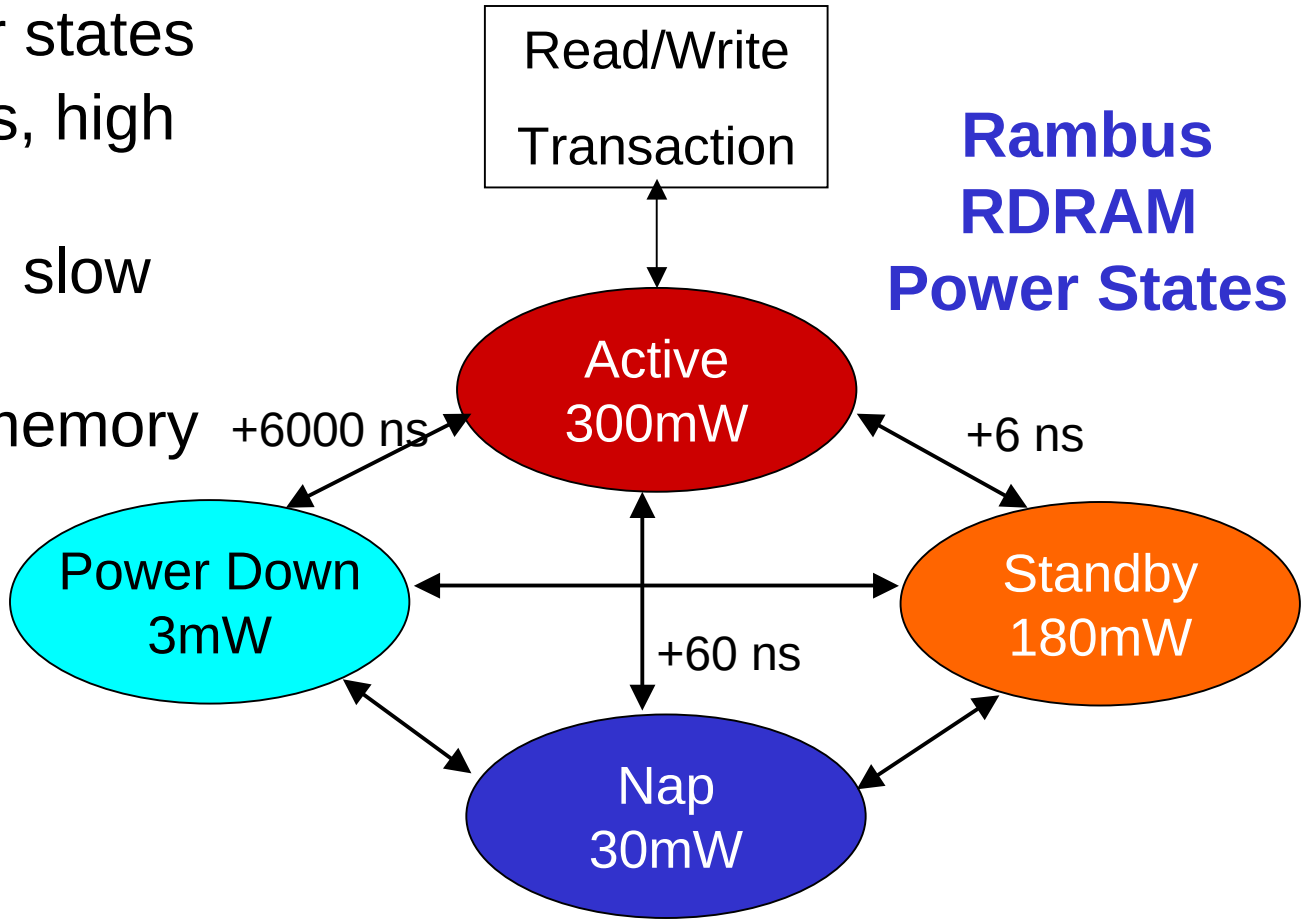
1 Watt Processor



- Laptop: memory is small percentage of total power budget
- Handheld: low power processor, memory is more important

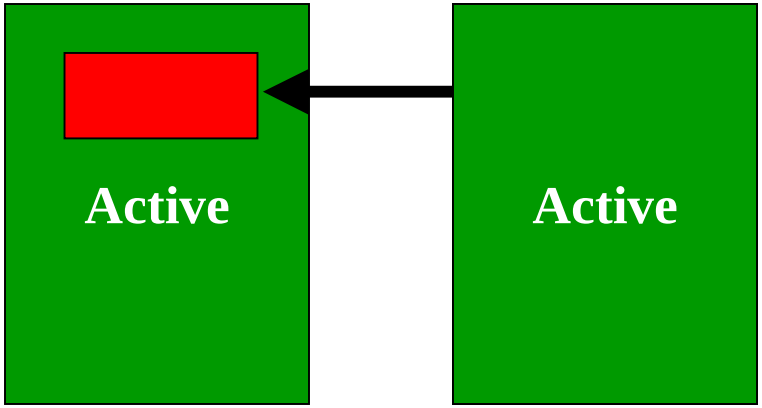
Opportunity: Power Aware DRAM

- Multiple power states
  - Fast access, high power
  - Low power, slow access
- New take on memory hierarchy
- How to exploit opportunity?



RDRAM as a Memory Hierarchy

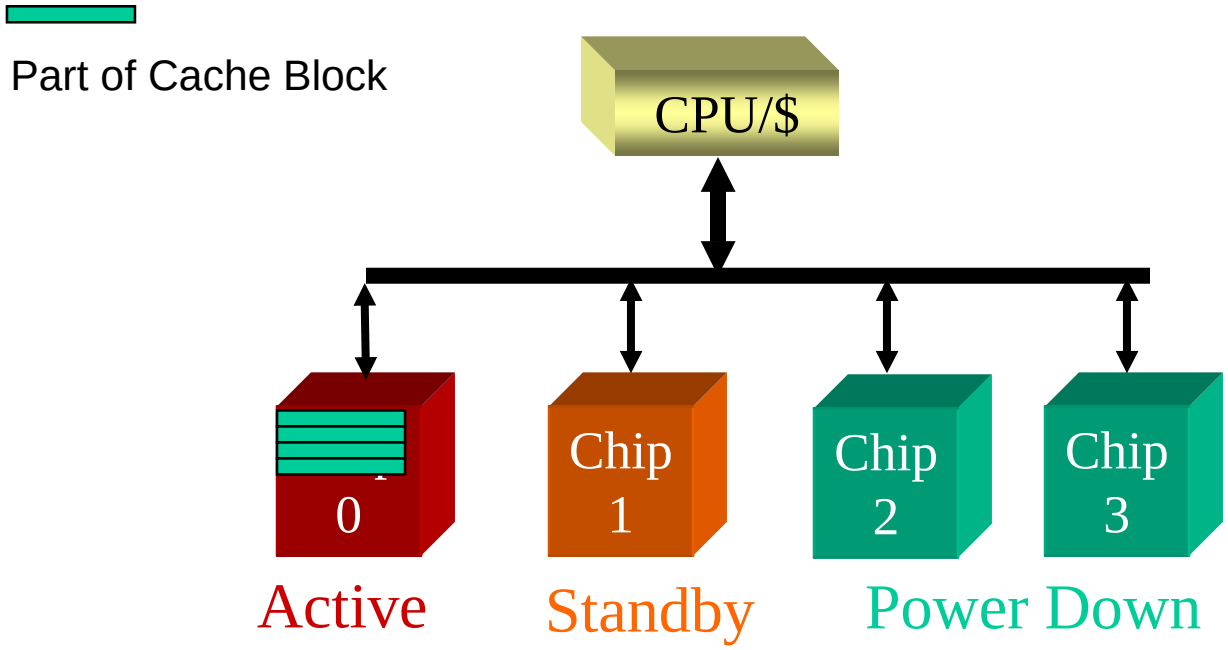
- Each chip can be independently put into appropriate power mode
- Number of chips at each “level” of the hierarchy can vary dynamically.



