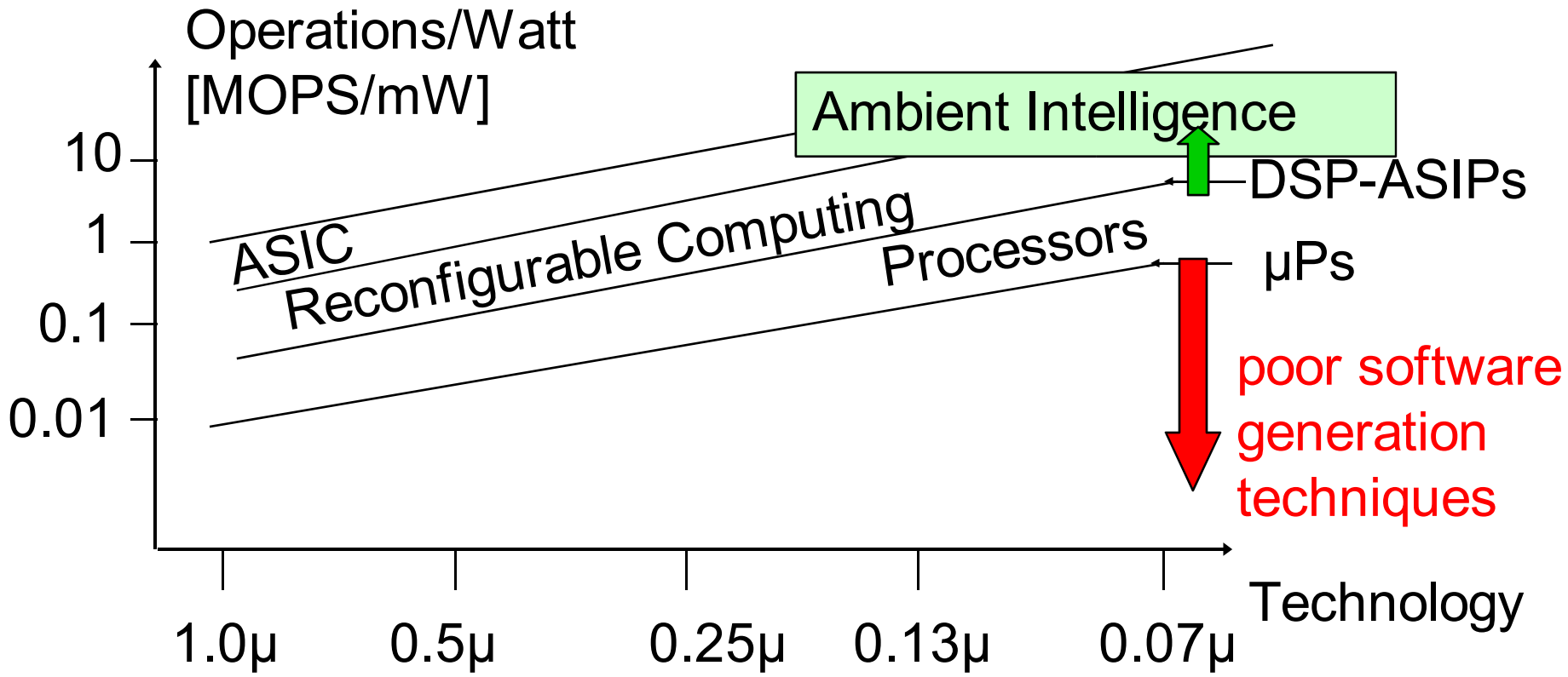
Exploitation of the memory hierarchy

Peter Marwedel
University of Dortmund, Germany

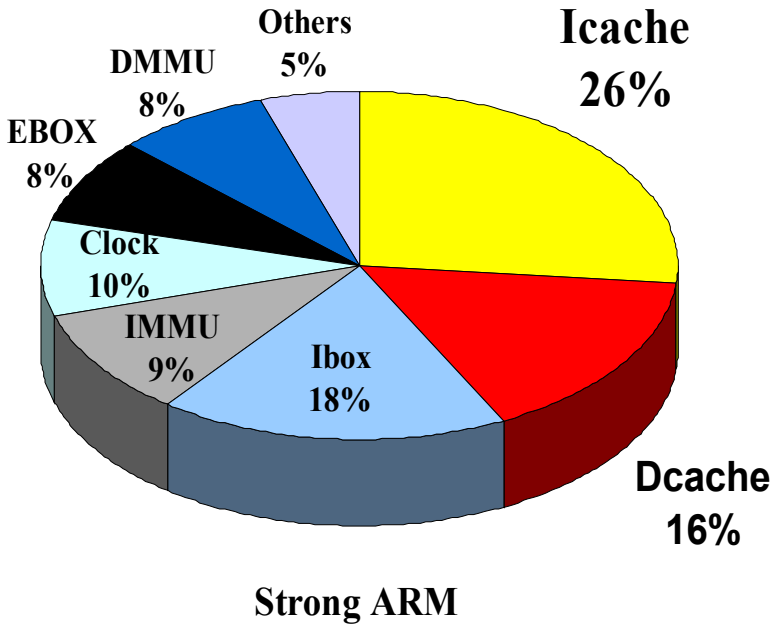
Facts about power consumption



Necessary to optimize software; otherwise the price for software flexibility cannot be paid!

[H. de Man, Keynote, DATE'02; T. Claasen, ISSCC99]

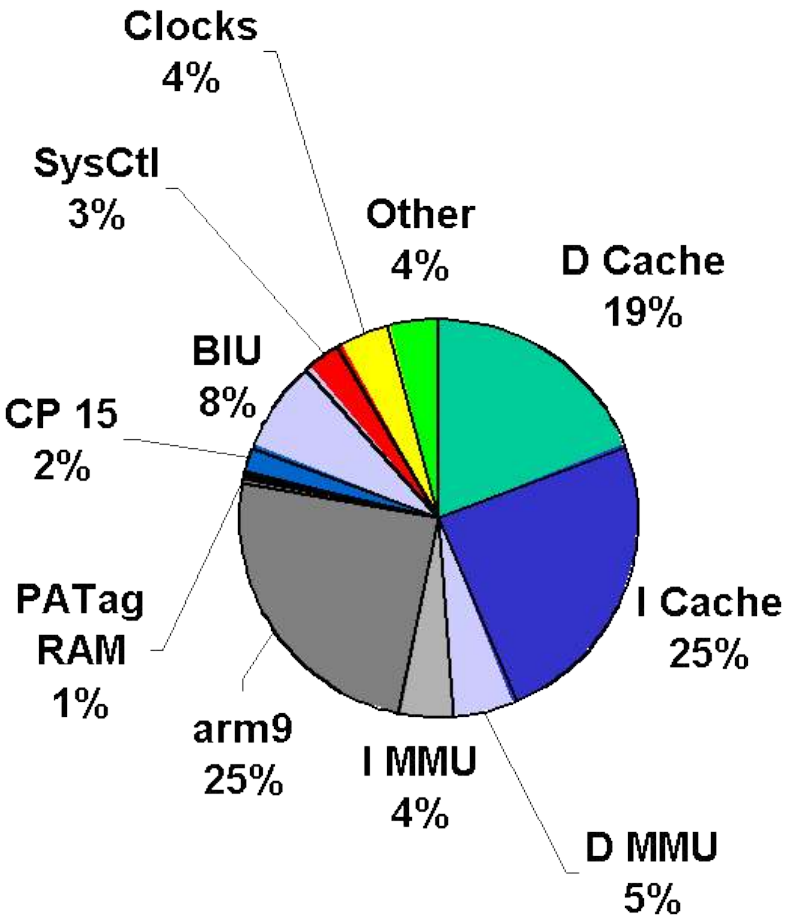
Major share of the power consumption due to the memory



Strong ARM

IEEE Journal of SSC
Nov. 96

Based on slide by and ©: Osman S. Unsal, Israel Koren, C. Mani Krishna, Csaba Andras Moritz, U. of Massachusetts, Amherst, 2001



[Segars 01 according to Vahid@ISSS01]

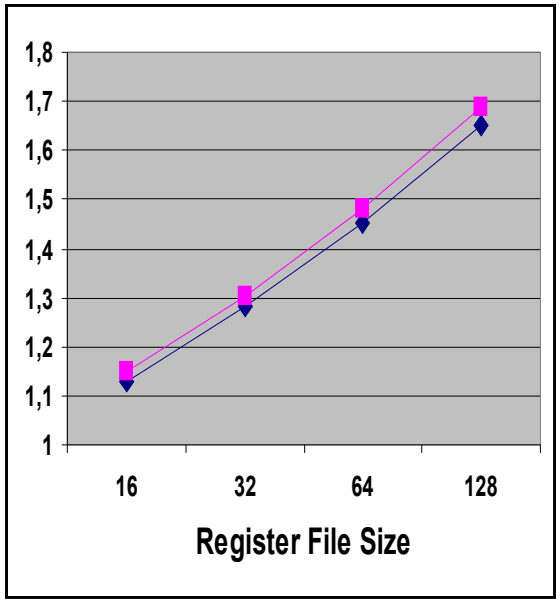
Multiple objectives for designing memory system

- (Average) Performance
  - Throughput 
  - Latency 
- Predictability, good worst case execution time bound (WCET) 
- Energy consumption 
- Size 
- Cost 

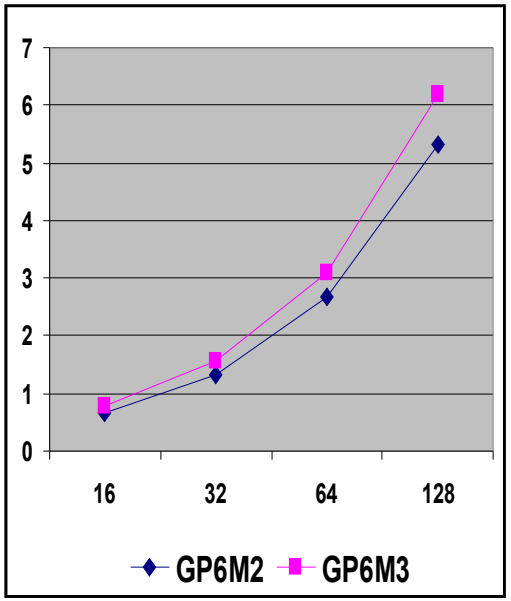
….

„Everything“ is large for large memories

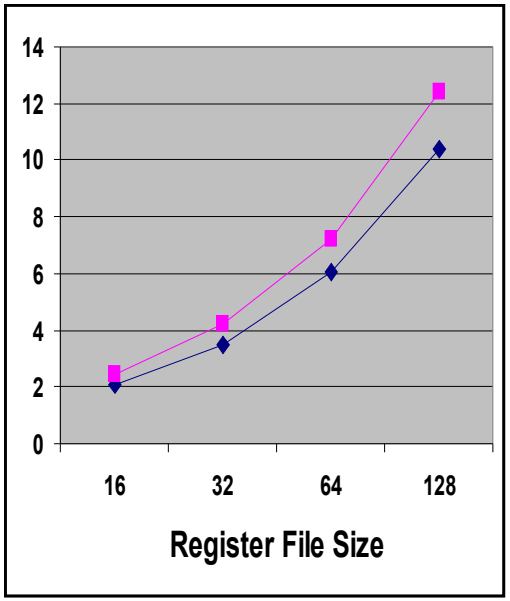
Cycle Time (ns)*



Area (λ^2x10^6)



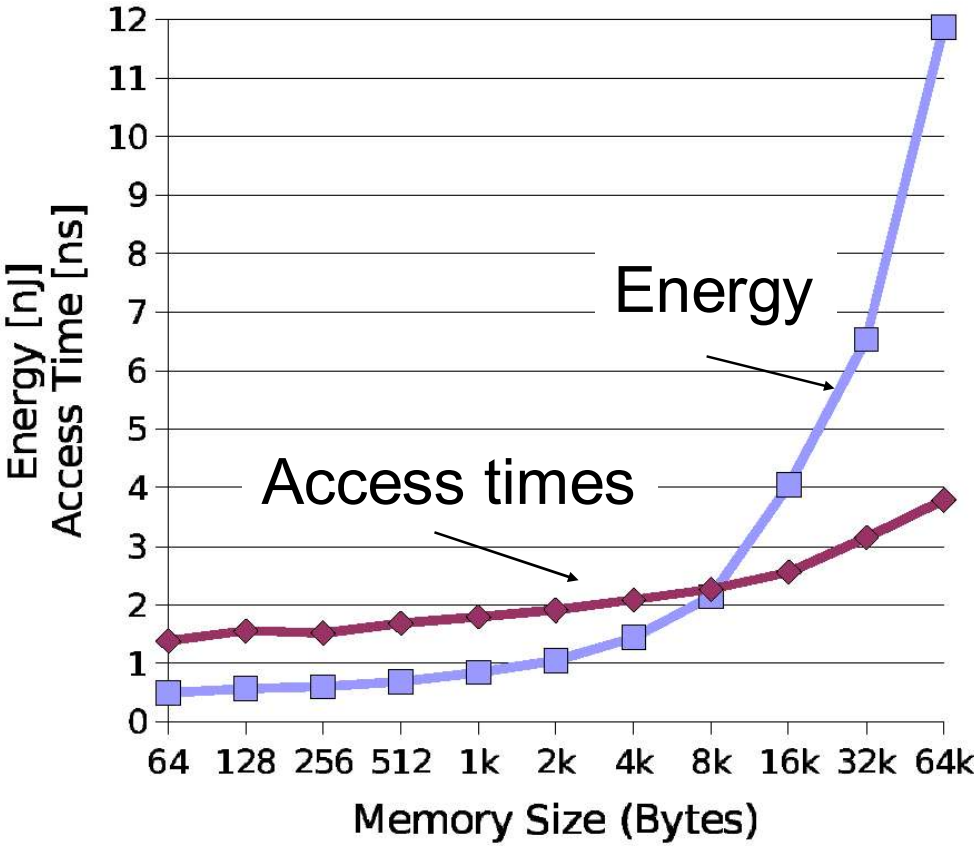
Power (W)



☞ Memory hierarchies

* Monolithic register file; Rixner’s et al. model [HPCA’00], Technology of 0.18 mm; VLIW configurations for a certain number of ports („GP*x*M*y*REG*z* where: x={6}, y={2, 3} and z={16, 32, 64, 128“}; Based on slide by and ©: Harry Valero, U. Barcelona, 2001

