

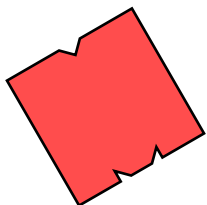
Cache Architectures for Real-Time Embedded Systems

Prof. Bruce Jacob

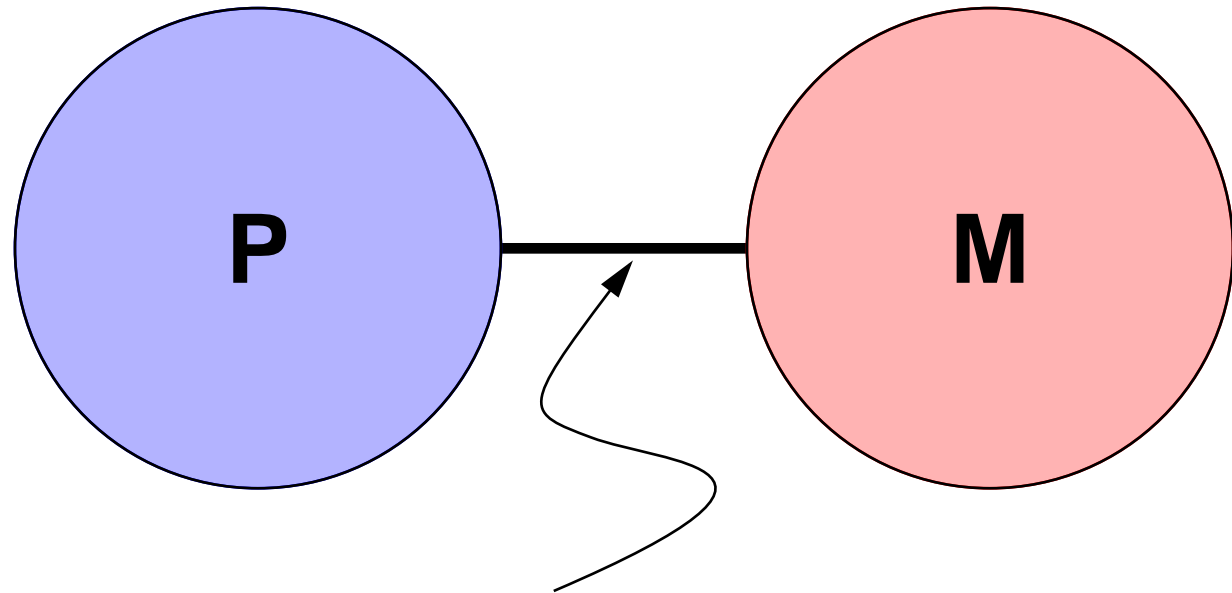
**Electrical & Computer Engineering
University of Maryland, College Park**

OUTLINE:

- **Cache Primer**
- **Memory Management Primer**
- **Caches & Embedded Systems**
- **Cache Architectures for Real-Time**