Policy choices

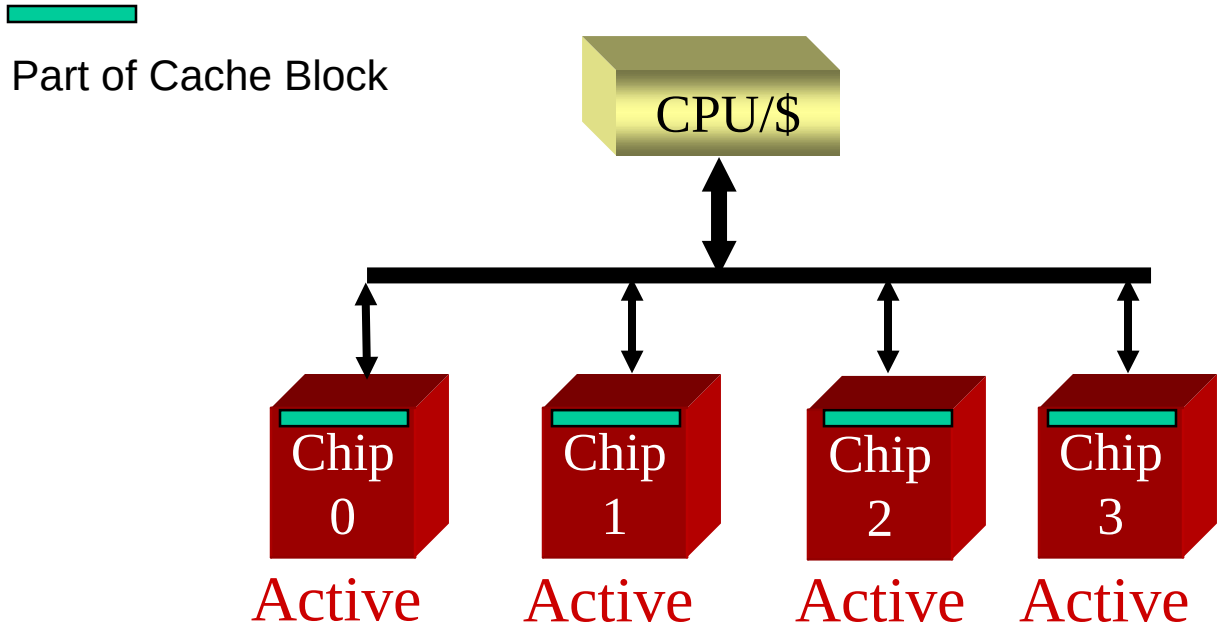
- initial page placement in an “appropriate” chip
- dynamic movement of page from one chip to another
- transitioning of power state of chip containing page

RAMBUS RDRAM Main Memory Design



- Single RDRAM chip provides high bandwidth per access
  - Novel signaling scheme transfers multiple bits on one wire
  - Many internal banks: many requests to one chip
- Energy implication: Activate only one chip to perform access at same high bandwidth as conventional design

Conventional Main Memory Design



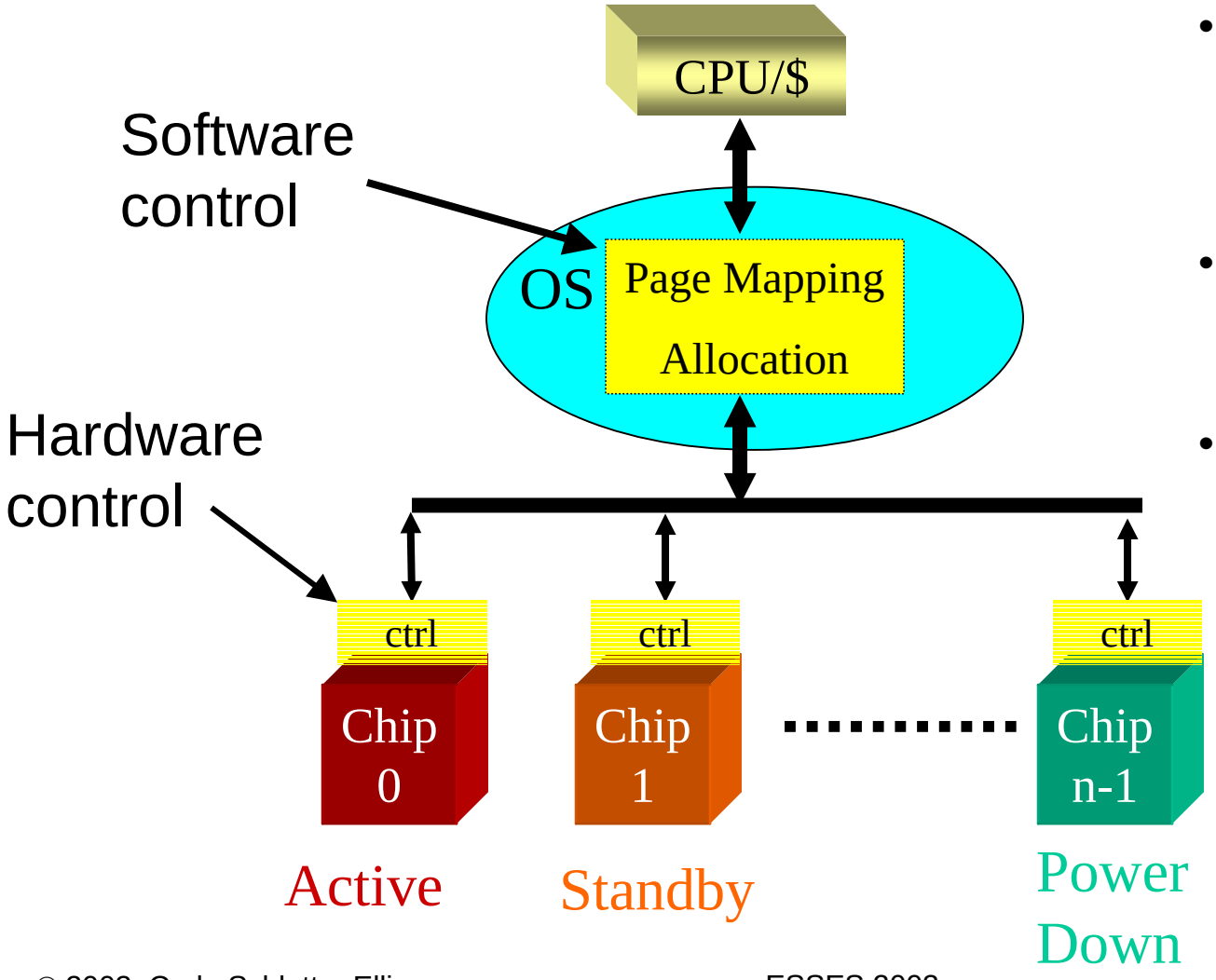
- Multiple DRAM chips provide high bandwidth per access
  - Wide bus to processor
  - Few internal banks
- Energy implication: Must activate all those chips to perform access at high bandwidth

Exploiting the Opportunity

Interaction between power state model and access locality

- How to manage the power state transitions?
  - Memory controller policies
  - Quantify benefits of power states
- What role does software have?
  - Energy impact of allocation of data/text to memory.

Power-Aware DRAM Main Memory Design



- Properties of PA-DRAM allow us to access and control each chip individually
- 2 dimensions to affect energy policy: HW controller / OS
- Energy strategy:
  - Cluster accesses to already powered up chips
  - Interaction between power state transitions and data locality