Same problem for a larger range of sizes

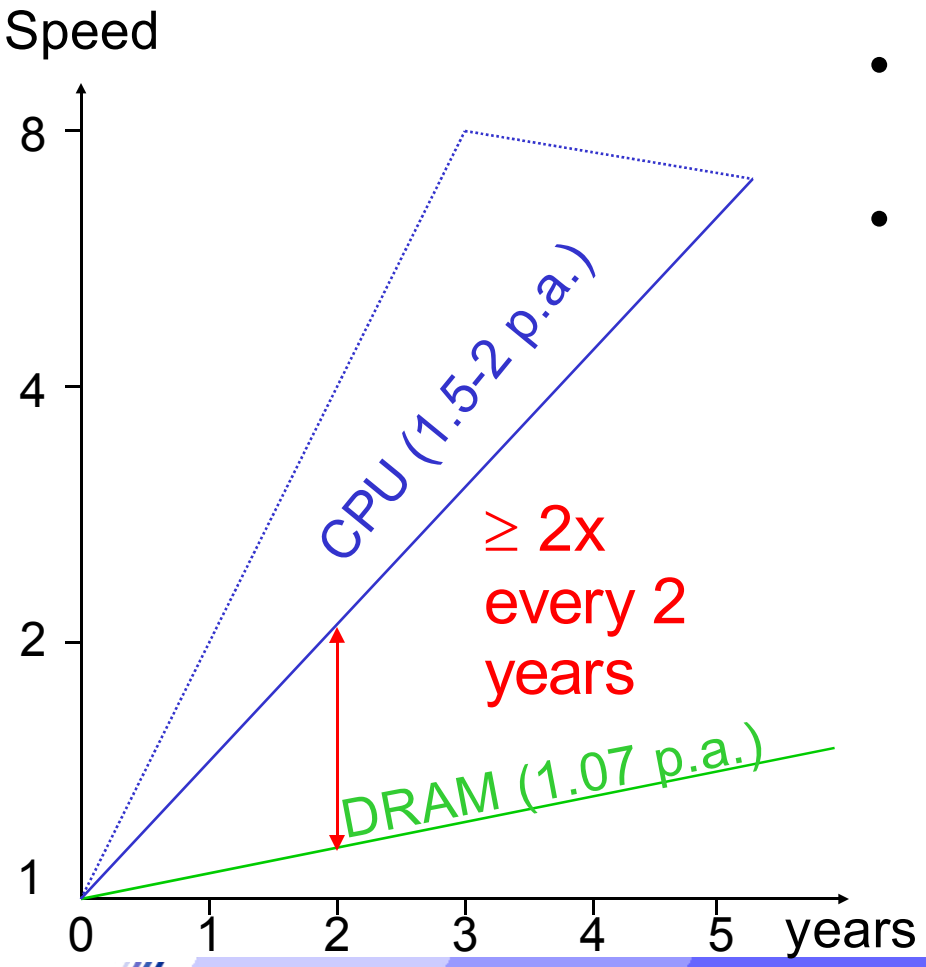


In the future: Memory access times >> processor cycle times

☞ "Memory wall" problem

Trends for the speeds

Speed gap between processor and main DRAM increases



- early 60ties (Atlas): page fault ~ 2500 instructions
- 2002 (2 GHz µP): access to DRAM ~ 500 instructions
  ☞ penalty for cache miss about the same as for page fault in Atlas

[P. Machanik: Approaches to Addressing the Memory Wall, TR Nov. 2002, U. Brisbane]

Approaches for improving latency, throughput, energy efficiency and predictability

Memory Technologies:

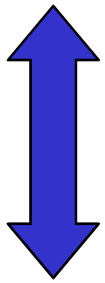
- RAMBUS DRAM, Sub-banking
- Magnetic RAM (energy efficient, high performance, ..)
- …

Memory Architectures:

- Caches (reconfigurable caches, way-locking …)
- Loop Caches
- Scratchpad Memories

Software Approaches:

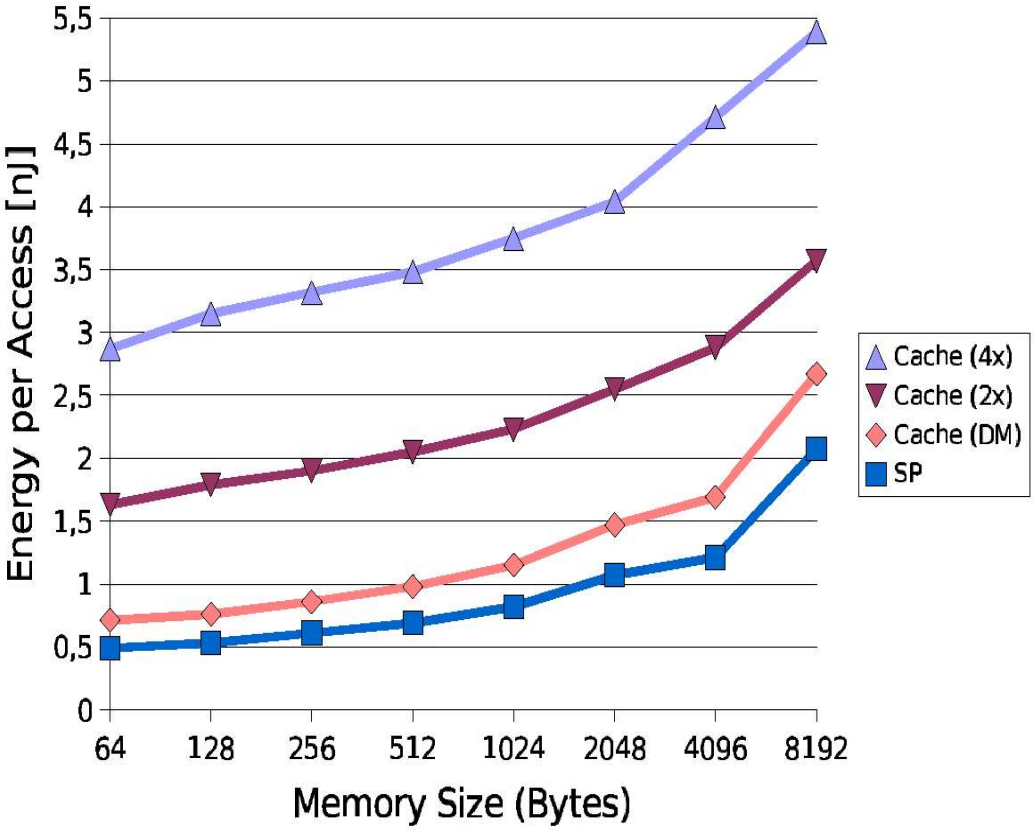
- Transformations to improve data locality



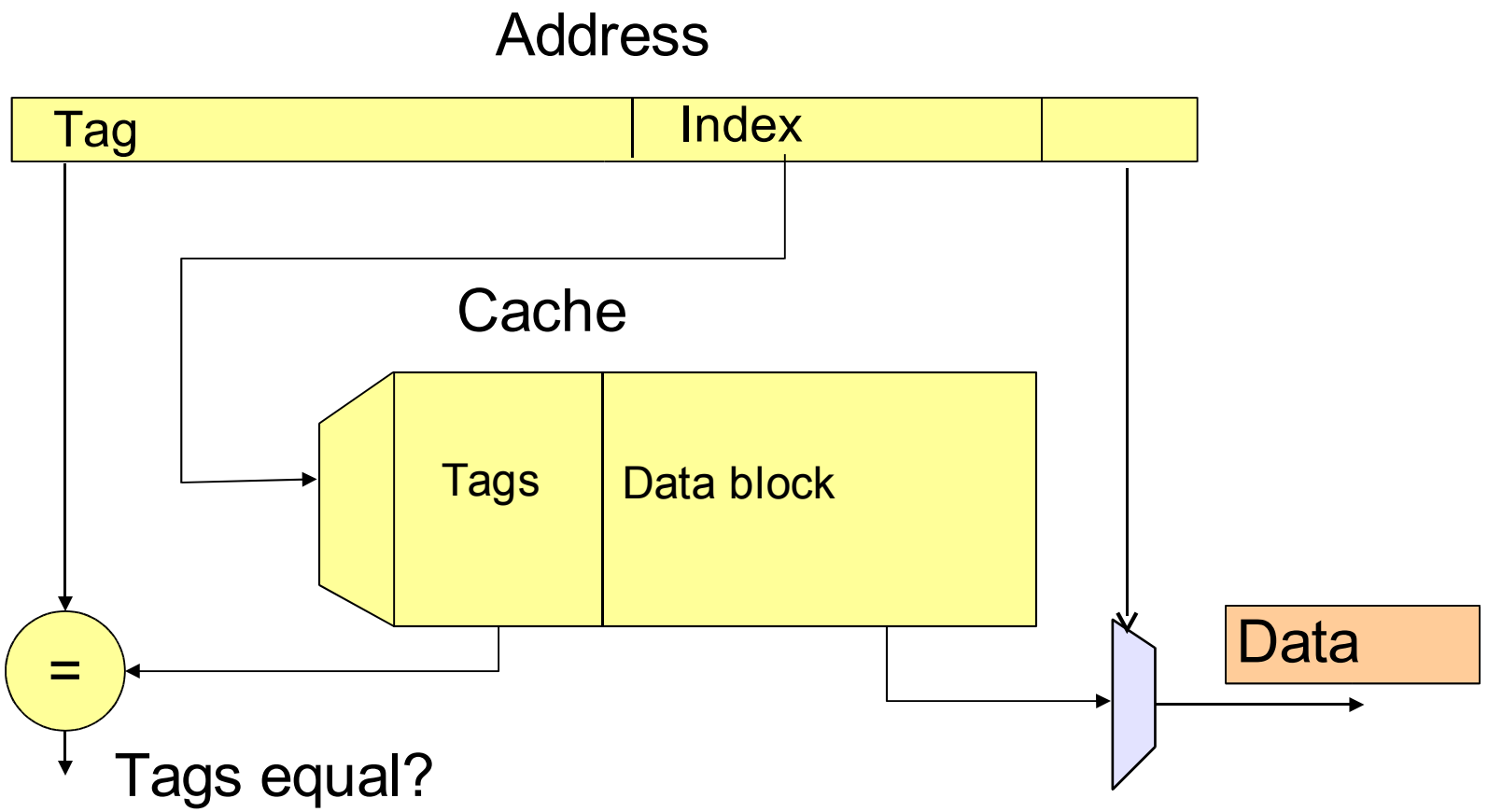
Focus

What's wrong with caches ? (1)

High associativity leads to high energy consumption

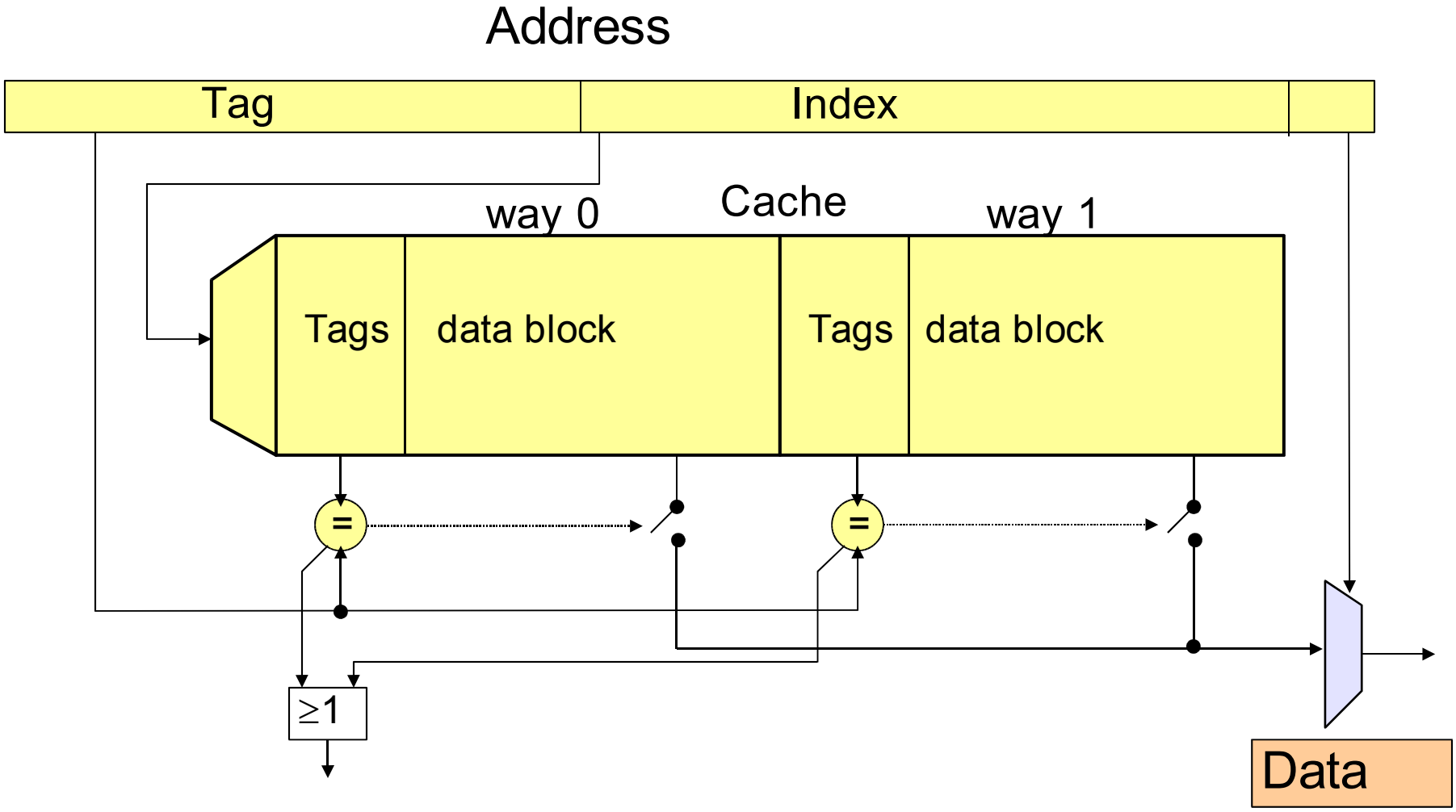


Fundamentals: Direct Mapped Caches



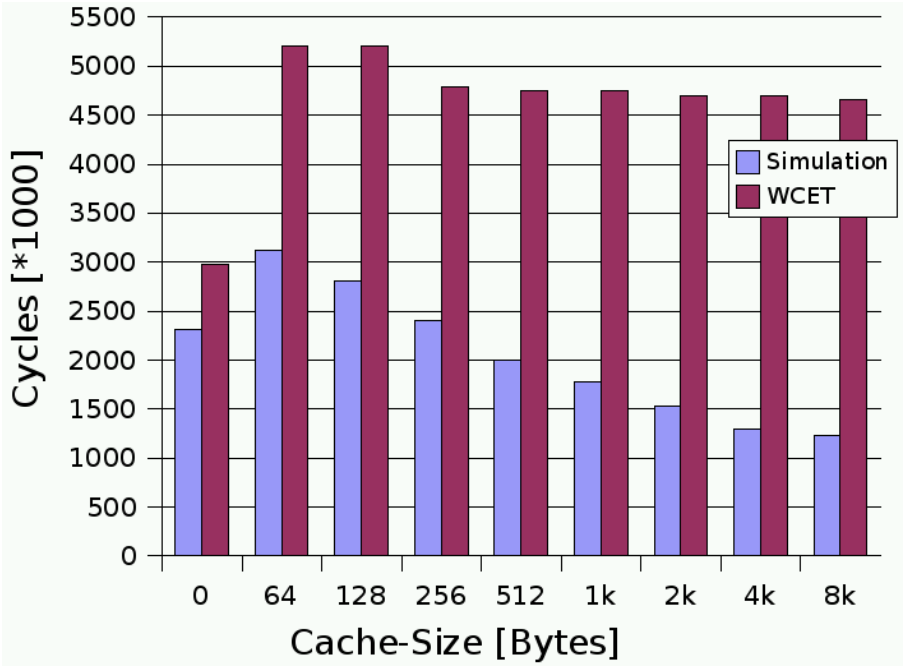
Recap: Set-associative cache
n-way cache

|Set| = 2



What's wrong with caches ? (2)

Caches have poor timing predictability



Worst case execution time (WCET) larger than without cache

G.721: using unified Cache@ARM7TDMI

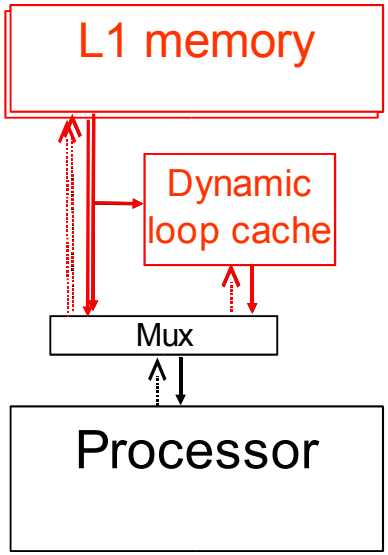
Proposal for avoiding limitations of caches: "Cool cache"

- Essentially a technology for selecting the ways during replacement. Otherwise, cache remains unchanged.
- Based on including way-identifiers in virtual to real-address translation (does not work without virtual addressing).
- Predictable presence of data in certain ways (sets).