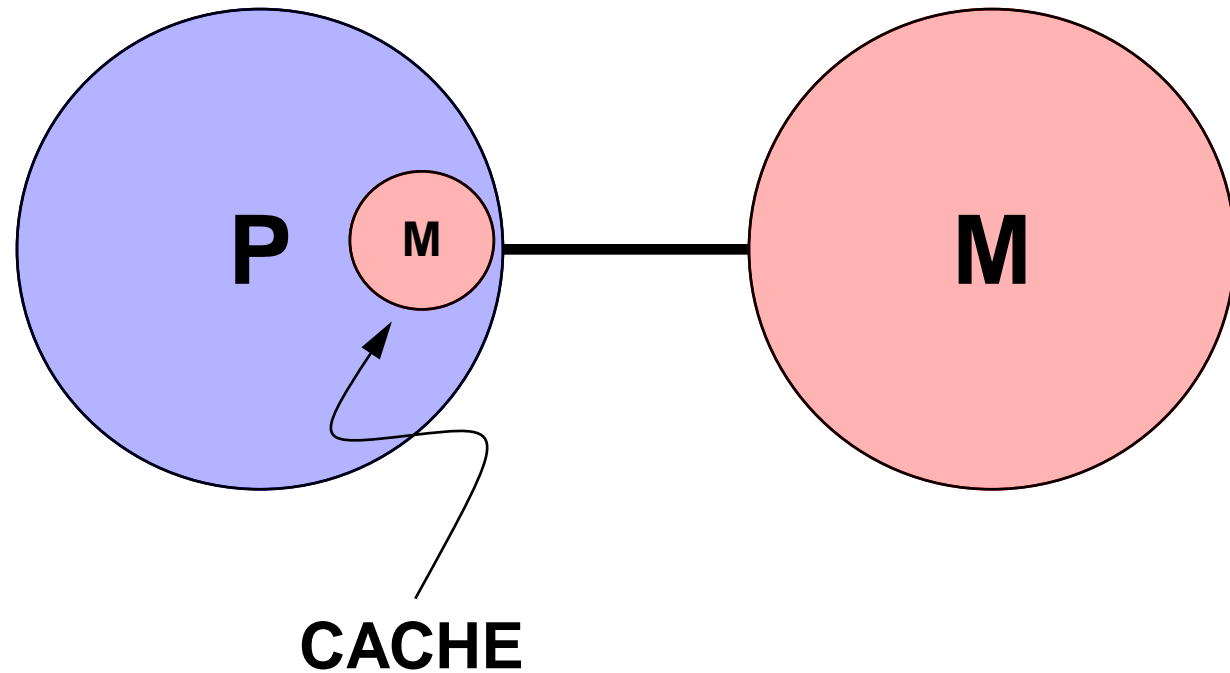


What is a Cache?



Von Neumann Bottleneck

What is a Cache?