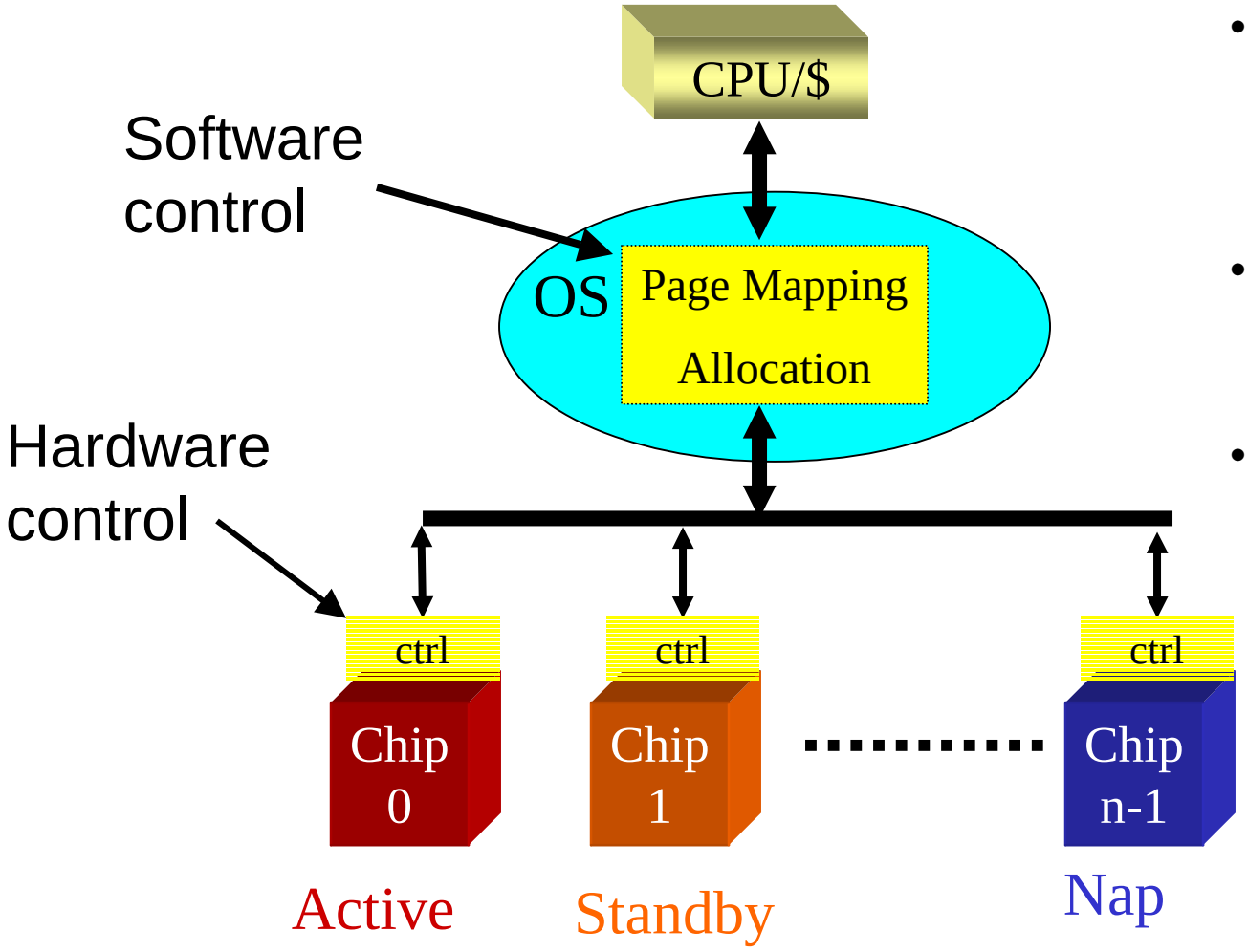
Power-Aware Virtual Memory Based On Context Switches

Huang, Pillai, Shin, “Design and Implementation of Power-Aware Virtual Memory”, USENIX 03.

Basic Idea

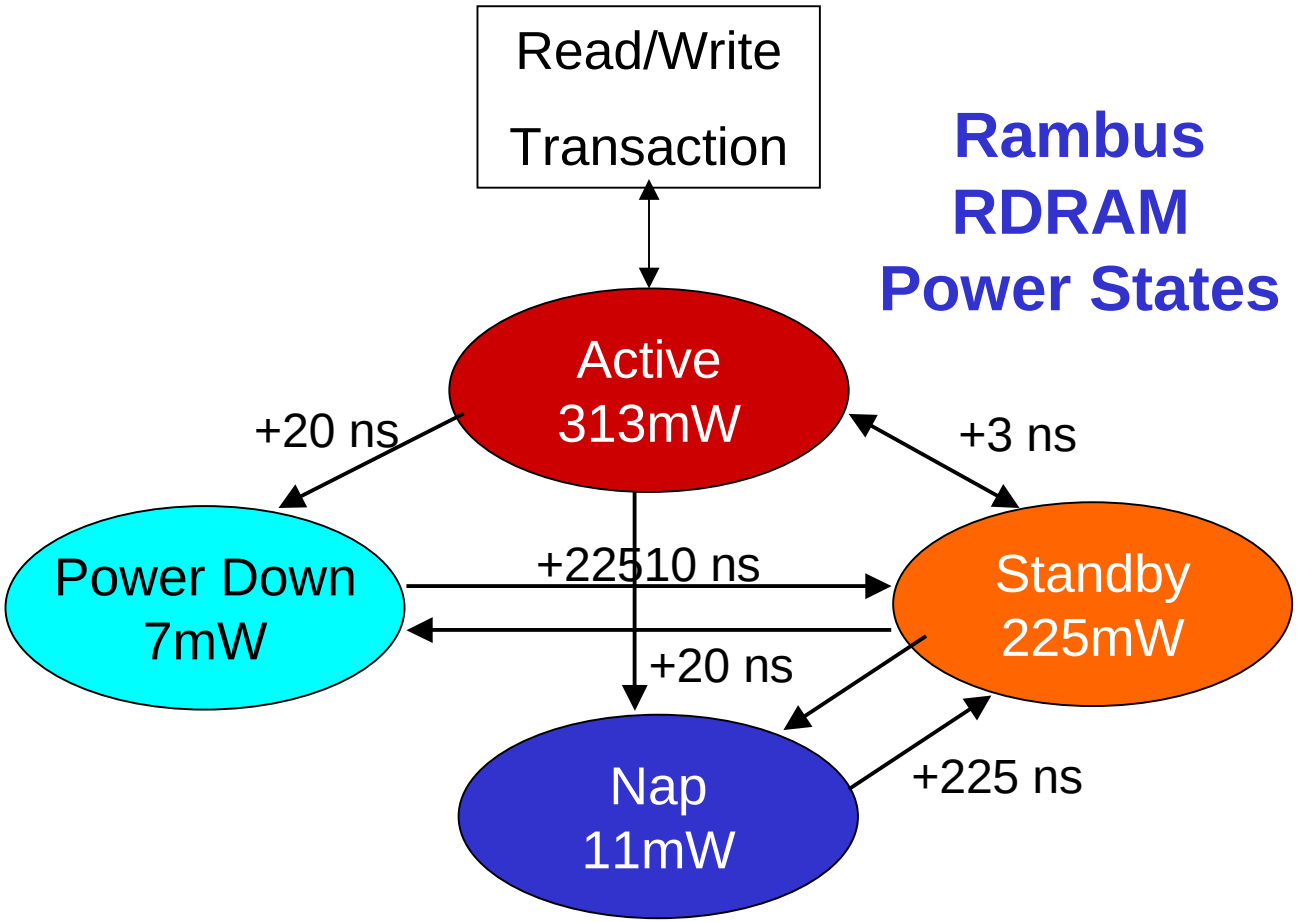
- Power state transitions under SW control (not HW controller)
- Treated explicitly as memory hierarchy: a process's active set of nodes is kept in higher power state
- Size of active node set is kept small by grouping process's pages in nodes together – "energy footprint"
  - Page mapping - viewed as NUMA layer for implementation
  - Active set of pages, α_i, put on preferred nodes, ρ_i
- At context switch time, hide latency of transitioning
  - Transition the union of active sets of the next-to-run and likely next-after-that processes to standby (pre-charging) from nap
  - Overlap transitions with other context switch overhead

Power-Aware DRAM Main Memory Design



- Properties of PA-DRAM allow us to access and control each chip individually
- 2 dimensions to affect energy policy: HW controller / OS
- Energy strategy:
  - Cluster accesses to preferred memory nodes per process
  - OS triggered power state transitions on context switch

Rambus RDRAM



RDRAM Active Components

	Refresh	Clock	Row decoder	Col decoder
Active	X	X	X	X
Standby	X	X	X	
Nap	X	X		
Pwrdn	X			

Determining Active Nodes

- A node is active iff at least one page from the node is mapped into process *i*'s address space.
- Table maintained whenever page is mapped in or unmapped in kernel.
- Alternatives rejected due to overhead:
  - Extra page faults
  - Page table scans
- Overhead is only one incr/decr per mapping/unmapping op

count	n_0	n_1	…	n_{15}
p_0	108	2		17
…				
p_n	193	240		4322

Implementation Details

Problem:

DLLs and files shared by multiple processes (buffer cache) become scattered all over memory with a straightforward assignment of incoming pages to process's active nodes – large energy footprints afterall.