[O. S. Unsal, R. Ashok, I. Koren, C. M. Krishna, and C. A. Moritz: Cool-Cache: A Compiler-Enabled Energy Efficient Data Caching Framework for Embedded/ Multimedia Processors, ACM Transactions on Embedded Computing Systems, Vol. 2, No. 3, 2003, Pages 373–392.]

Dynamically Loaded Loop Cache

- Small tagless loop cache
- Alternative location to fetch instructions
- Dynamically fills loop cache
  - Triggered by short backwards branch (sbb) instruction
- Flexible variation
  - Loops > |loop cache| can be partially stored
- Limitation: does not support loops with control of flow changes (cof)



[based on slide by F. Vahid, ISSS01]

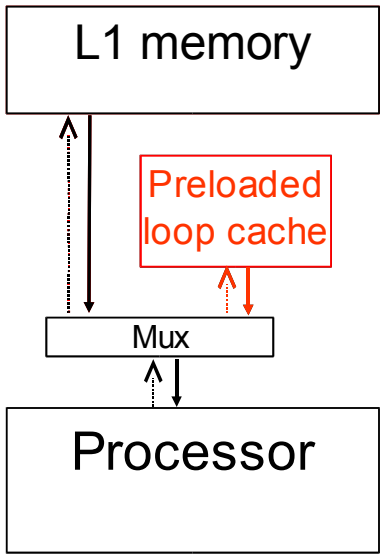
```
...
add r1,2
...
sbb -5
```

Iteration 1: detect sbb instruction
Iteration 2 : fill loop cache
Iteration 3 : fetch from loop cache

Works only for ICache

Preloaded Loop Cache

- Small tagless loop cache
- Alternative location to fetch instructions
- Loop cache filled at compile time and remains fixed
  - Supports loops with cof
- Fetch triggered by short backwards branch
- Start address variation
  - Fetch begins on first loop iteration




```
...
add r1,2
bne r1, r2, 3
...
sbb -5
```

Iteration 1: detect sbb instruction

Iteration 2 : check to see if loop preloaded, if so fetch from cache

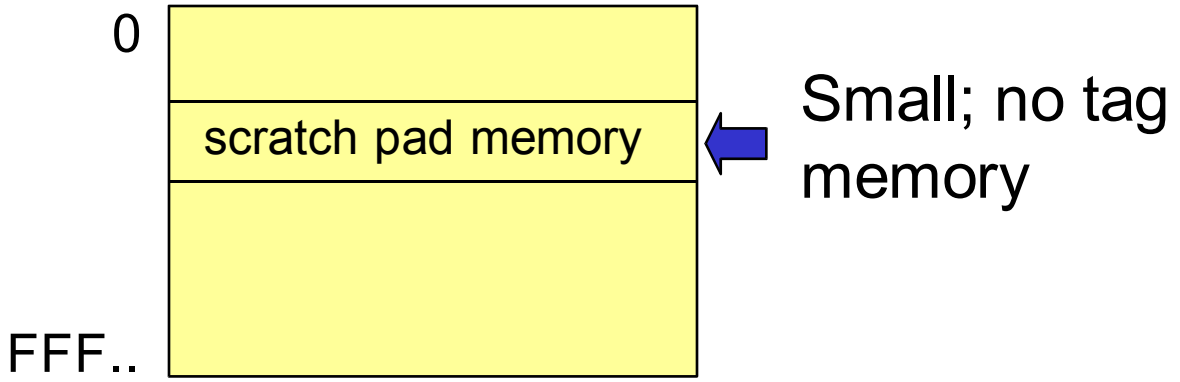
☞ "Caches behave almost like a scratch pad"

Why not try a scratch pad right away ?

[based on slide by F. Vahid, ISSS01]

Scratch pad memories (SPM): Fast, energy-efficient, timing-predictable

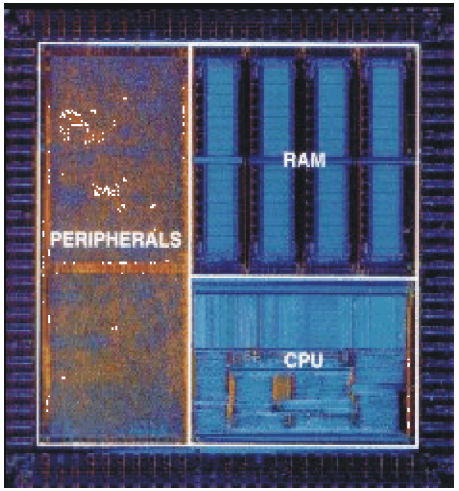
Address space



Small; no tag memory

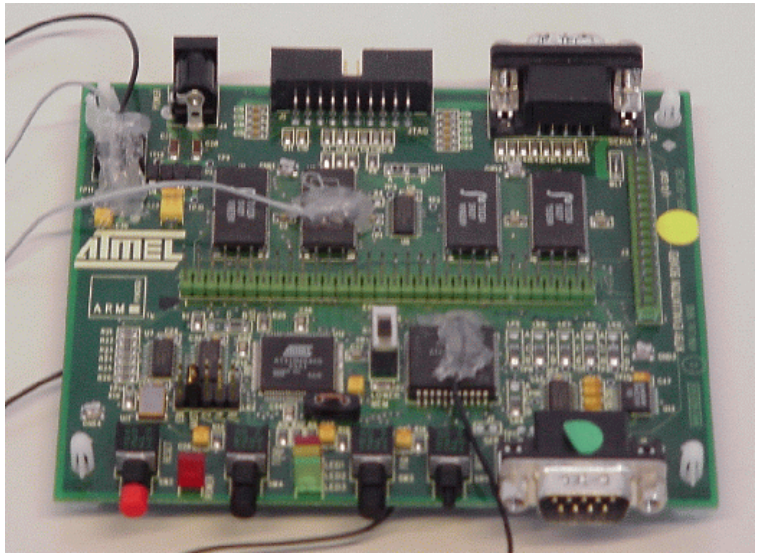
Example

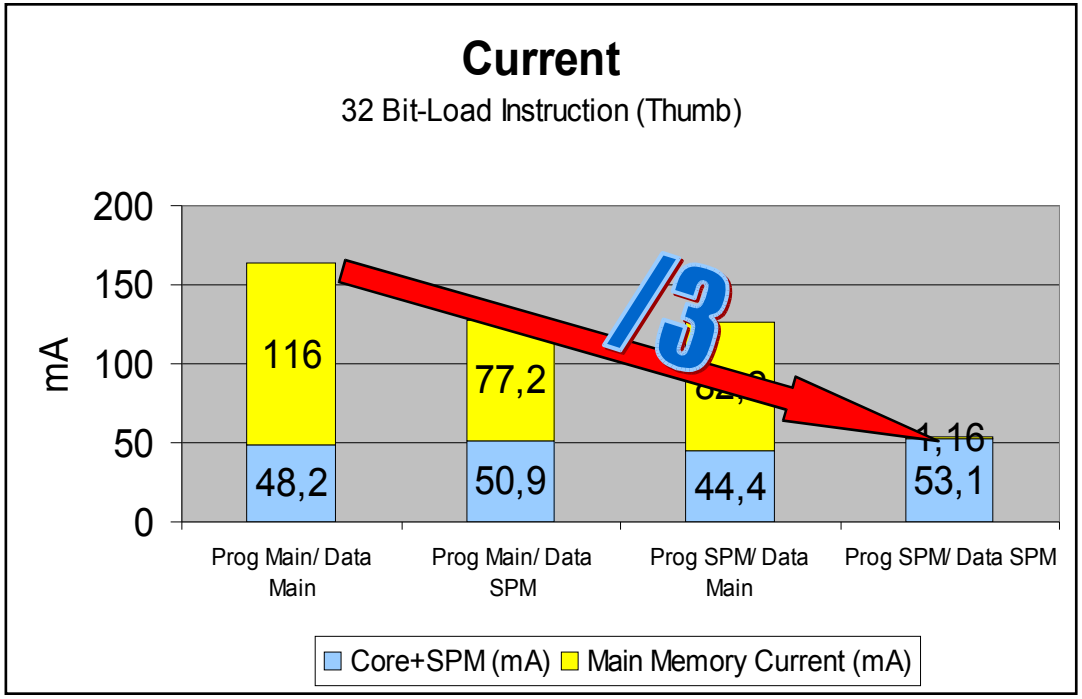
ARM7TDMI cores, well-known for low power consumption



Comparison of currents using measurements

E.g.: ATMEL board with ARM7TDMI and ext. SRAM





Availability of "Tightly coupled memories"

ARM CPU Core	Caches	TCM Available
ARM 1026EJ-S	Variable	yes
ARM 1136J(F)-S	Variable	yes
ARM 1176JZ(F)-S	Variable	yes
ARM 926EJ-S	Variable	yes
ARM 1026EJ-S	Variable	yes
ARM 1156T2(F)-S	Variable	yes
ARM 946E-S	Variable	yes
ARM 966E-S	-	yes
ARM 968E-S	-	yes
All others		no

Source: http://www.arm.com/products/CPUs/core_selector.html

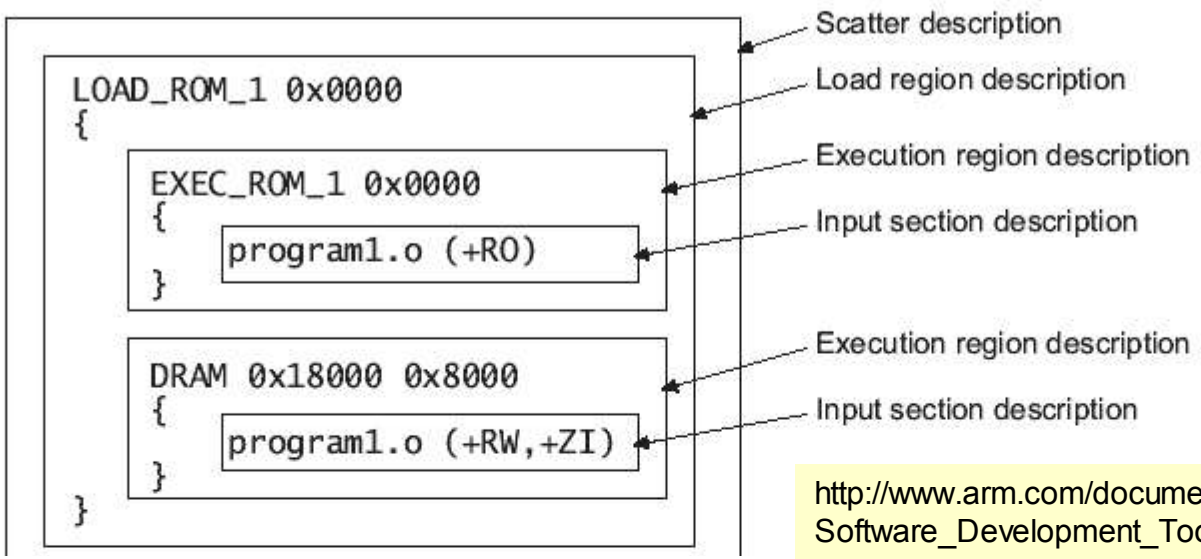
Using these ideas with an ARMcc-based tool flow

Use pragma in C-source to allocate to specific section:

For example:

```
#pragma arm section rwdata = "foo", rodata = "bar"
int x2 = 5; // in foo (data part of region)
int const z2[3] = {1,2,3}; // in bar
```

Input scatter loading file to linker for allocating section to specific address range



Too complicated!

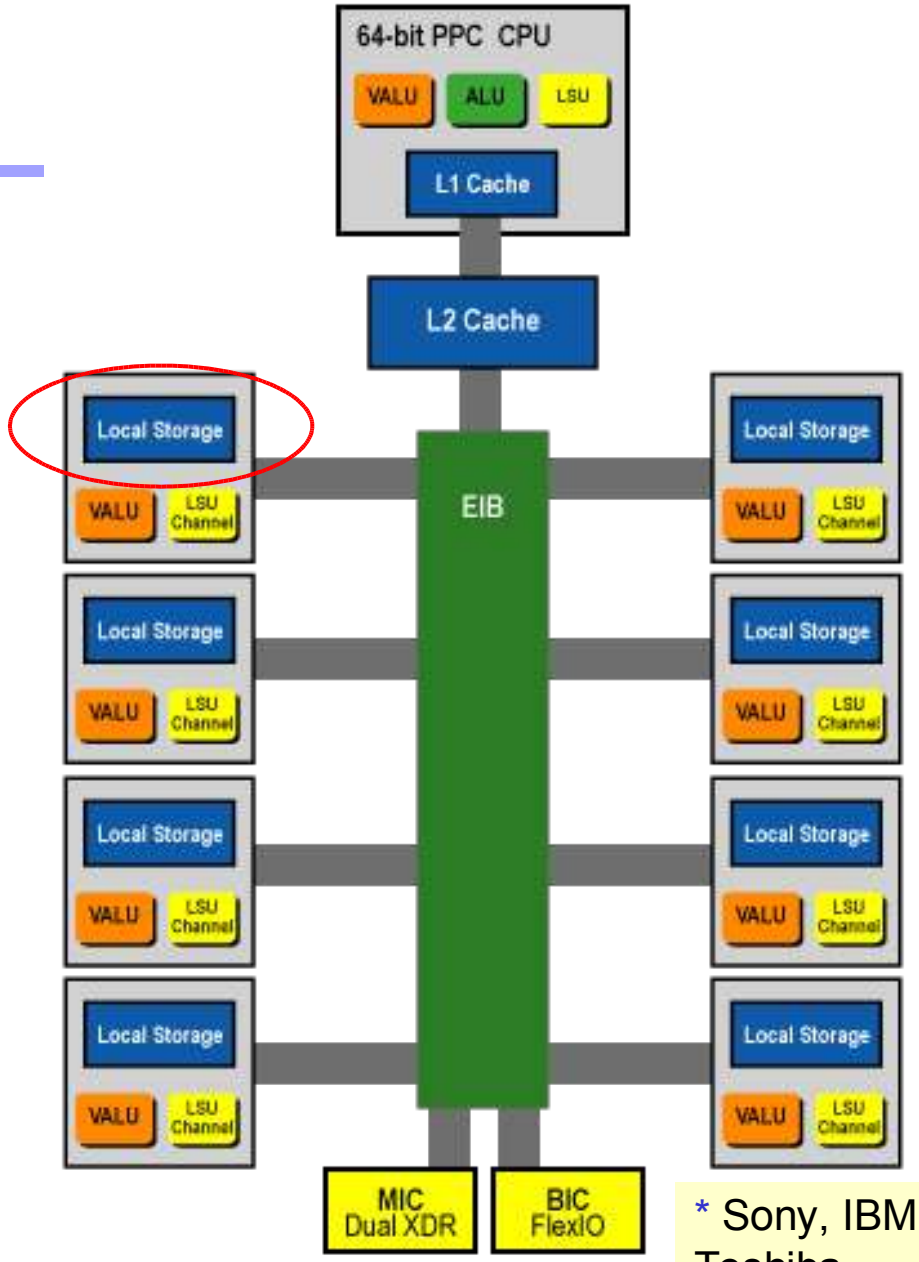
http://www.arm.com/documentation/Software_Development_Tools/index.html

Cell processor *

Local SPE processors fetch instructions and data from local storage LS (256 kB). LS **not** designed as a cache. Separate DMA transfers required to fill and spill.