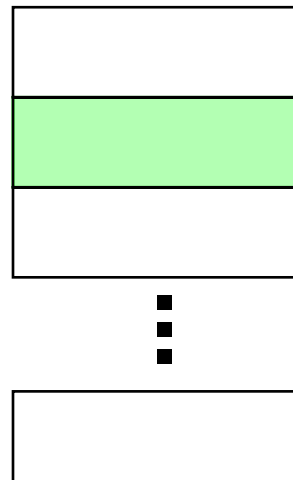
Cache Organizations



Fundamental Unit: CACHE BLOCK

Purpose: HOLD DATA

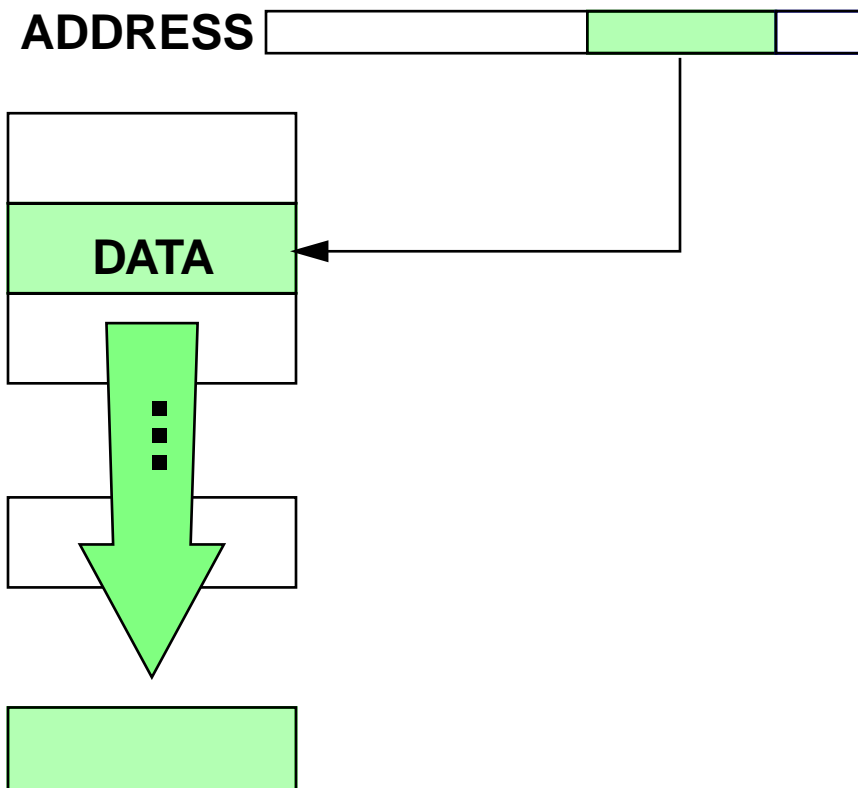
Cache Organizations



A Simple Cache

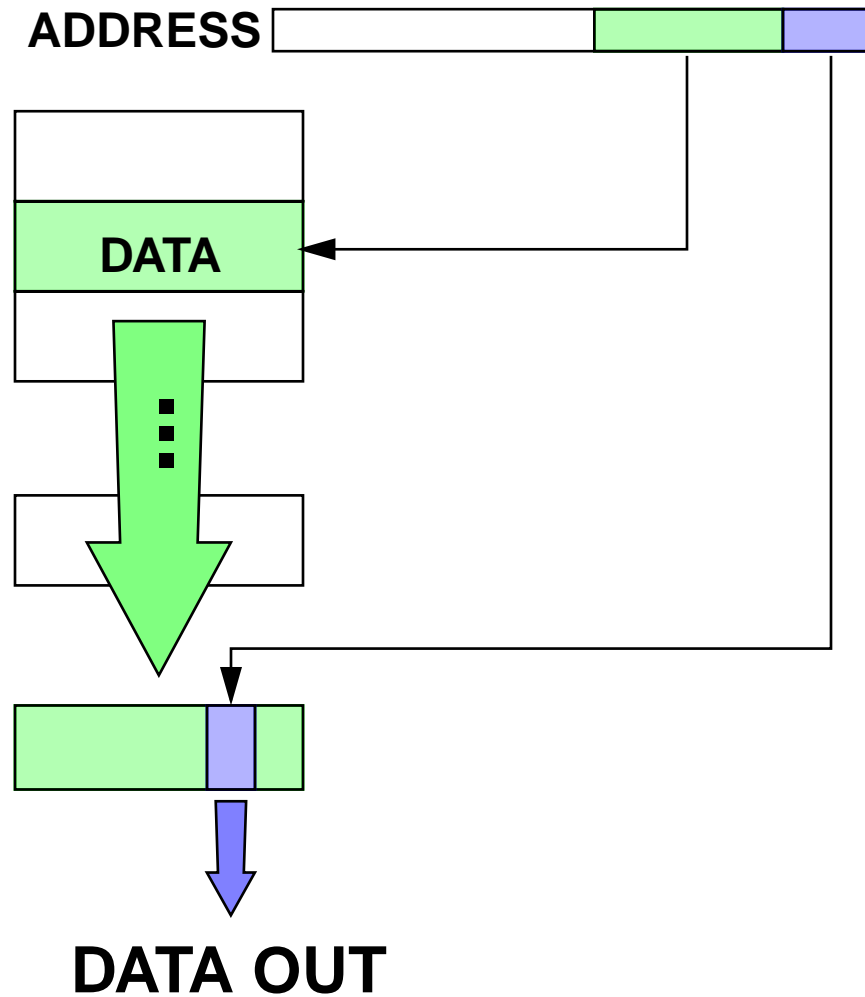