Implementation Details

Solutions:

- DLL Aggregation
  - Special case DLLs by allocating Sequential first-touch in low-numbered nodes
- Migration
  - Kernal thread – kmigrated – running in background when system is idle (waking up every 3s)
  - Scans pages used by each process, migrating if conditions met
    - Private page not on
    - Shared page outside $\cap \rho_i$

Process	ρ	α							
syslog	**14**	**0**(3)	**8**(5)	**9**(51)	**10**(1)	**11**(1)	**13**(3)	**14**(76)	
login	**11**	**0**(12)	**8**(7)	**9**(112)	**11**(102)	**12**(5)	**14**(20)	**15**(1)	
startx	**13**	**0**(21)	**7**(12)	**8**(3)	**9**(7)	**10**(12)	**11**(25)	**13**(131)	**14**(43)
X	**12**	**0**(125)	**7**(23)	**8**(47)	**9**(76)	**10**(223)	**11**(19)	**12**(1928)	**13**(82)
		14(77)	**15**(182)						
sawfish	**10**	**0**(180)	**7**(5)	**8**(12)	**9**(1)	**10**(278)	**13**(25)	**14**(5)	**15**(233)
vim	**10,15**	**0**(12)	**9**(218)	**10**(5322)	**14**(22)	**15**(4322)			
...	...	...							

Table 2: A snapshot of processes' node usage pattern

Process	ρ	α						
syslog	**14**	**0**(108)	**1**(2)	**11**(13)	**14**(17)			
login	**11**	**0**(148)	**1**(4)	**11**(98)	**15**(9)			
startx	**13**	**0**(217)	**1**(12)	**13**(25)				
X	**12**	**0**(125)	**1**(417)	**9**(76)	**11**(793)	**12**(928)	**13**(169)	**14**(15)
sawfish	**10**	**0**(193)	**1**(281)	**10**(179)	**13**(25)	**14**(11)	**15**(50)	
vim	**10,15**	**0**(12)	**1**(240)	**10**(5322)	**15**(4322)			
...	...	...						

Table 3: Effect of aggregating pages used by DLLs.

Process	ρ	α			
syslog	**14**	**0**(15)	**14**(125)		
login	**11**	**0**(76)	**11**(183)		
startx	**13**	**0**(172)	**13**(82)		
X	**12**	**0**(225)	**1**(2)	**12**(2220)	
sawfish	**10**	**0**(207)	**1**(56)	**10**(436)	
vim	**10,15**	**0**(12)	**1**(240)	**10**(5322)	**15**(4322)
...	...	...			

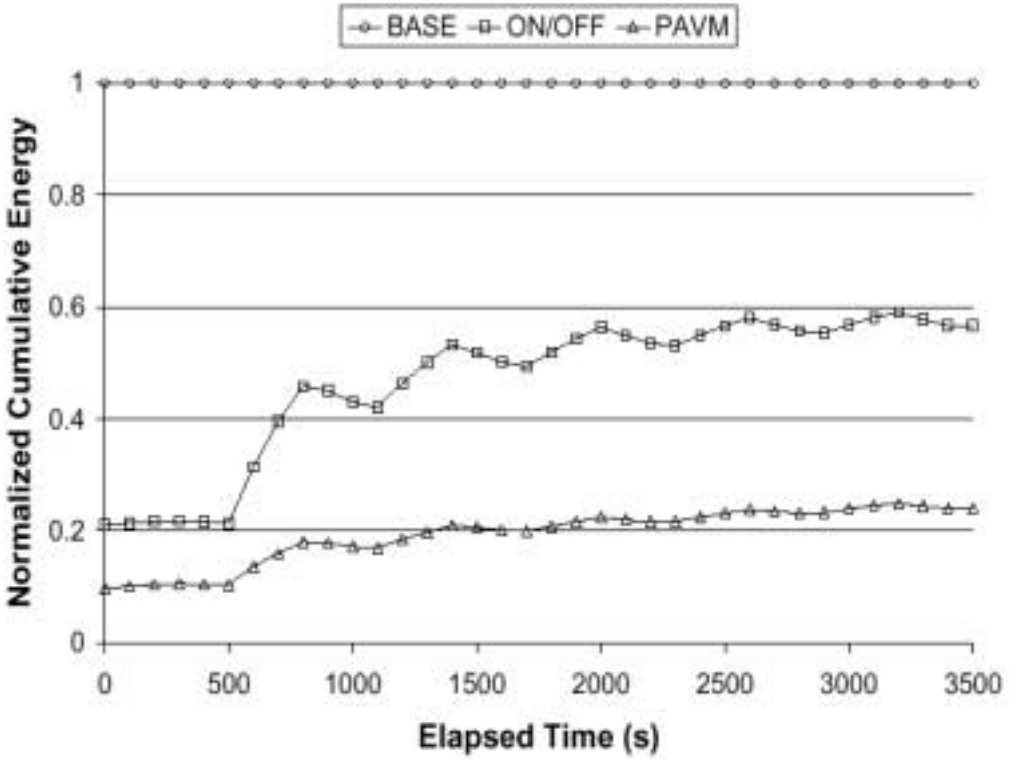
Table 4: Effect of library aggregation with page migration.

Evaluation Methodology

- Linux implementation
- Measurements/counts taken of events and energy results calculated (not measured)
- Metric – energy used by memory (only).
- Workloads – 3 mixes: light (editting, browsing, MP3), poweruser (light + kernel compile), multimedia (playing mpeg movie)
- Platform – 16 nodes, 512MB of RDRAM
- Not considered: DMA and kernel maintenance threads

Results

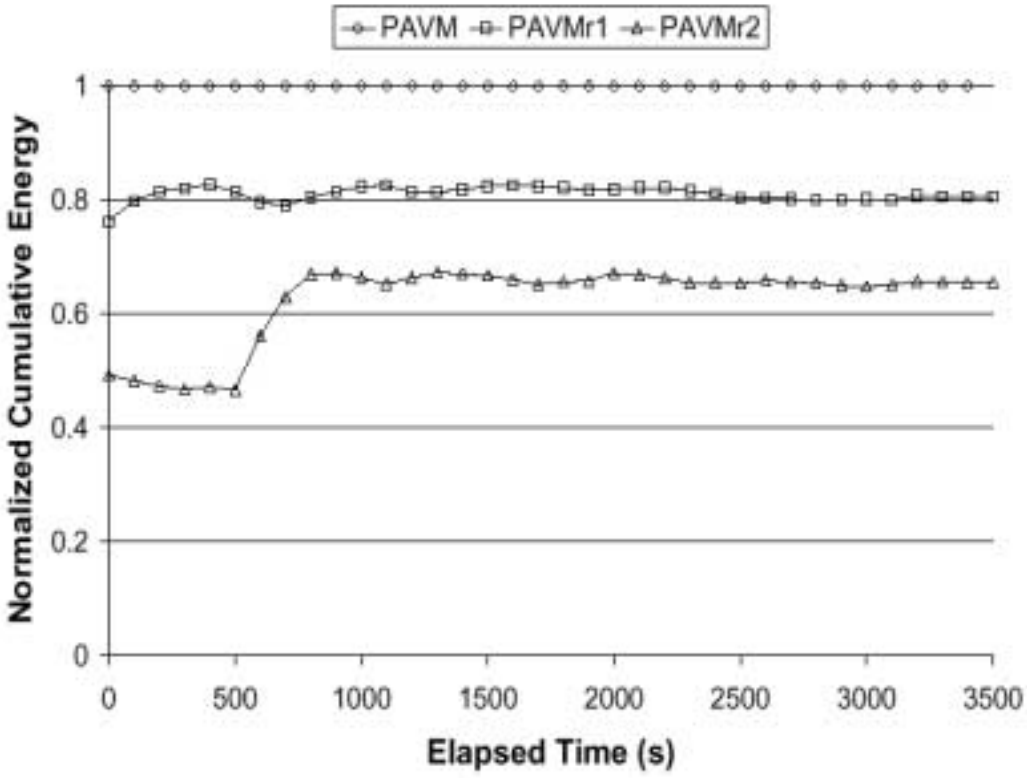
- Base – standby when not accessing
- On/Off – nap when system idle
- PAVM



(b) Poweruser workload

Results

- PAVM
- PAVMr1 - DLL aggregation
- PAVMr2 – both DLL aggregation & migration



(b) Poweruser workload

Results

	Light	Poweruser	Multimedia
Base	4100 mW	4118 mW	4230 mW
On/Off	892 mW	2324 mW	3991 mW
PAVM	465 mW	986 mW	2687 mW
PAVMr1	397 mW	791 mW	2442 mW
PAVMr2	237 mW	646 mW	1725 mW