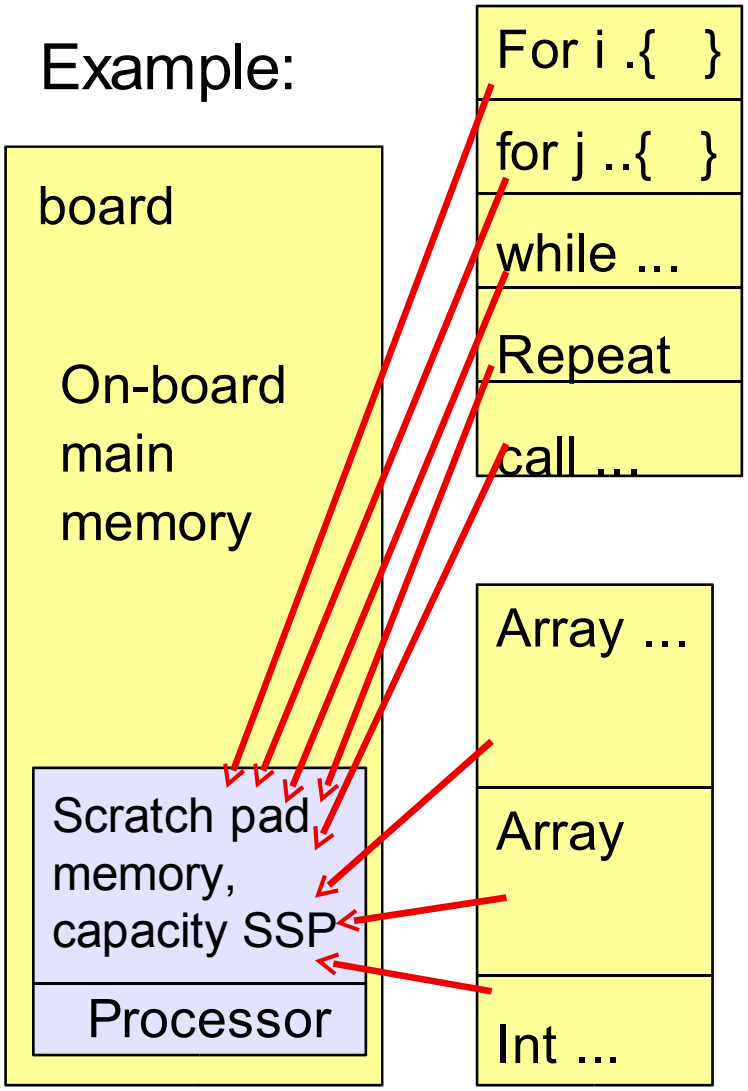
Motivation same as for this tutorial:

- Large memory latency
- Huge overhead for automatically managed caches



* Sony, IBM, Toshiba

Migration of data and instructions, global optimization model (U. Dortmund)



Which segment (array, loop, etc.) to be stored in SPM?

Gain g_k and size s_k for each segment *k*.

Maximise gain $G = \Sigma g_k$, respecting size of SPM SSP $\geq \Sigma\, s_k$.

Static memory allocation:

Solution: knapsack algorithm.

Dynamic reloading:

Finding optimal reloading points.

IP representation
- migrating functions and variables-

Symbols:

$S(var_k)$ = size of variable k

n_k = number of accesses to variable k

$\Delta e(var_k)$ = energy ***saved*** per variable access, if var_k is migrated

$\Delta E(var_k)$ = energy saved if variable var_k is migrated (= $\Delta e(var_k)\ n(var_k)$)

$x(var_k)$ = decision variable, =1 if variable k is migrated to SPM,
=0 otherwise

K = set of variables

Similar for functions I

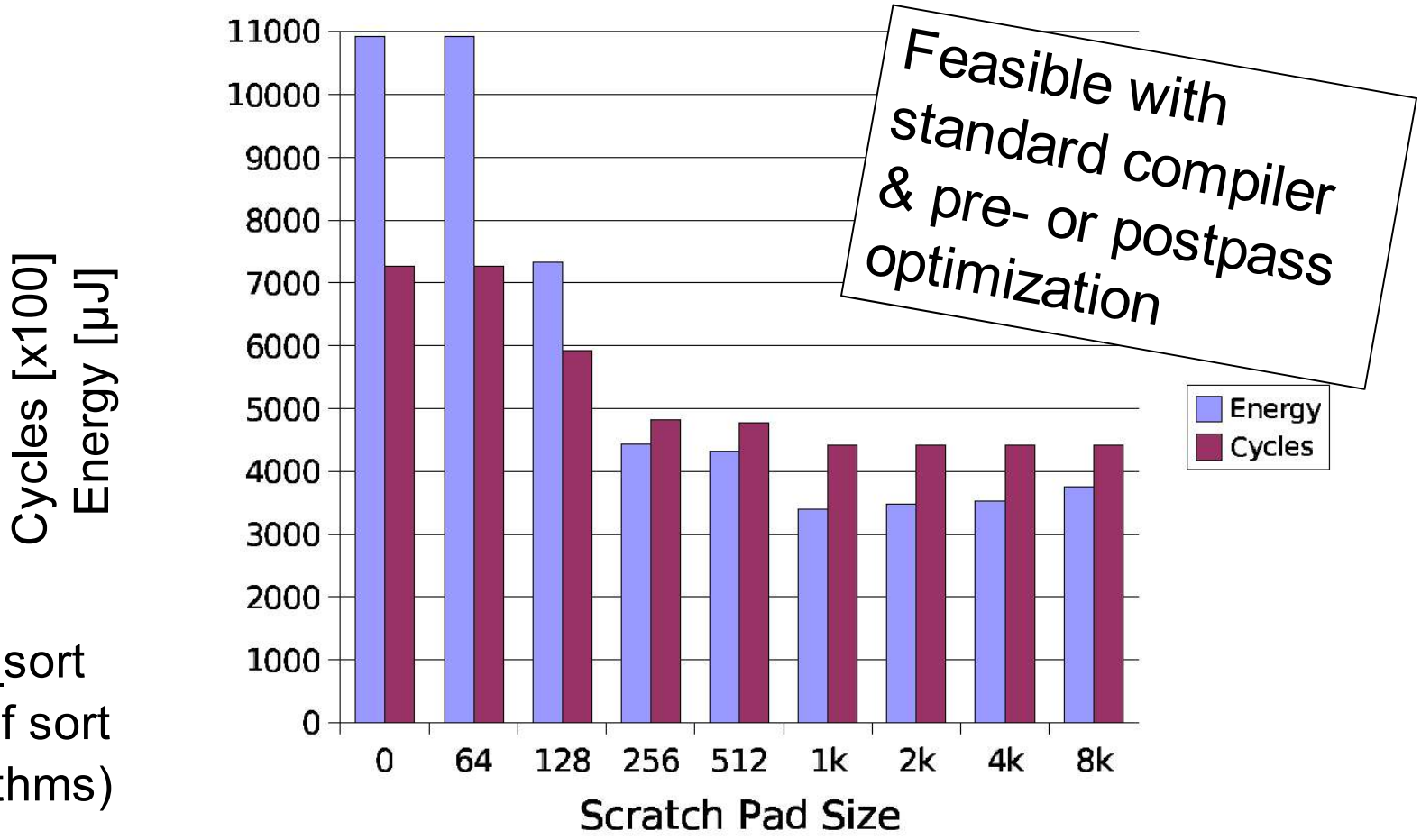
Integer programming formulation:

Maximize $\sum_{k \in K} x(var_k)\ \Delta E(var_k) + \sum_{i \in I} x(F_i)\ \Delta E(F_i)$

Subject to the constraint

$\sum_{i \in I} S(F_i)\ x(F_i) + \sum_{k \in K} S(var_k)\ x(var_k) \leq SSP$

Reduction in energy and average run-time



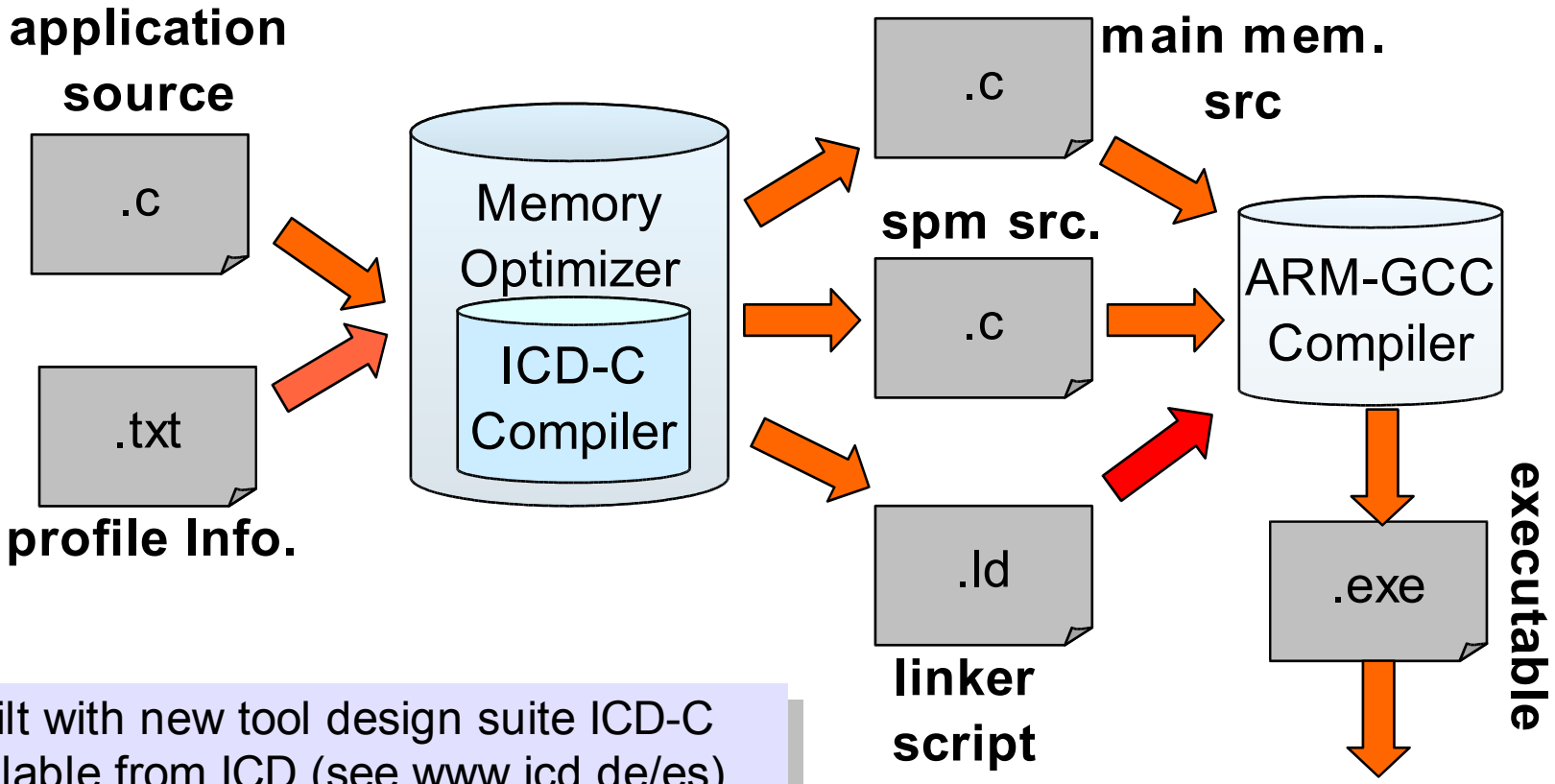
Multi_sort (mix of sort algorithms)

Measured processor / external memory energy + CACTI values for SPM (combined model)

Numbers will change with technology, algorithms remain unchanged.

Using these ideas with an gcc-based tool flow

Source is split into 2 different files by specially developed memory optimizer tool *.

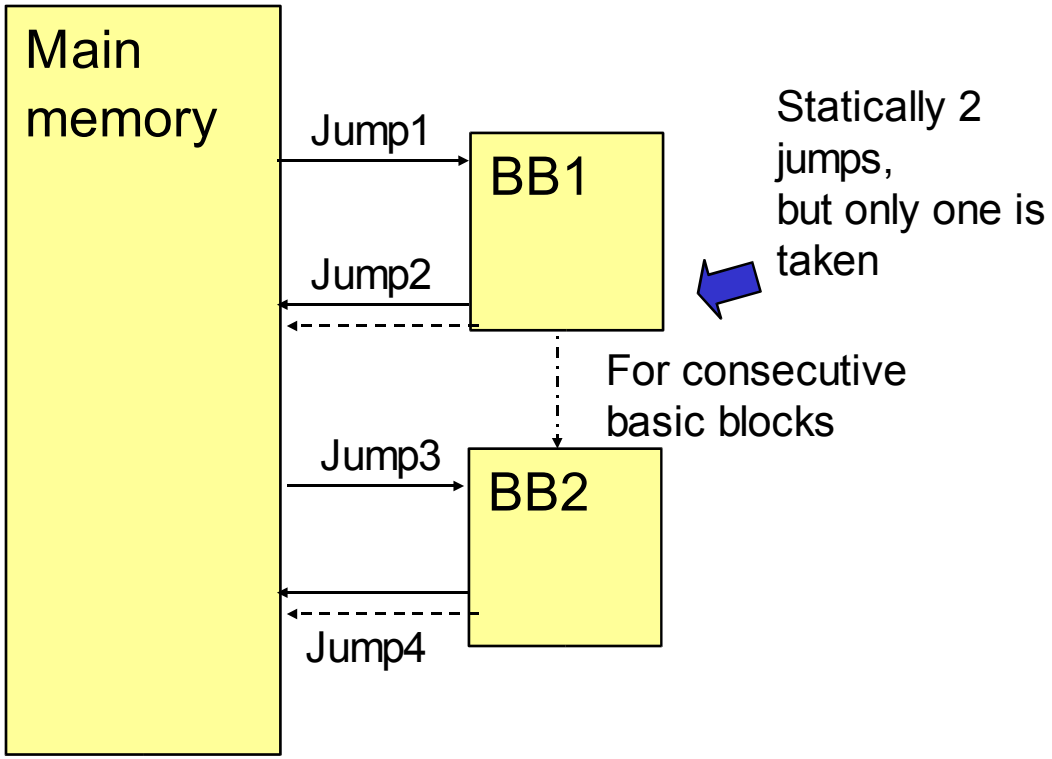


*Built with new tool design suite ICD-C available from ICD (see www.icd.de/es)