Cache Organizations

Cache Addressing Mechanism



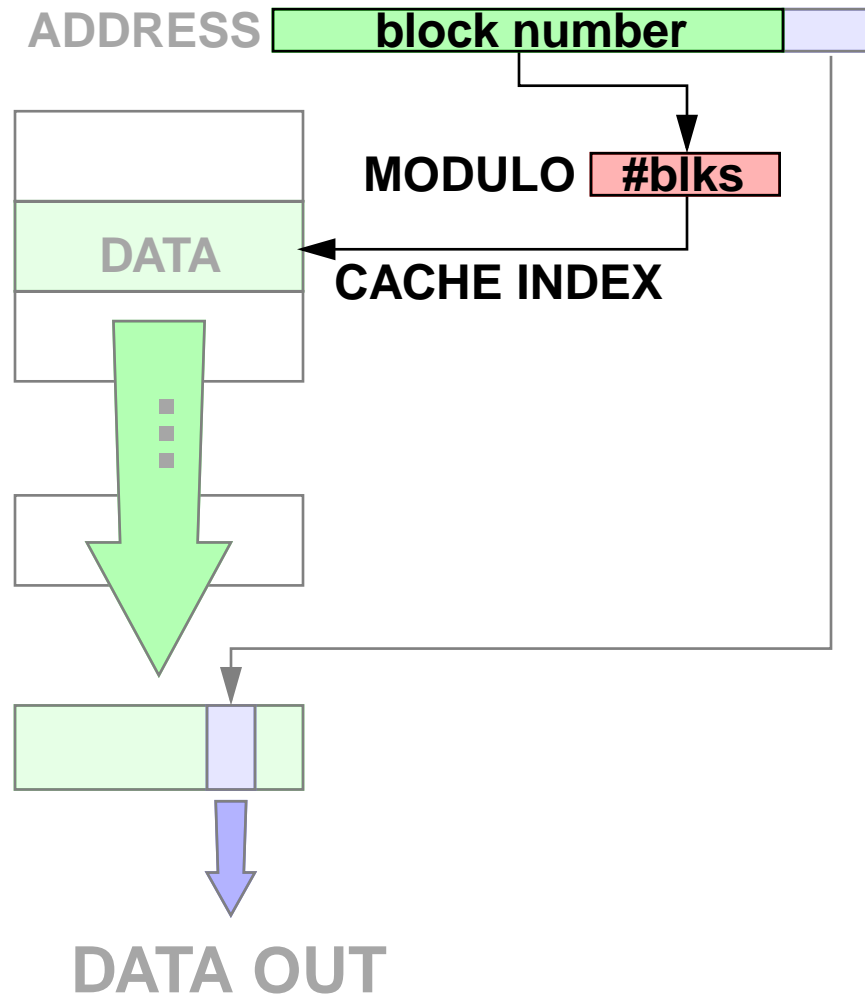
Cache Organizations

Cache Addressing Mechanism

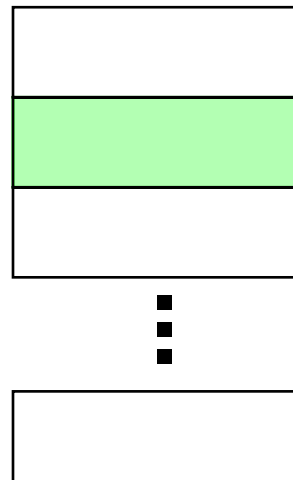


Cache Organizations

Cache Addressing Mechanism