Conclusions

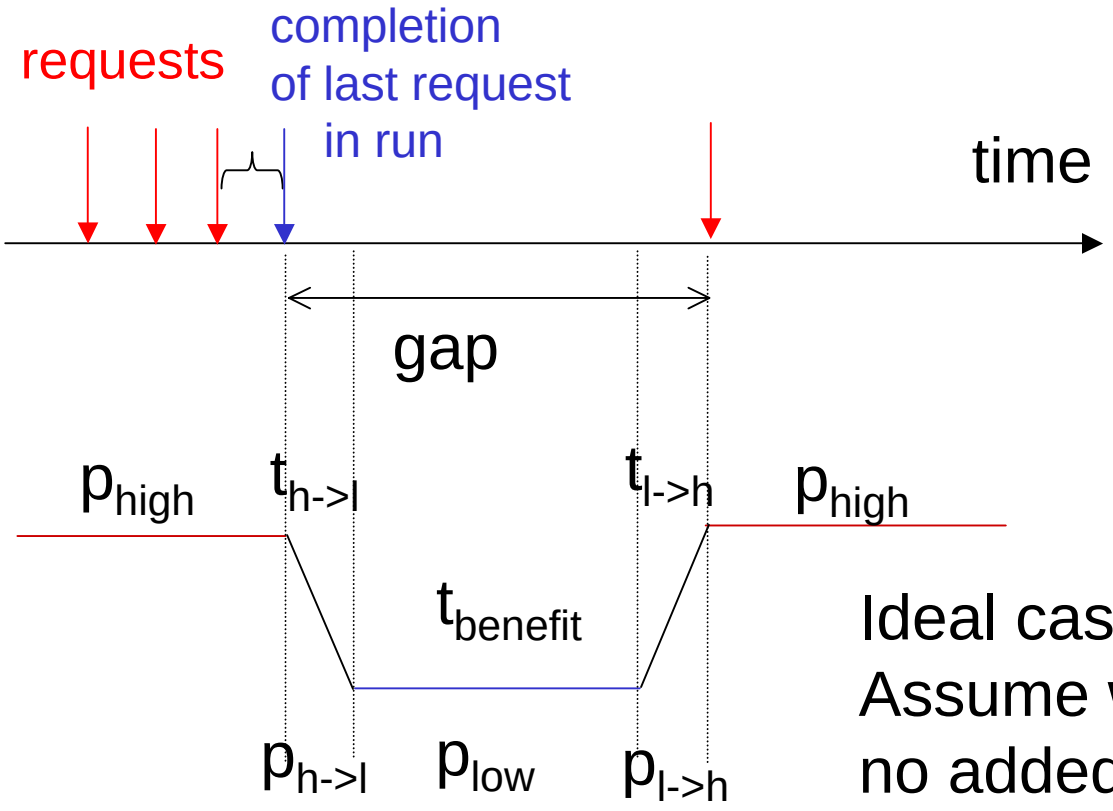
- Multiprogramming environment.
- Basic PAVM: save 34-89% energy of 16 node RDRAM
- With optimizations: additional 20-50%
- Works with other kinds of power-aware memory devices

Discussion: What about page replacement policies? Should (or *how* should) they be power-aware?

Related Work

- Lebeck et al, ASPLOS 2000 – dynamic hardware controller policies and page placement
- Fan et al
  - ISPLED 2001
  - PACS 2002
- Delaluz et al, DAC 2002

Power State Transitioning



Ideal case:
Assume we want
no added latency

$$(t_{h->l} + t_{l->h} + t_{benefit}) * p_{high} > t_{h->l} * p_{h->l} + t_{l->h} * p_{l->h} + t_{benefit} * p_{low}$$

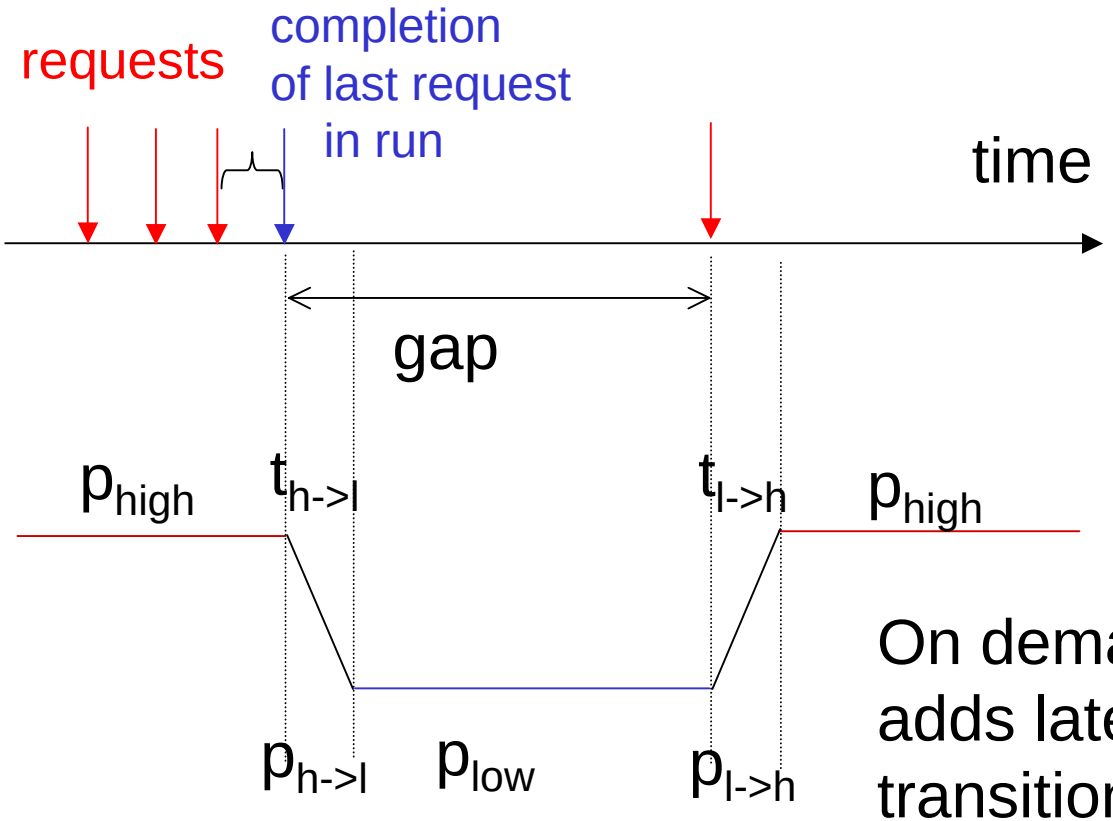
constant

Benefit Boundary

$$t_{benefit} > \frac{t_{h->l} * p_{h->l} + t_{l->h} * p_{l->h} - (t_{h->l} + t_{l->h}) * p_{high}}{(p_{high} - p_{low})}$$