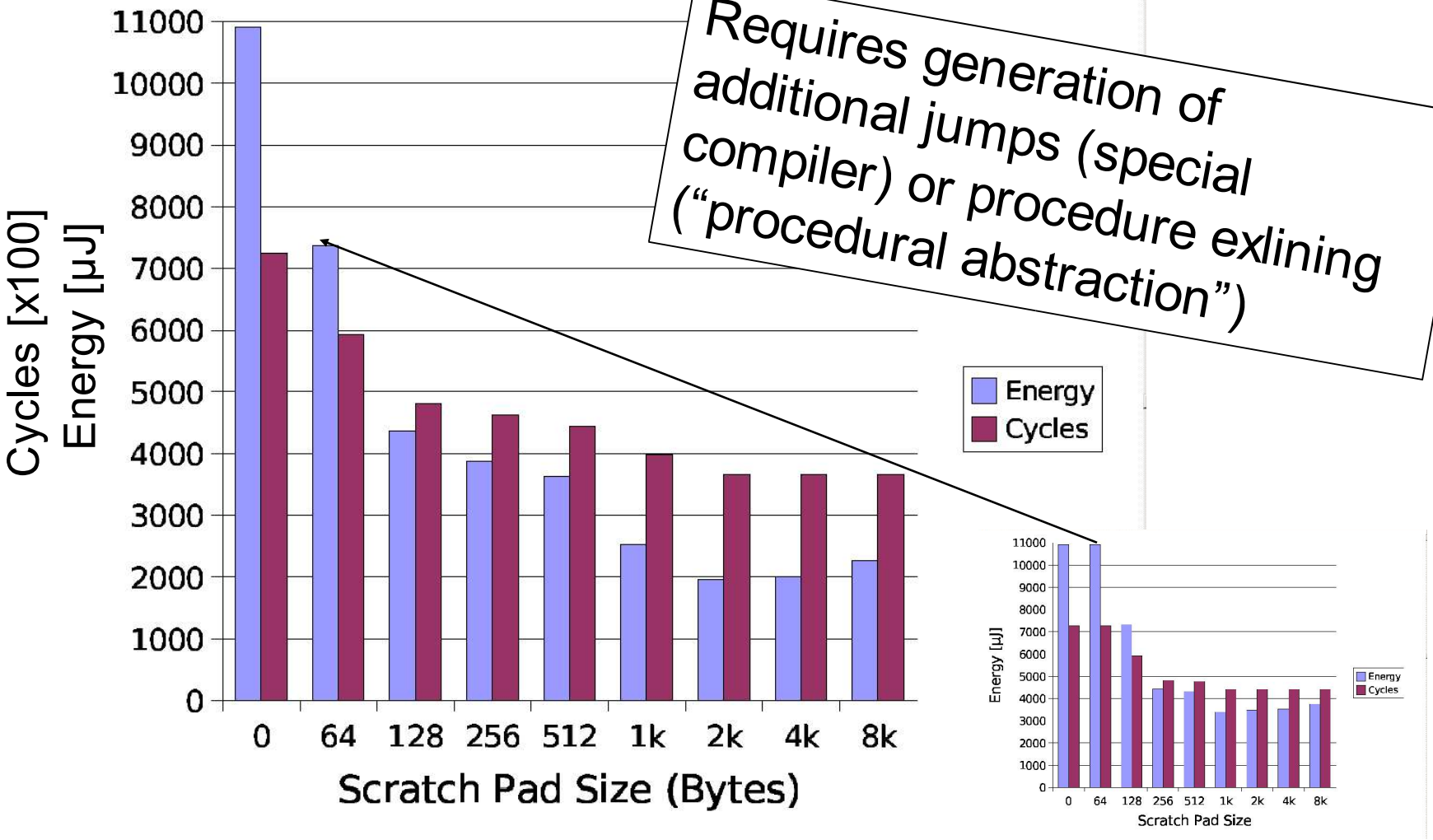
Allocation of basic blocks

Fine-grained granularity smoothens dependency on the size of the scratch pad.

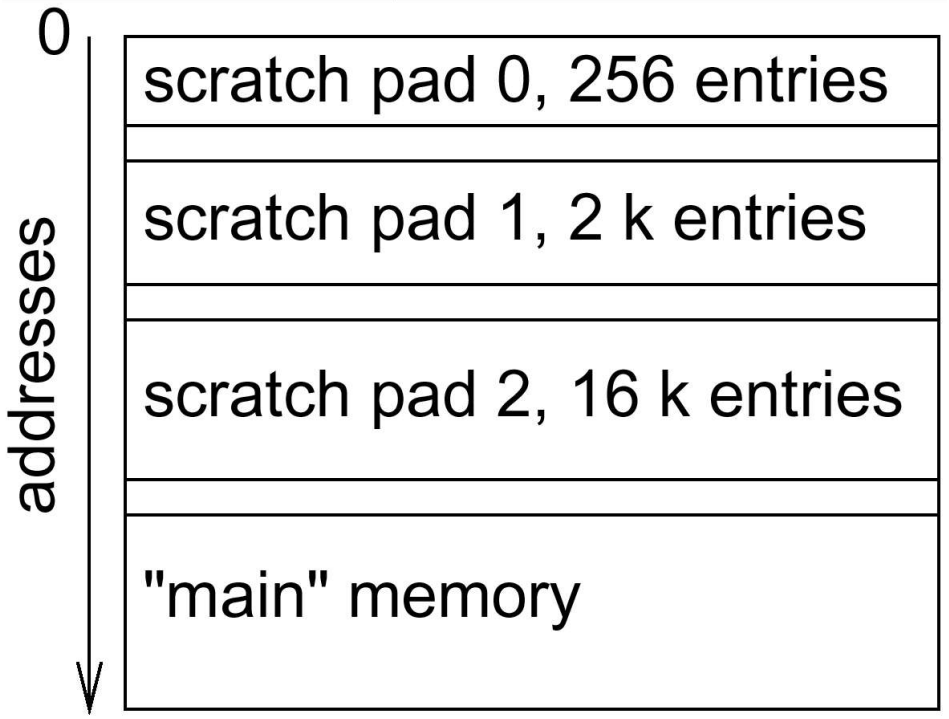
Requires additional jump instructions to return to "main" memory.



Allocation of basic blocks, sets of adjacent basic blocks and the stack



Multiple scratch pads



Considered partitions

# of partitions	number of partitions of size:						
	4K	2K	1K	512	256	128	64
7	0	1	1	1	1	1	2
6	0	1	1	1	1	2	0
5	0	1	1	1	2	0	0
4	0	1	1	2	0	0	0
3	0	1	2	0	0	0	0
2	0	2	0	0	0	0	0
1	1	0	0	0	0	0	0

Table 1: Example of all considered memory partitions for a total capacity of 4096 bytes

Optimization for multiple scratch pads

Minimize $$C = \sum_j e_j \cdot \sum_i x_{j,i} \cdot n_i$$

With e_j: energy ***required*** per access to memory j,

and $x_{j,i}$= 1 if object i is mapped to memory j, =0 otherwise,

and n_i: number of accesses to memory object i,

subject to the constraints:

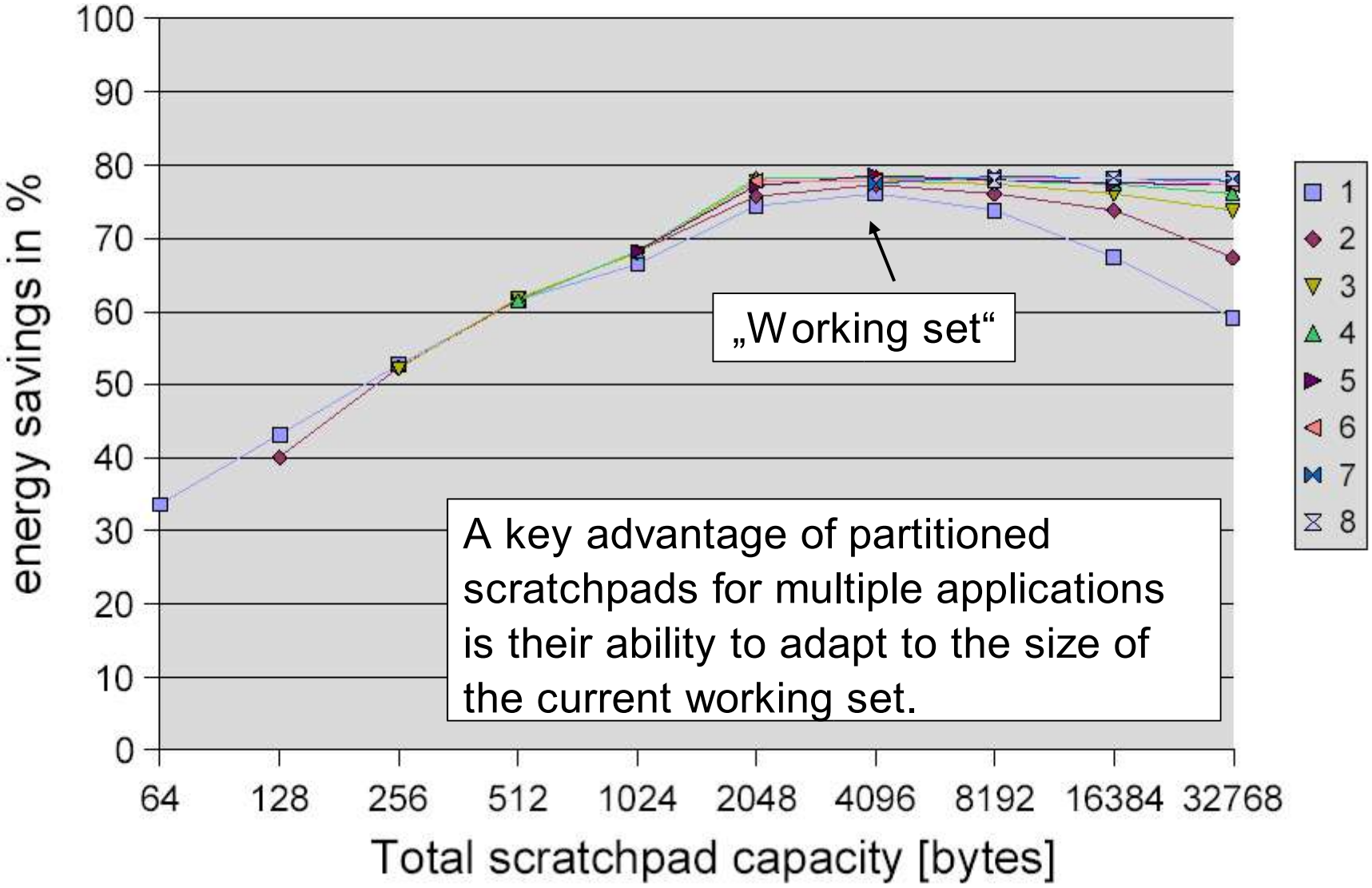
$$\forall j : \sum_i x_{j,i} \cdot S_i \leq SSP_j$$

$$\forall i : \sum_j x_{j,i} = 1$$

With S_i: size of memory object i,

SSP_j: size of memory j.

Results for parts of GSM coder/decoder

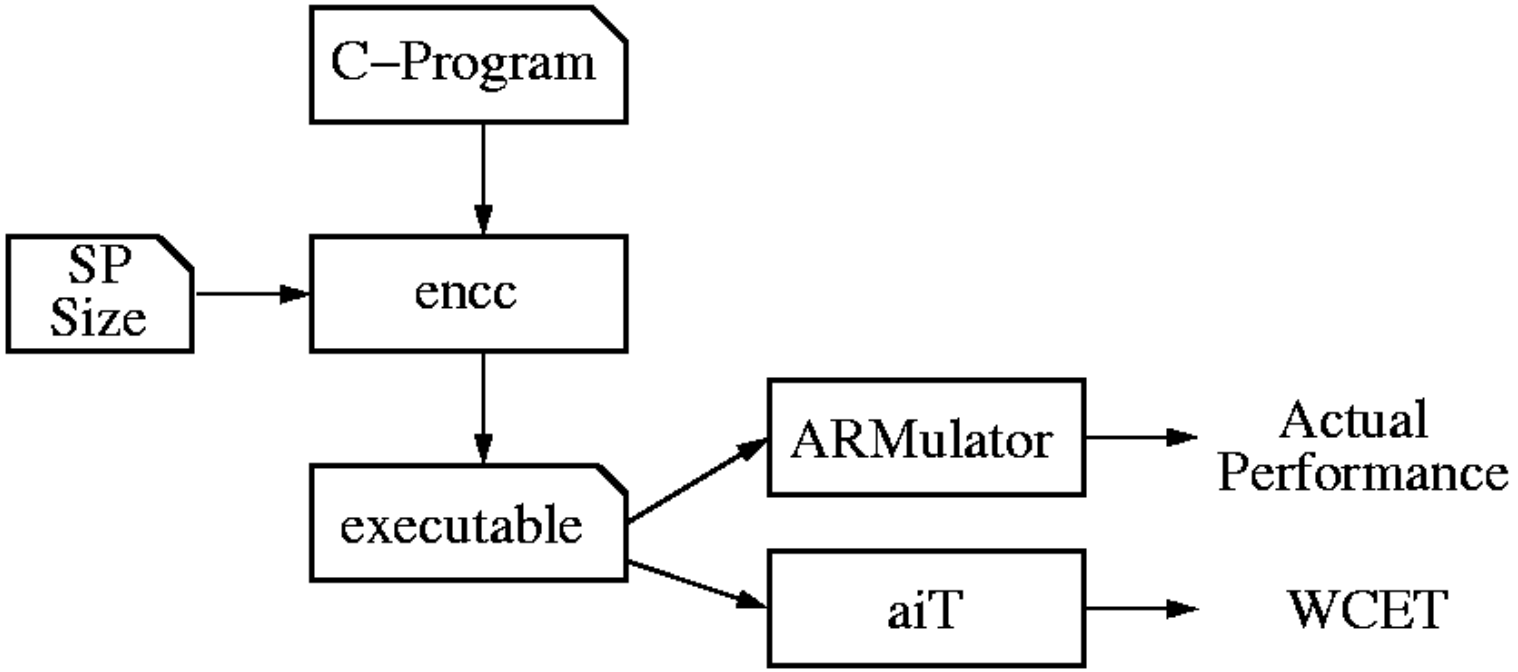


Objective function: worst case execution time *bound* (WCET)

Pre run-time scheduling *is often the only practical means of providing predictability in a complex system.* [Xu, Parnas]

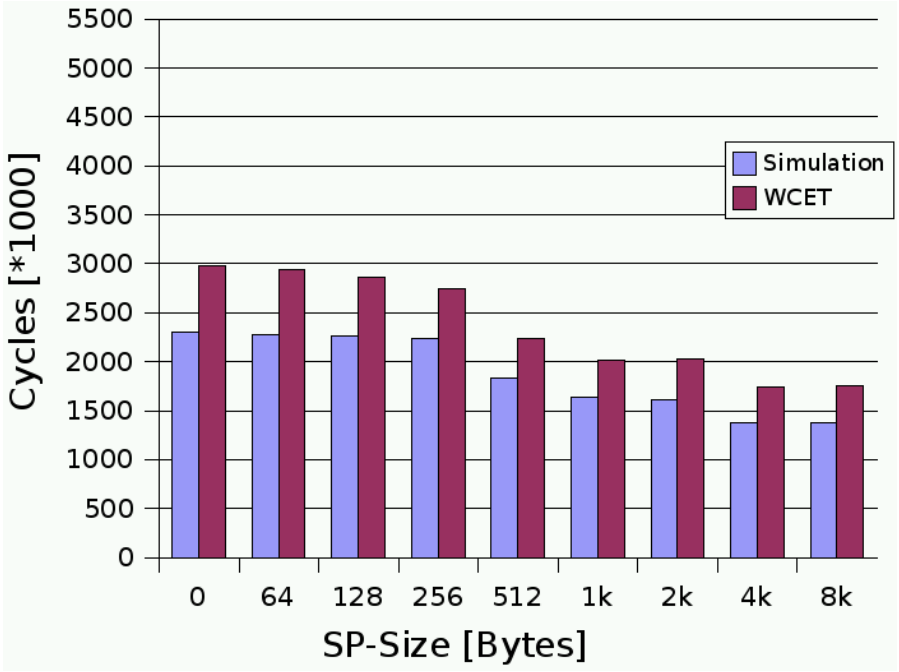
- ☞Time-triggered, statically scheduled operating systems
- ☞ Let‘s do the same for the memory system
  - ☞Scratch-pad/tightly coupled memory based predictability