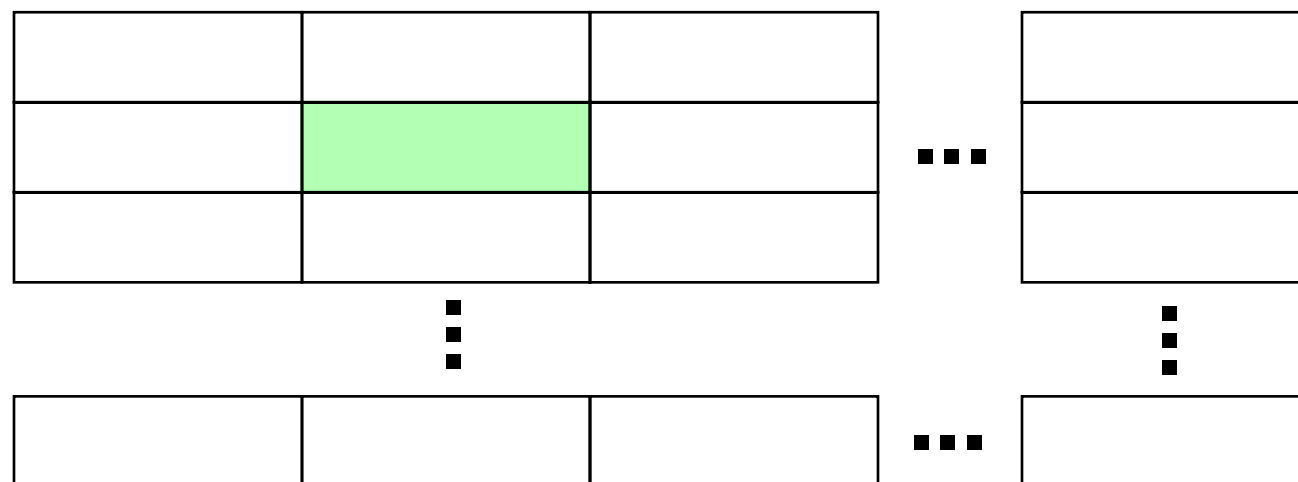


Cache Organizations



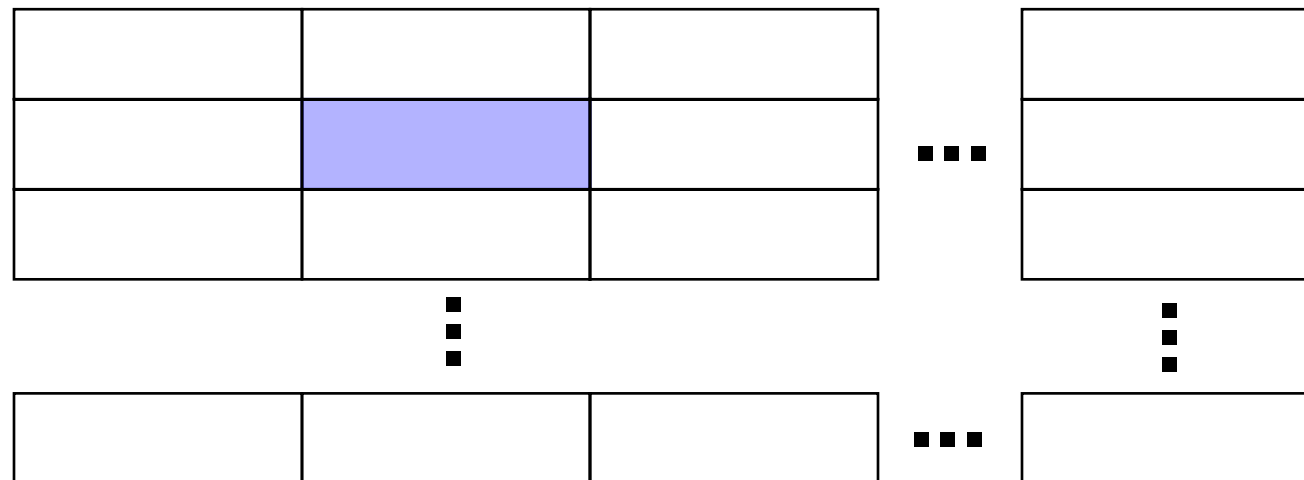
A Simple Cache

Cache Organizations



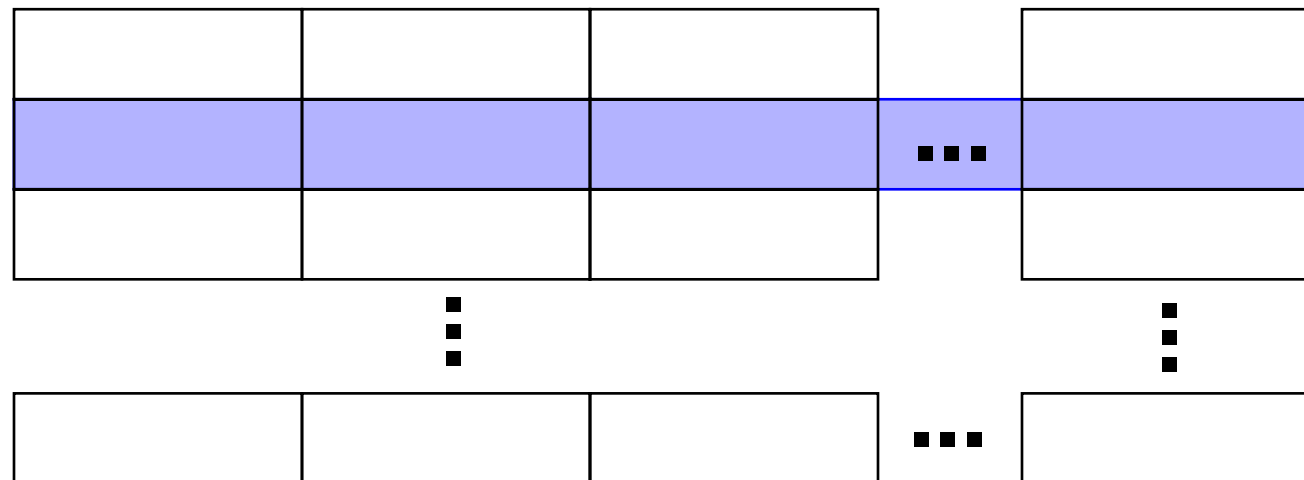
**A More Complex Cache
(similar to having several caches)**