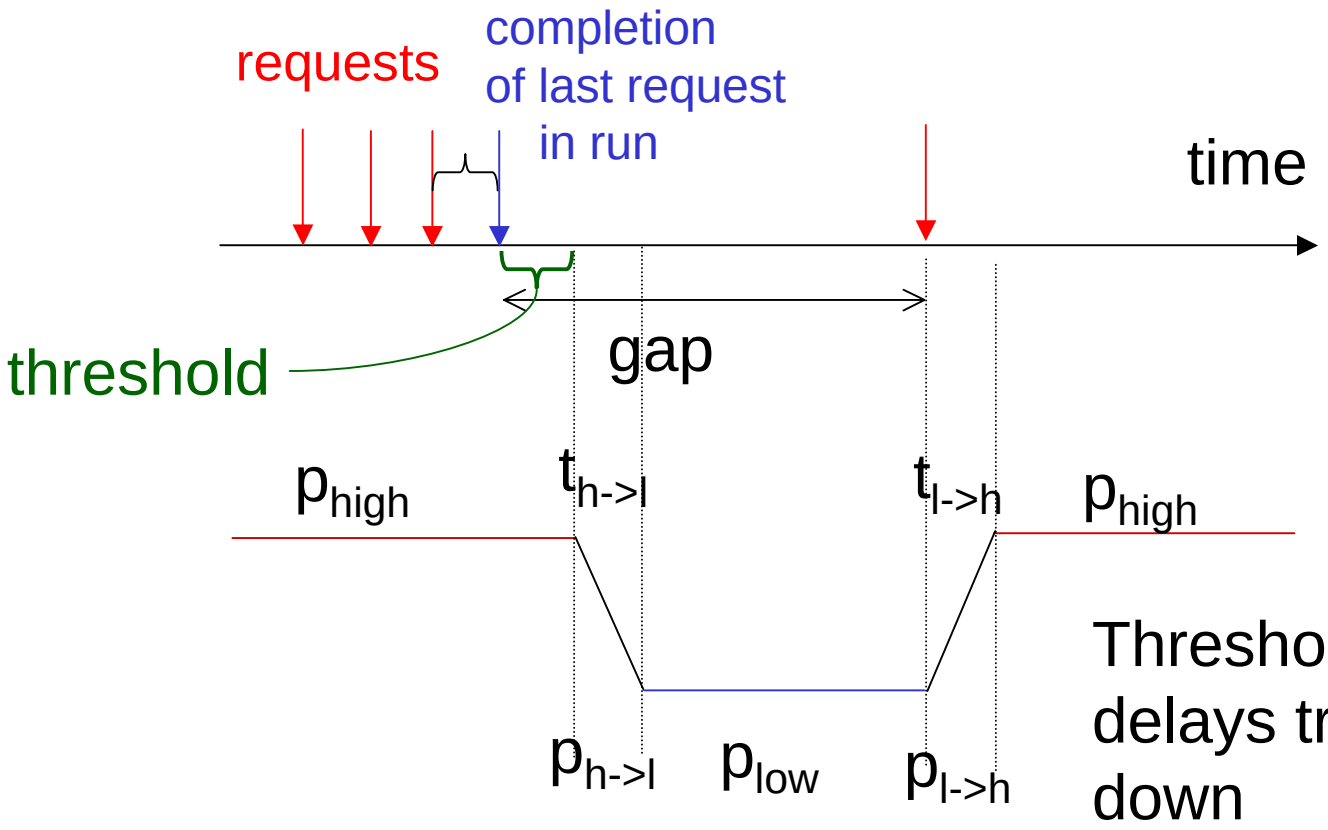
$$gap \geq t_{h->l} + t_{l->h} + t_{benefit}$$

Power State Transitioning



On demand case- adds latency of transition back up

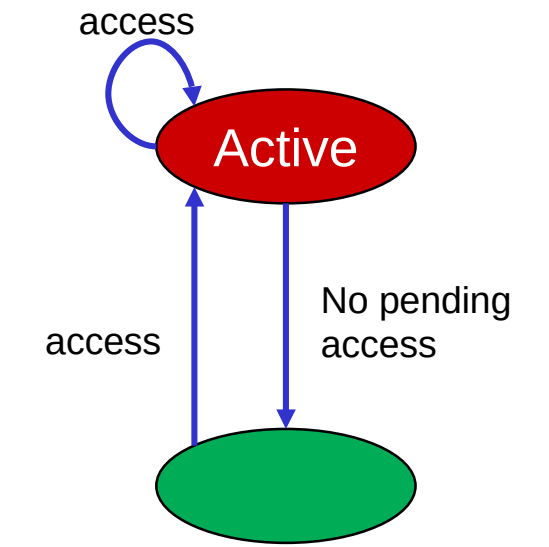
Power State Transitioning

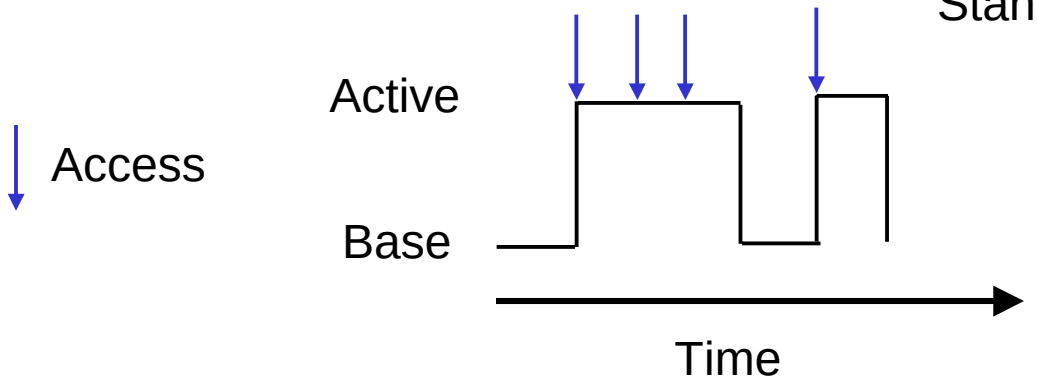


Threshold based-delays transition down

Dual-state HW Power State Policies

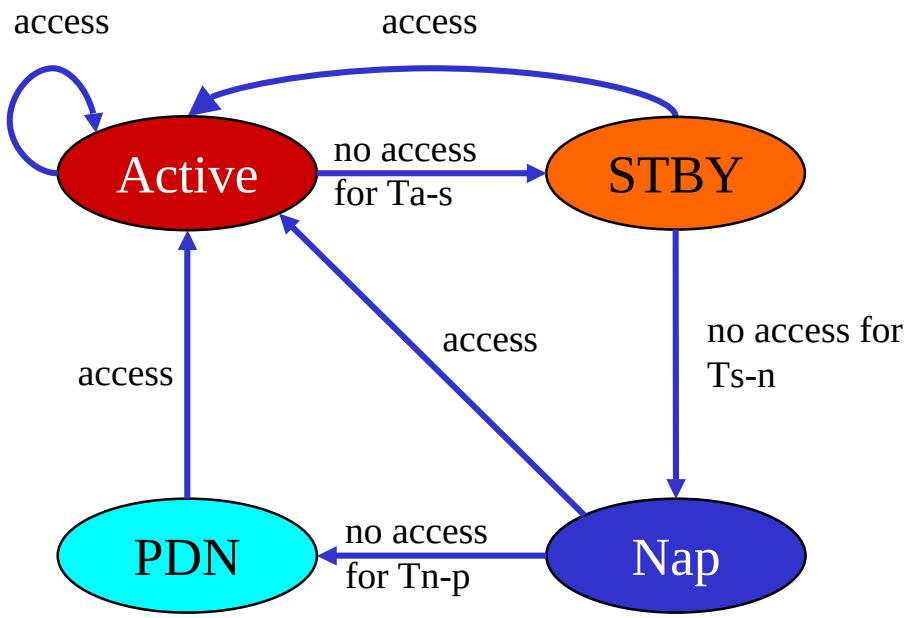
- All chips in one base state
- Individual chip Active while pending requests
- Return to base power state if no pending access