Worst case timing analysis using aiT

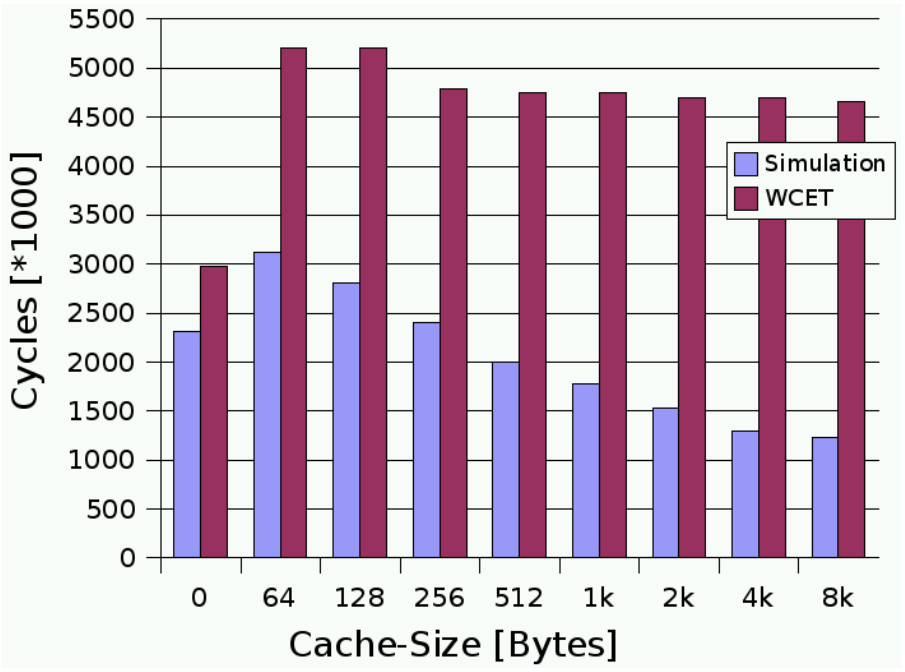


Results for G.721

Using Scratchpad:

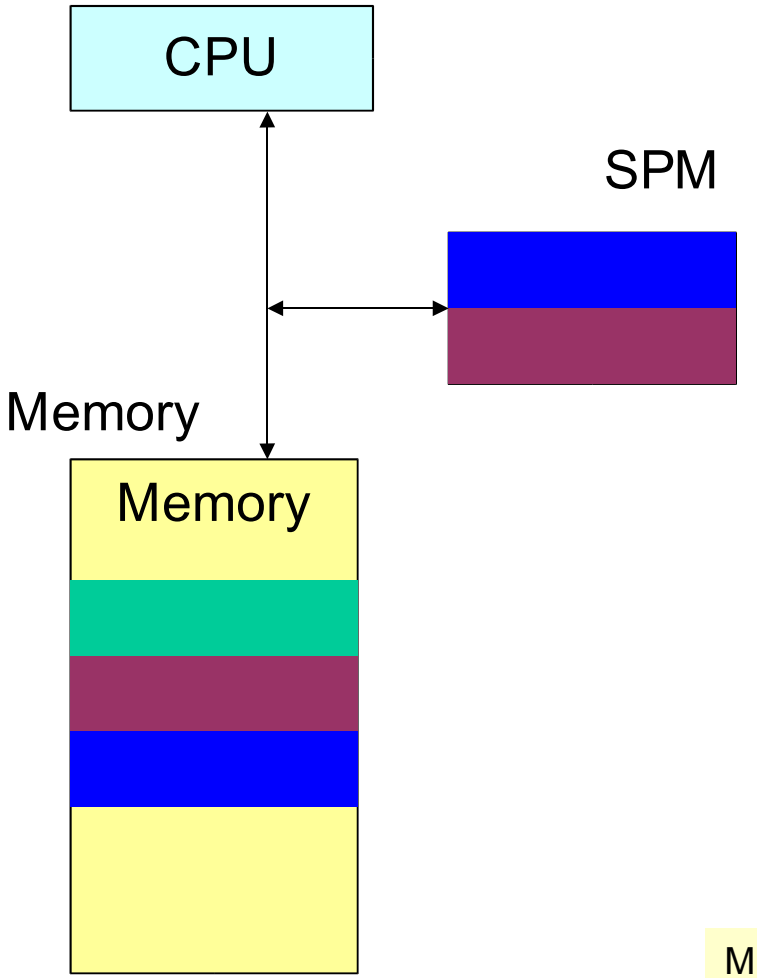


Using Unified Cache:



- L. Wehmeyer, P. Marwedel: Influence of Onchip Scratchpad Memories on WCET: *4th Intl Workshop on worst-case execution time analysis, (WCET)*, 2004
- L. Wehmeyer, P. Marwedel: Influence of Memory Hierarchies on Predictability for Time Constrained Embedded Software*, Design Automation and Test in Europe (DATE),* 2005

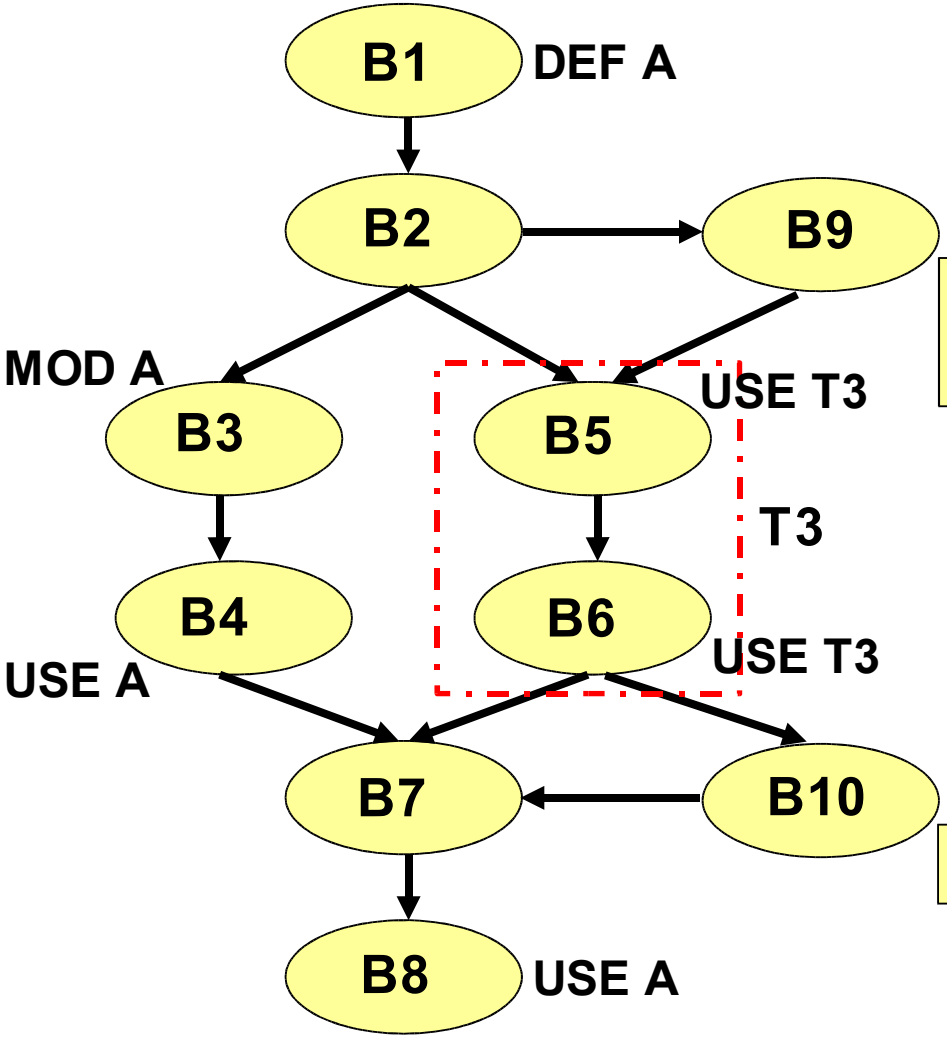
Dynamic replacement within scratch pad



- Effectively results in a kind of **compiler-controlled segmentation/ paging** for SPM
- Address assignment within SPM required (paging or segmentation-like)

M.Verma, P.Marwedel (U. Dortmund): Dynamic Overlay of Scratchpad Memory for Energy Minimization, *ISSS,* 2004

Dynamic replacement of *data* within scratch pad: based on liveness analysis



MO = {A, T1, T2, T3, T4}

SP Size = |A| = |T1| …= |T4|

```
SPILL_STORE(A);
SPILL_LOAD(T3);
```

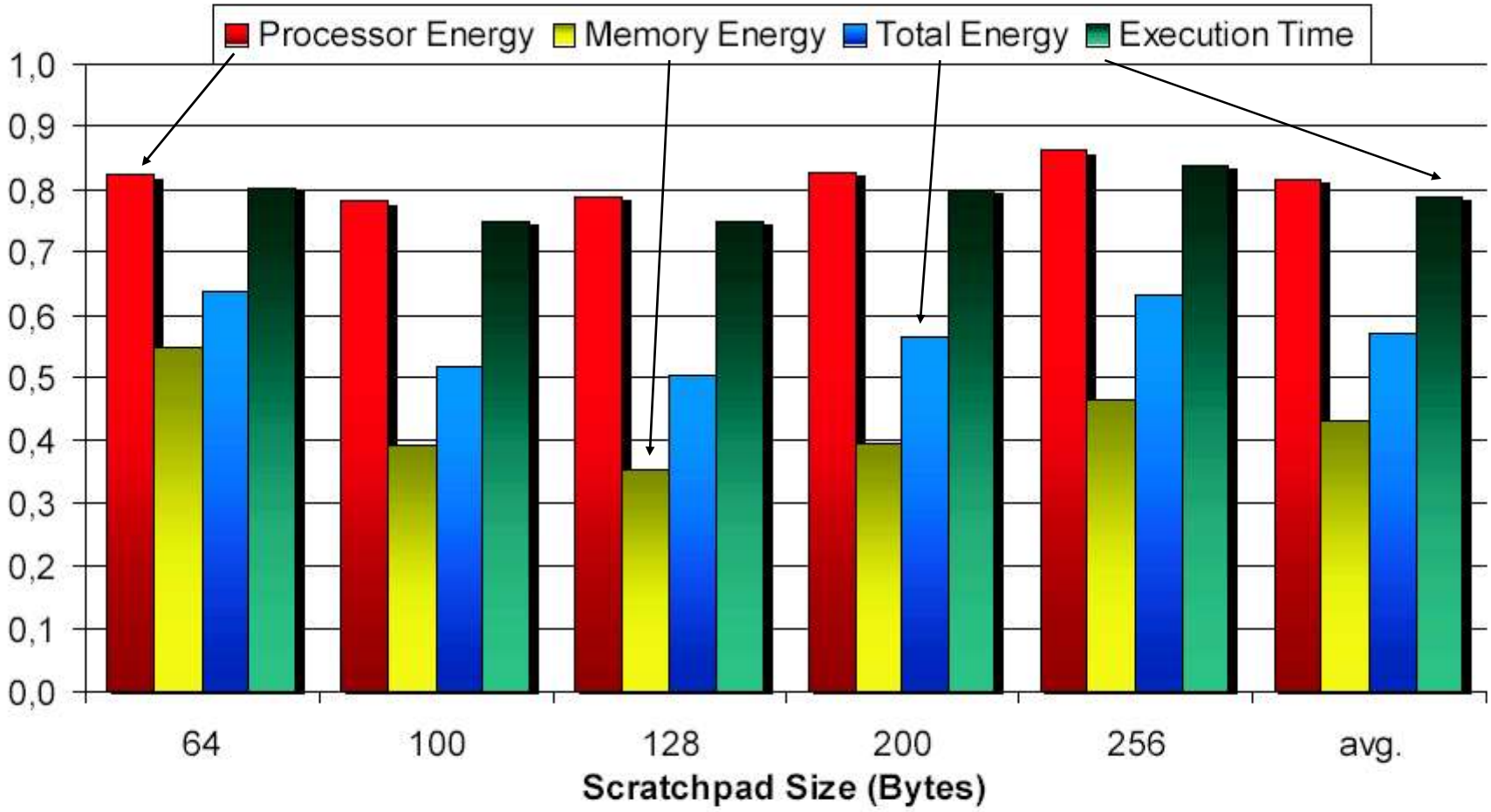
Solution:

A ➔ SP & T3 ➔ SP

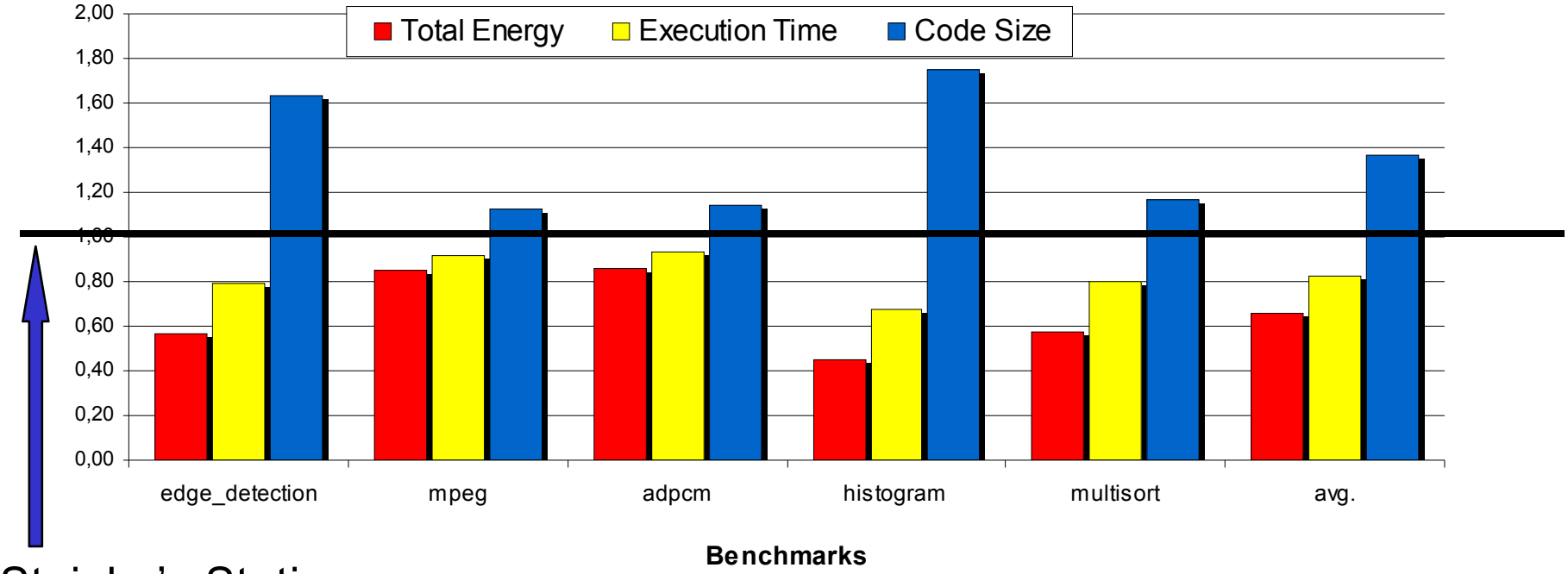
```
SPILL_LOAD(A);
```

Dynamic replacement within scratch pad

- Results for edge detection relative to static allocation -

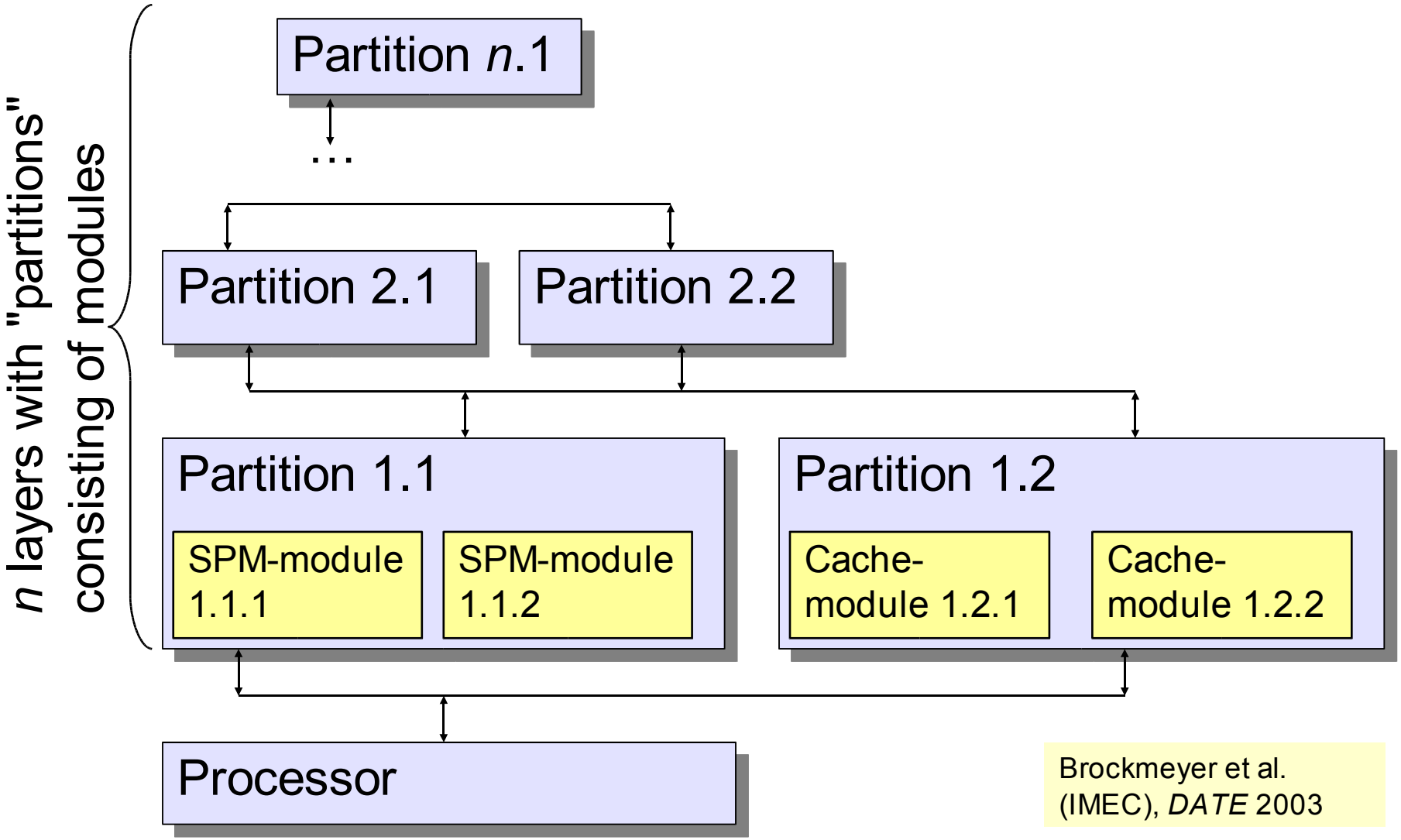


Dynamic replacement within scratch pad
- Energy, execution time, code size -



Steinke's Static Allocation

Hierarchical memories: Memory hierarchy layer assignment (MHLA) (IMEC)



Brockmeyer et al. (IMEC), *DATE* 2003

Memory hierarchy layer assignment (MHLA) - Copy candidates -

```
int A[250]
for (i=0; i<10; i++)
 for (j=0; j<10; j++)
  for (k=0; k<10; k++)
   for (l=0; l<10; l++)
    f(A[j*10+l])
size=0; reads(A)=10000
```

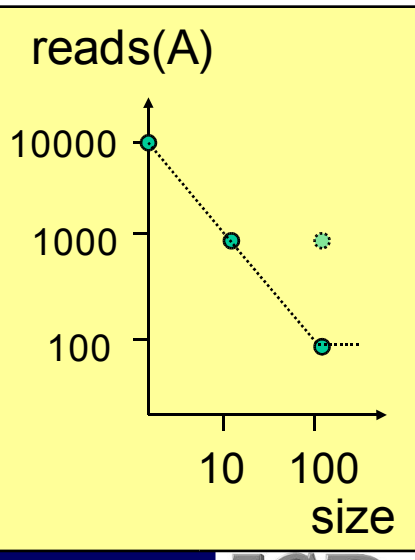
```
int A[250]
for (i=0; i<10; i++)
 for (j=0; j<10; j++)
 {A"[0..9]=A[j*10..j*10+9];
  for (k=0; k<10; k++)
   for (l=0; l<10; l++)
    f(A"[l])}
size=10; reads(A)=1000
```

Copy candidate

A', A" in small memory

```
int A[250]
for (i=0; i<10; i++)
 {A'[0..99]=A[0..99];
 for (j=0; j<10; j++)
  for (k=0; k<10; k++)
   for (l=0; l<10; l++)
    f(A'[j*10+l])}
size=100;reads(A)=1000
```

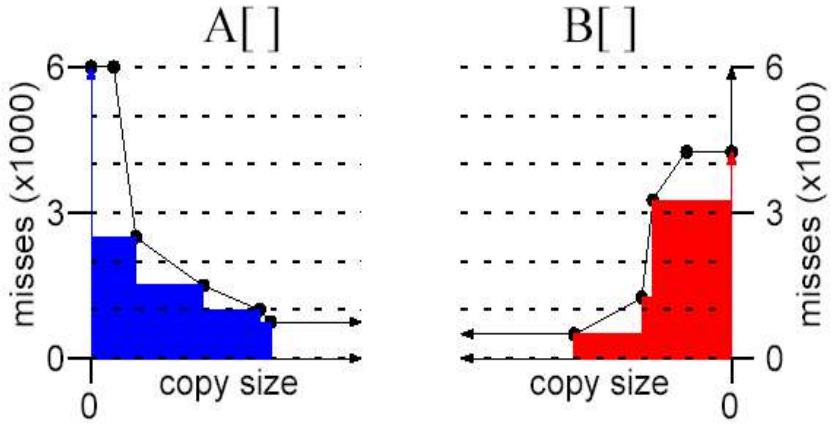
```
int A[250]
A'[0..99]=A[0..99];
for (i=0; i<10; i++)
 for (j=0; j<10; j++)
  for (k=0; k<10; k++)
   for (l=0; l<10; l++)
    f(A'[j*10+l])
size=100; reads(A)=100
```

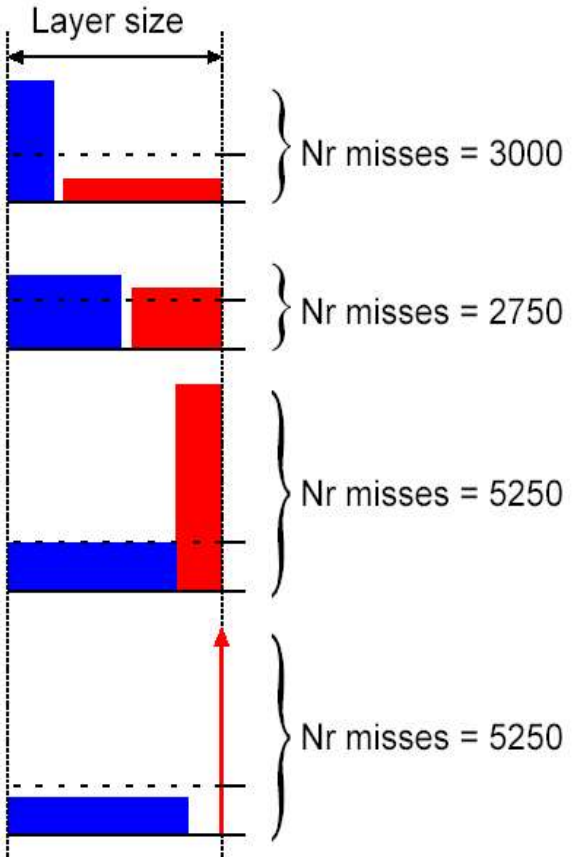


Memory hierarchy layer assignment (MHLA) - Goal -

Goal: For each variable: find permanent layer, partition and module & select copy candidates such that energy is minimized.

Conflicts between variables





Brockmeyer et al., *DATE* 2003

Memory hierarchy layer assignment (MHLA) - Approach -

Approach:

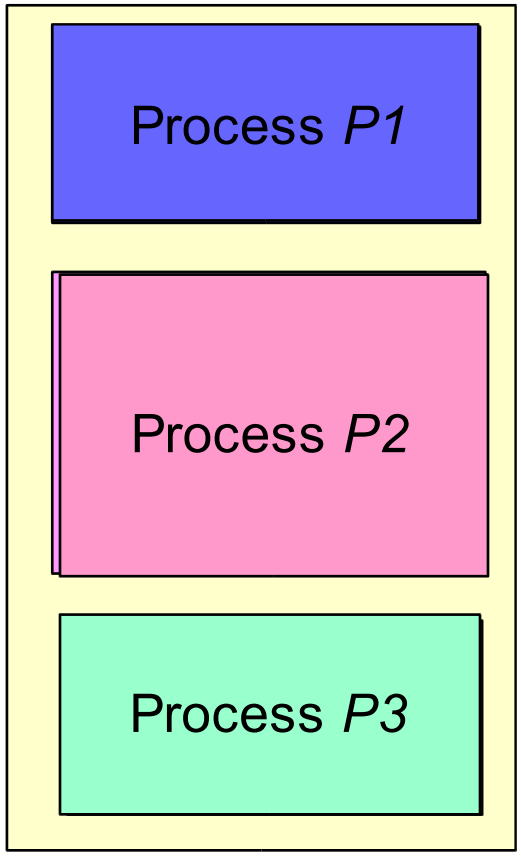
start with initial variable allocation

incrementally improve initial solution

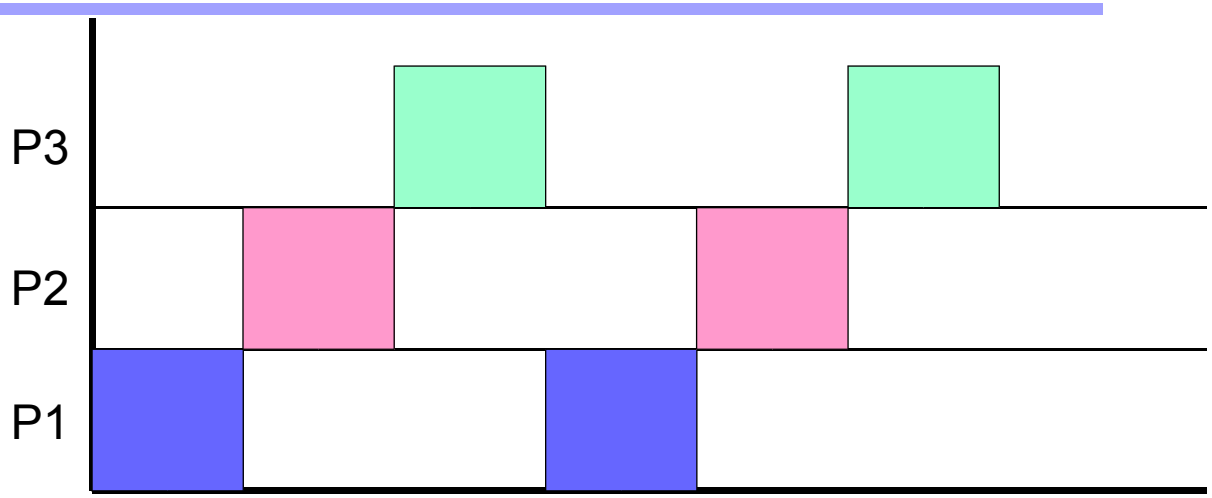
such that total energy is minimized.

More general hardware architecture than the Dortmund approach, but no global optimization.

Non-Saving Context Switch



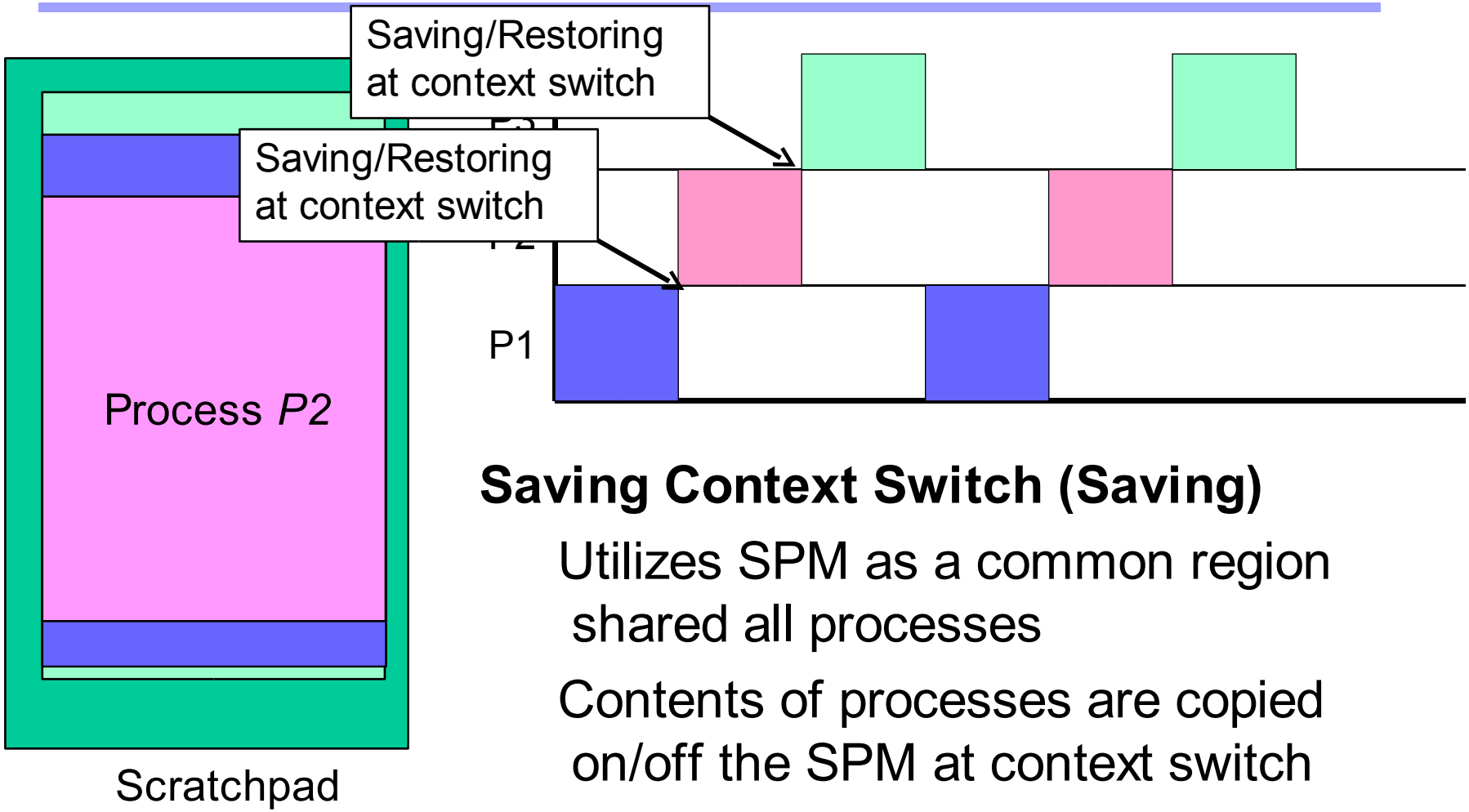
Scratchpad



Non-Saving Context Switch (Non-Saving)