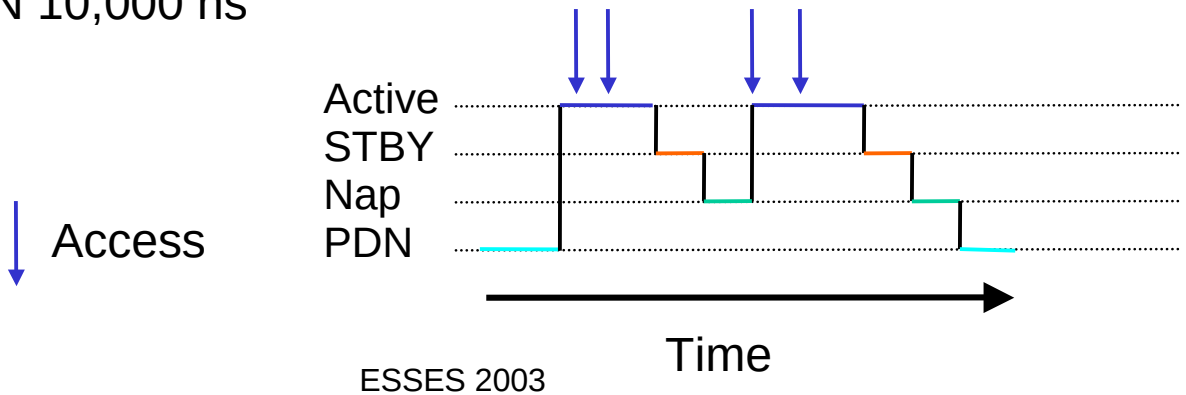




Quad-state HW Policies

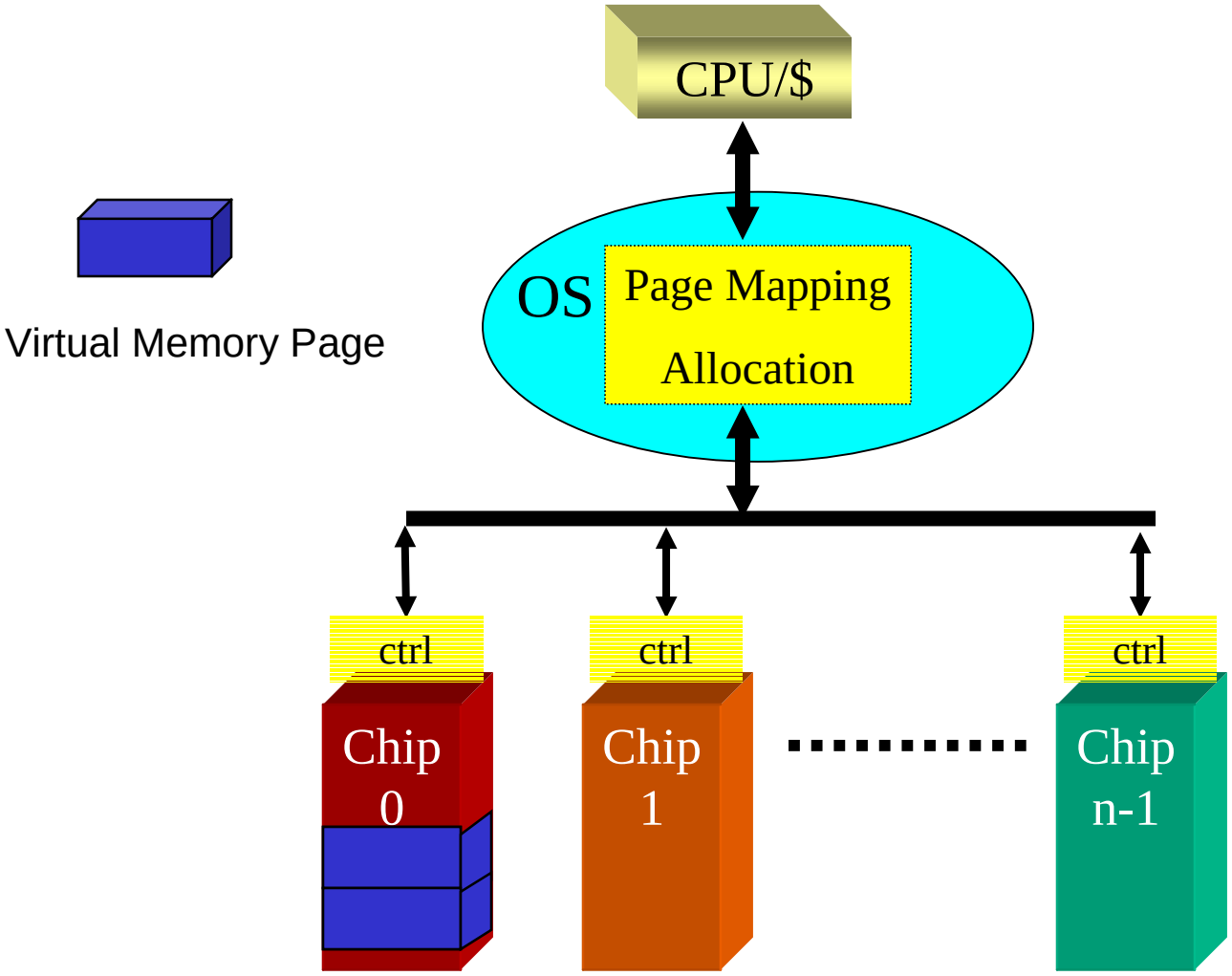
- Downgrade state if no access for threshold time
- Independent transitions based on access pattern to each chip
- Competitive Analysis
  - rent-to-buy
  - Active to nap 100's of ns
  - Nap to PDN 10,000 ns





Access

Page Allocation and Power-Aware DRAM



- Physical address determines which chip is accessed
- Assume non-interleaved memory
  - Addresses 0 to N-1 to chip 0, N to 2N-1 to chip 1, etc.
- Entire virtual memory page in one chip
- Virtual memory page allocation influences chip-level locality

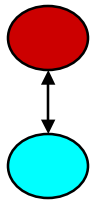
Page Allocation Polices

Virtual to Physical Page Mapping

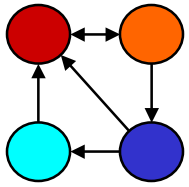
- Random Allocation – baseline policy
  - Pages spread across chips
- Sequential First-Touch Allocation
  - Consolidate pages into minimal number of chips
  - One shot
- Frequency-based Allocation
  - First-touch not always best
  - Allow (limited) movement after first-touch

The Design Space

	Random Allocation	Sequential Allocation
Dual-state Hardware	1 Simple HW	2 Can the OS help?
Quad-state Hardware	3 Sophisticated HW	4 Cooperative HW & SW



2 state model



4 state model

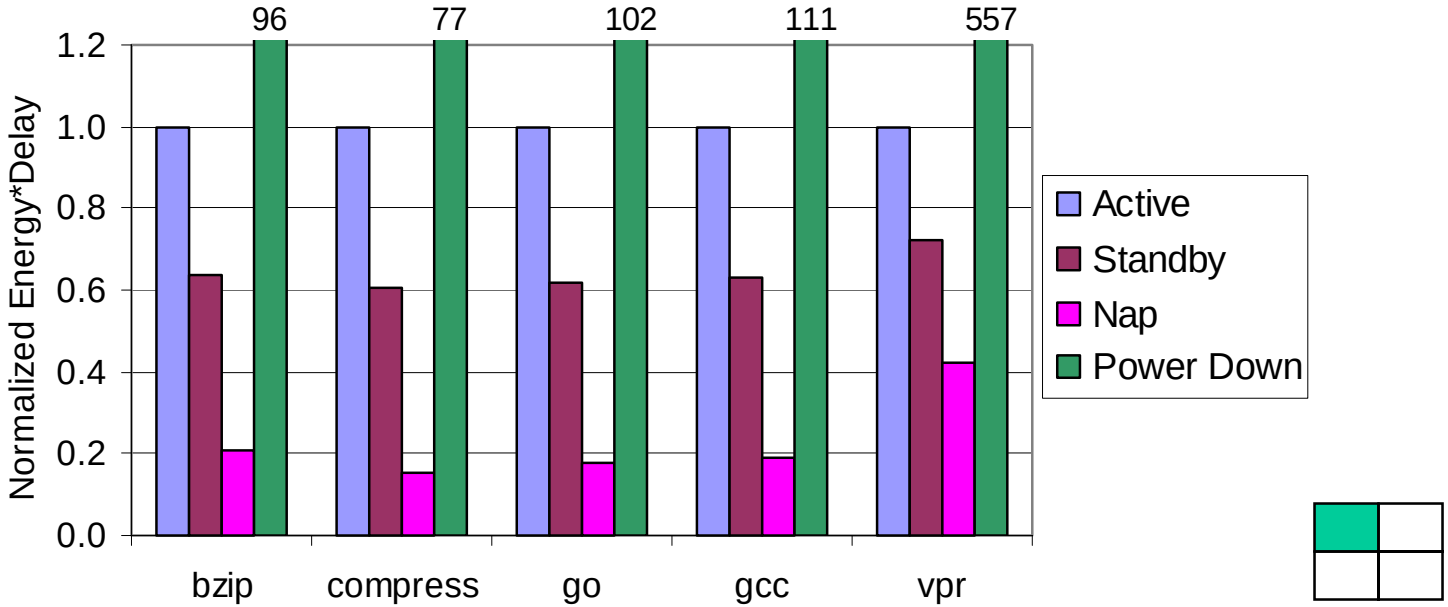
Methodology

- Metric: Energy*Delay Product
  - Avoid very slow solutions
- Energy Consumption (DRAM only)
  - Processor & Cache affect runtime
  - Runtime doesn't change much in most cases
- 8KB page size
- L1/L2 non-blocking caches
  - 256KB direct-mapped L2
  - Qualitatively similar to 4-way associative L2
- Average power for transition from lower to higher state
- Trace-driven and Execution-driven simulators