- Partitions SPM into disjoint regions
- Each process is assigned a SPM region
- Copies contents during initialization
- Good for large scratchpads

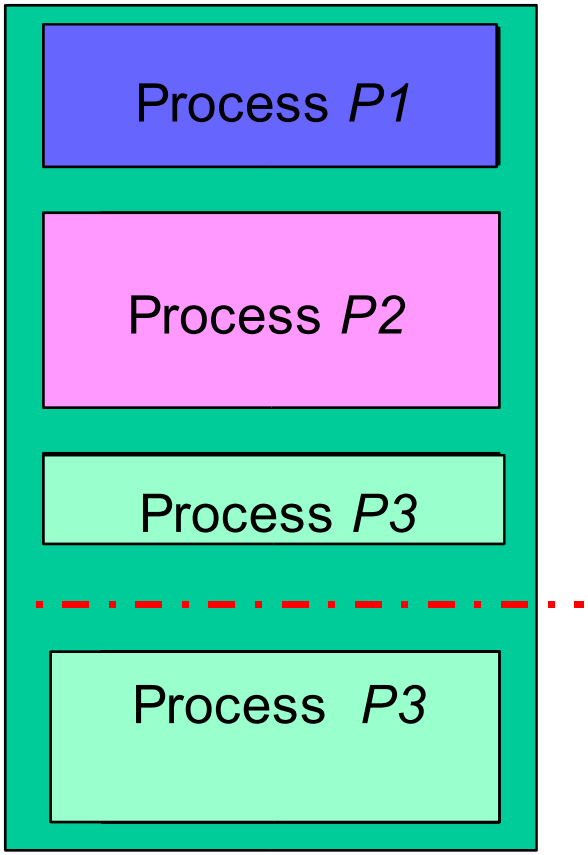
Saving/Restoring Context Switch

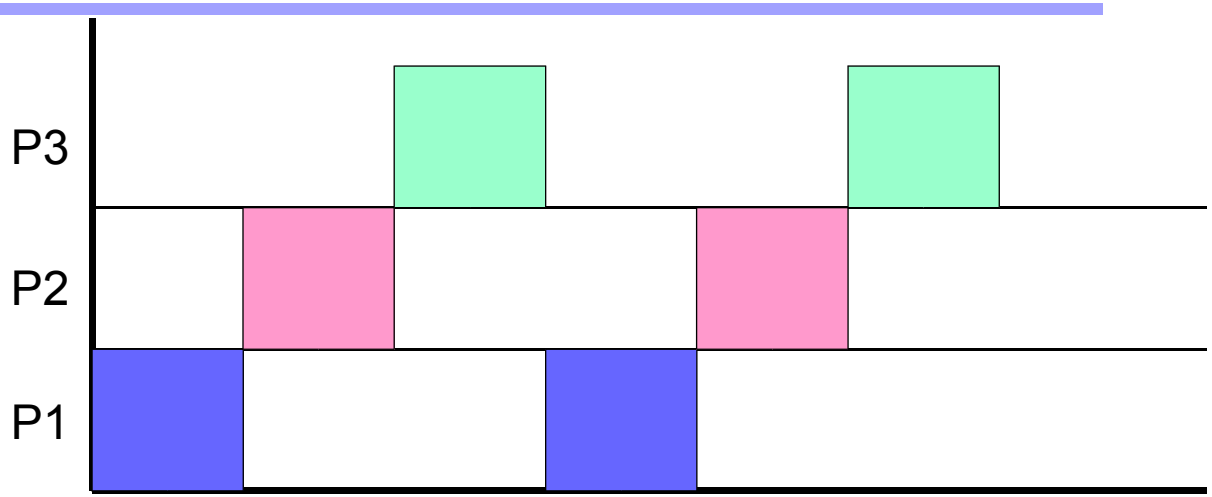


Saving Context Switch (Saving)

- Utilizes SPM as a common region shared all processes
- Contents of processes are copied on/off the SPM at context switch
- Good for small scratchpads

Hybrid Context Switch

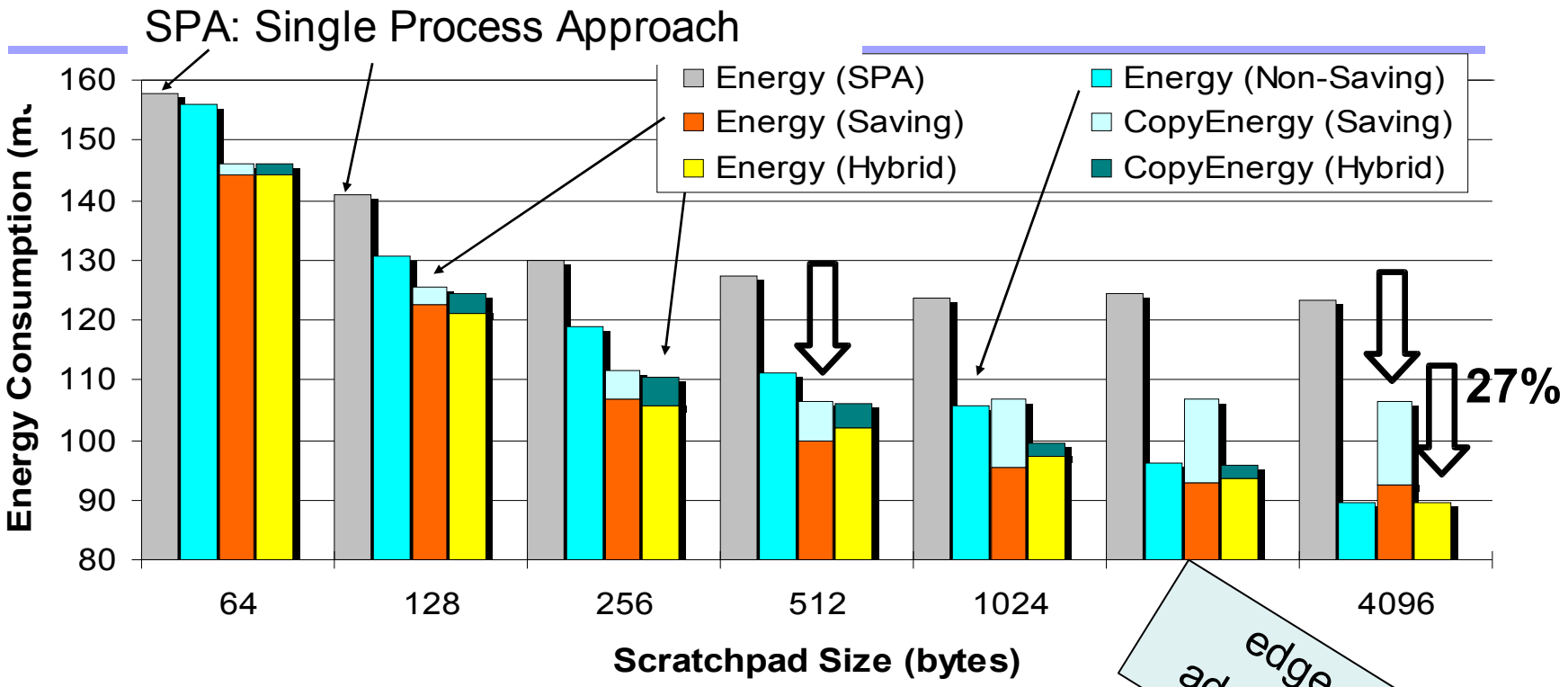




Hybrid Context Switch (Hybrid)

- Disjoint + Shared SPM regions
- Good for all scratchpads
- Analysis is similar to Non-Saving Approach
- Runtime: $O(nM^3)$

Multi-process Scratchpad Allocation: Results



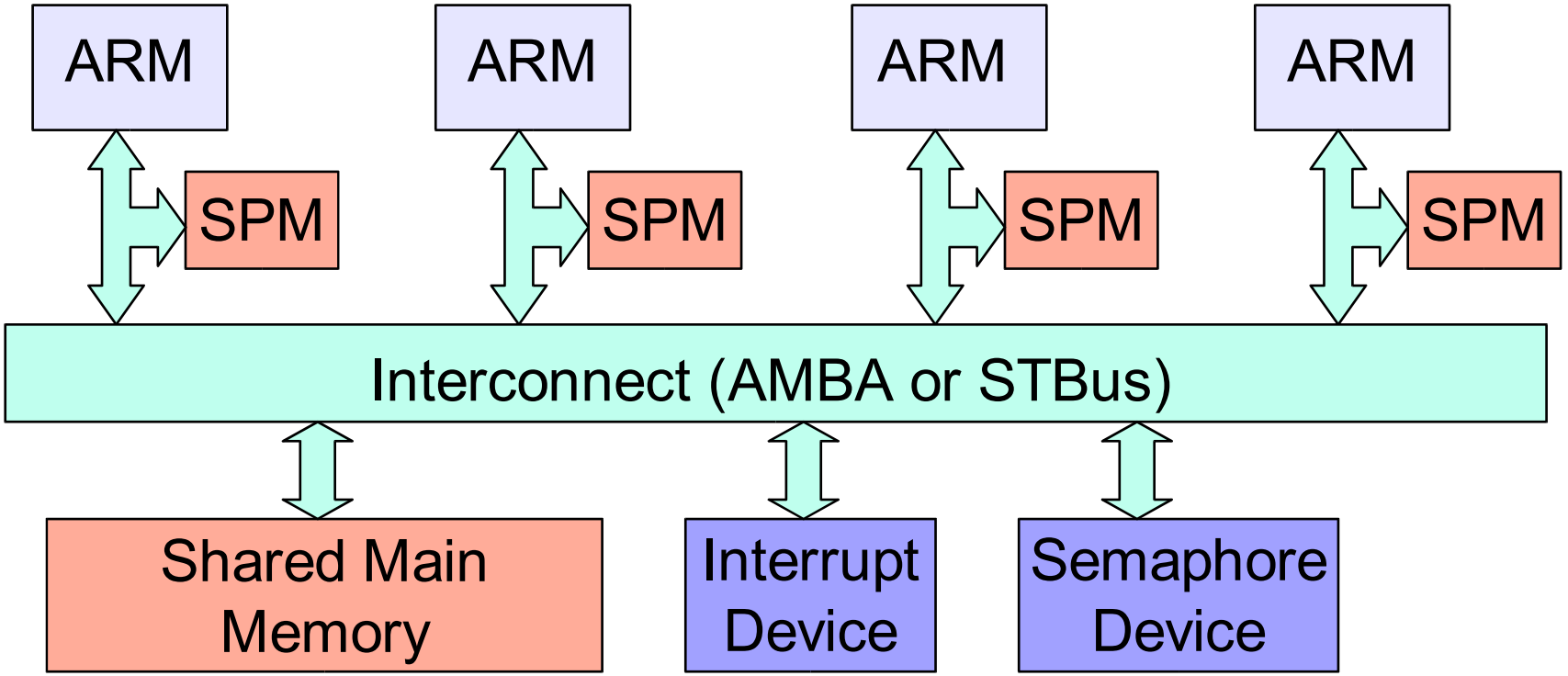
For small SPMs (64B-512B) Saving is better

For large SPMs (1kB- 4kB) Non-Saving is better

Hybrid is the best for all SPM sizes.

Energy reduction @ 4kB SPM is 27% for Hybrid approach

Multi-processor ARM (MPARM) Framework



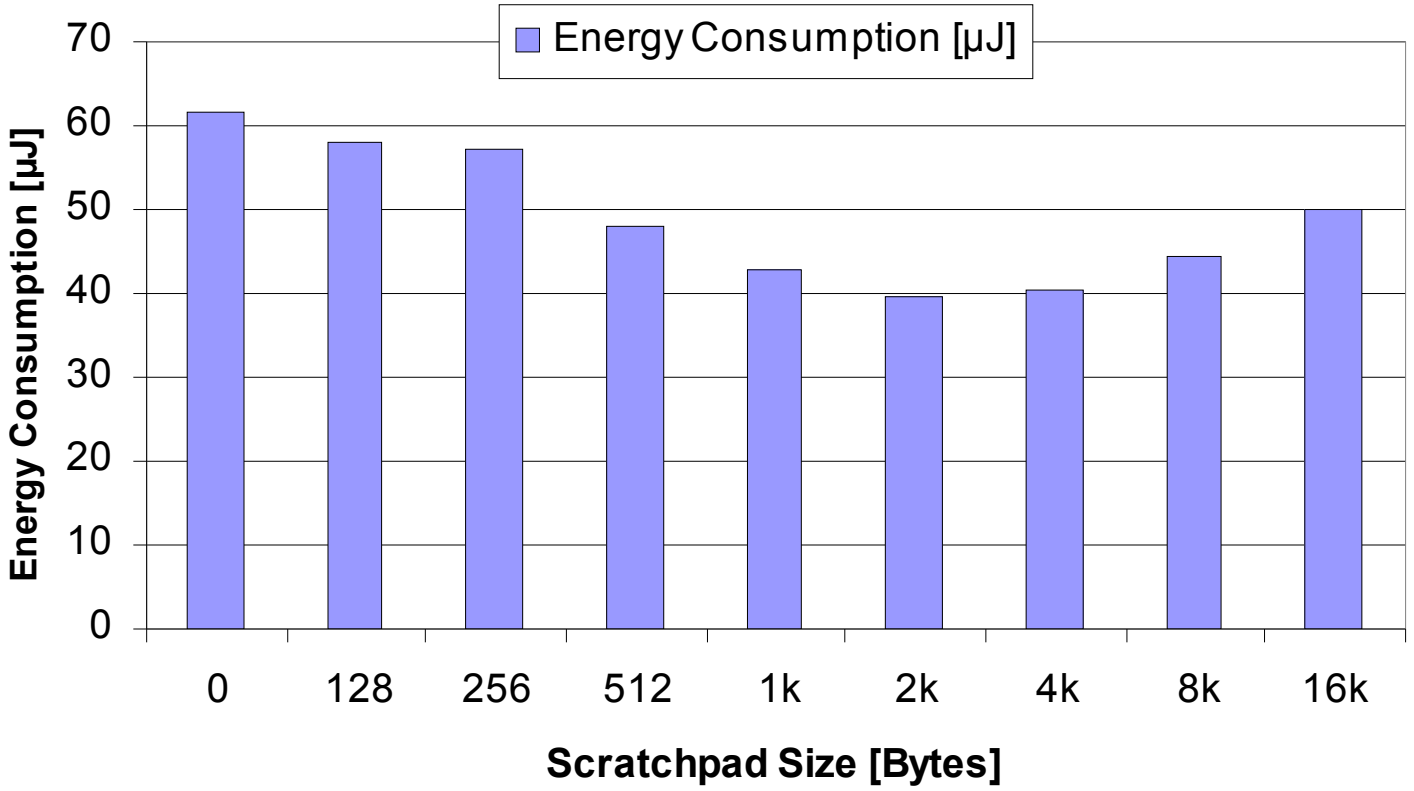
Homogenous SMP ~ CELL processor

Processing Unit : ARM7T processor

Shared Coherent Main Memory

Private Memory: Scratchpad Memory

Results (MOMPARM)



DES-Encryption: 4 processors: 2 Controllers+2 Compute Engines

Energy values from ST Microelectronics

Result of ongoing cooperation between U. Bologna and U. Dortmund supported by ARTIST2 network of excellence.

Summary

Careful design of the memory system is becoming more and more important also for embedded systems.

Multiple objectives: predictability, energy, latency, …

Scratch pad memories have several advantages.

Allocation algorithms considered:

- static allocation
  - partitioned scratch pad
  - analysis of worst case execution time
- dynamic allocation, hierarchy layer assignment
- multiple threads
- multiple processors