Methodology Continued

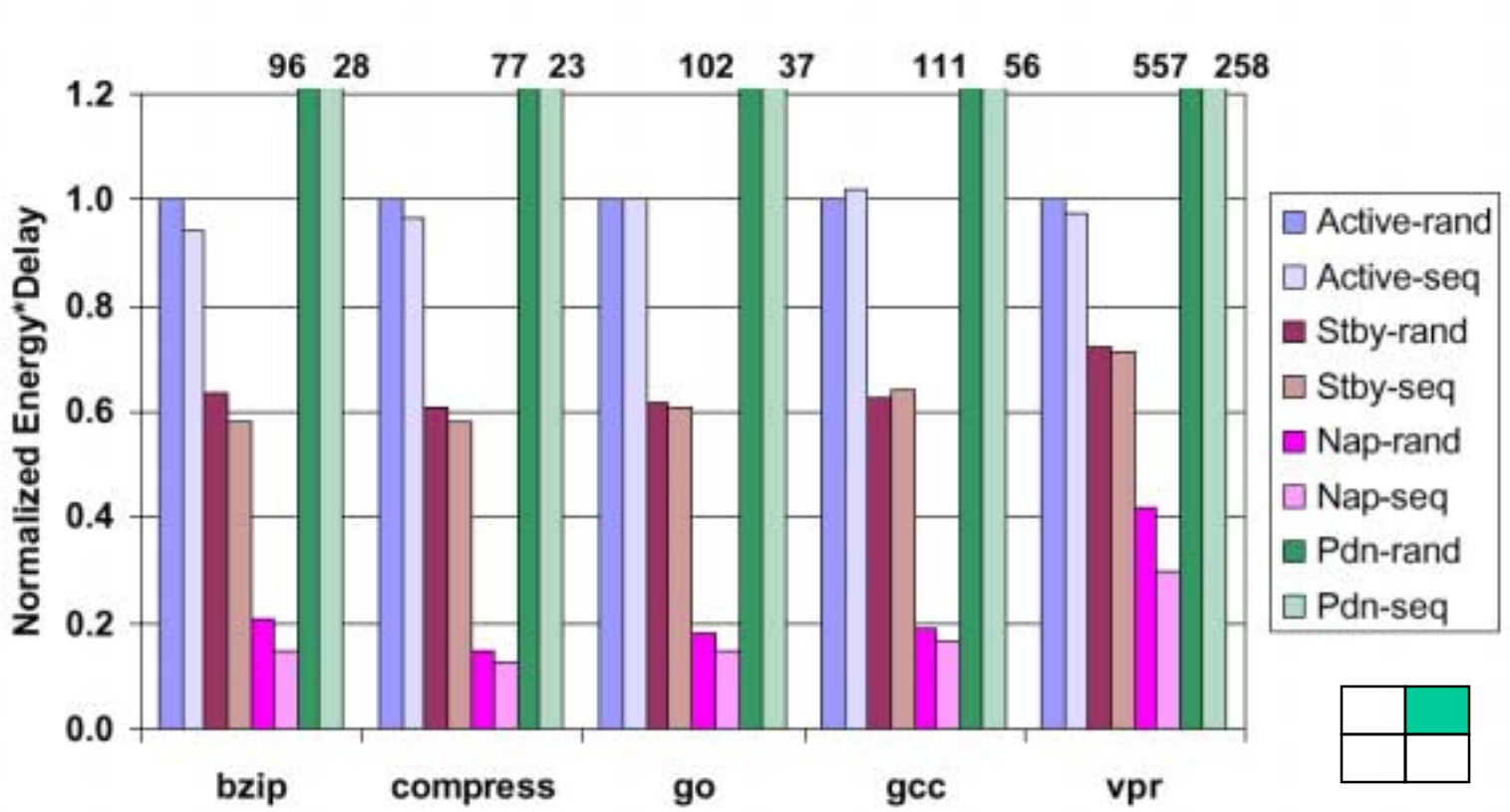
- Trace-Driven Simulation
  - Windows NT personal productivity applications (Etch at Washington)
  - Simplified processor and memory model
  - Eight outstanding cache misses
  - Eight 32Mb chips, total 32MB, non-interleaved
- Execution-Driven Simulation
  - SPEC benchmarks (subset of integer)
  - SimpleScalar w/ detailed RDRAM timing and power models
  - Sixteen outstanding cache misses
  - Eight 256Mb chips, total 256MB, non-interleaved

Dual-state + Random Allocation (SPEC)



- All chips use same base state
- Nap is best 60% to 85% reduction in E*D over full power
- Simple HW provides good improvement

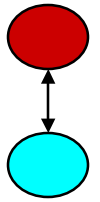
Benefits of Sequential Allocation (SPEC)



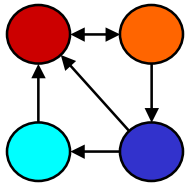
- 10% to 30% additional improvement for dual-state nap
- Some benefits due to cache effects

Results (Energy*Delay product)

	Random Allocation	Sequential Allocation
Dual-state Hardware	Nap is best 60%-85% improvement	10% to 30% improvement for nap. Base for future results
Quad-state Hardware	What about smarter HW?	Smart HW and OS support?

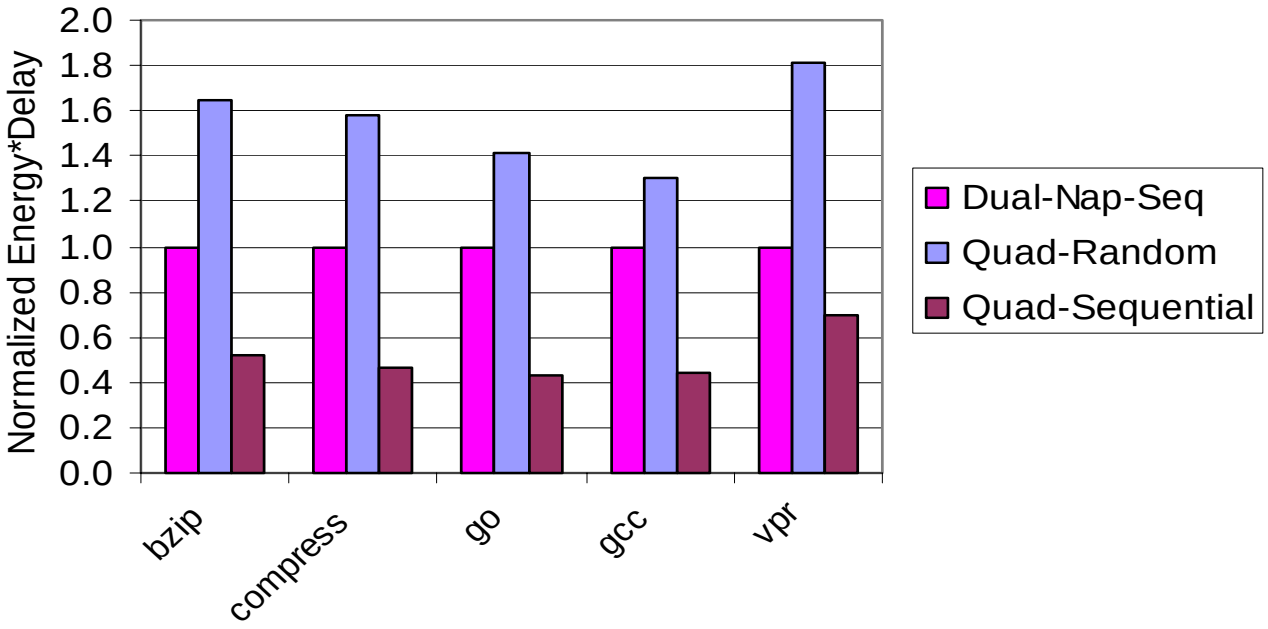


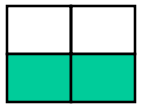
2 state model



4 state model

Quad-state HW (SPEC)



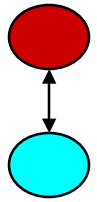


- Base: Dual-state Nap Sequential Allocation
- Thresholds: 0ns A->S; 750ns S->N; 375,000 N->P
- Quad-state + Sequential 30% to 55% additional improvement over dual-state nap sequential
- HW / SW Cooperation is important

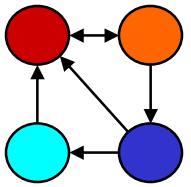
Summary of Results

(Energy*Delay product, RDRAM, ASPLOS00)

	Random Allocation	Sequential Allocation
Dual-state Hardware	Nap is best dual-state policy 60%-85%	Additional 10% to 30% over Nap
Quad-state Hardware	Improvement not obvious, Could be equal to dual-state	Best Approach: 6% to 55% over dual-nap-seq, 80% to 99% over all active.



2 state model



4 state model

Conclusion

- New DRAM technologies provide opportunity
  - Multiple power states
- Simple hardware power mode management is effective
- Cooperative hardware / software (OS page allocation) solution is best