Cache Organizations



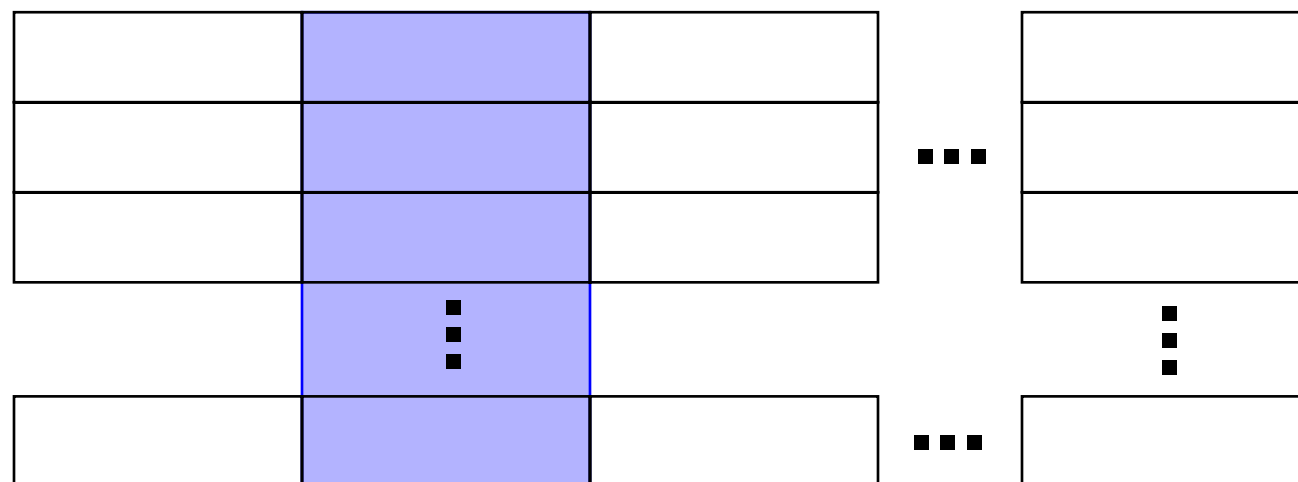
**A Single CACHE BLOCK (or LINE)
Parameter: BLOCK SIZE (or LINE SIZE)**

Cache Organizations



**A Single CACHE SET (equivalence class)
Parameter: ASSOCIATIVITY**

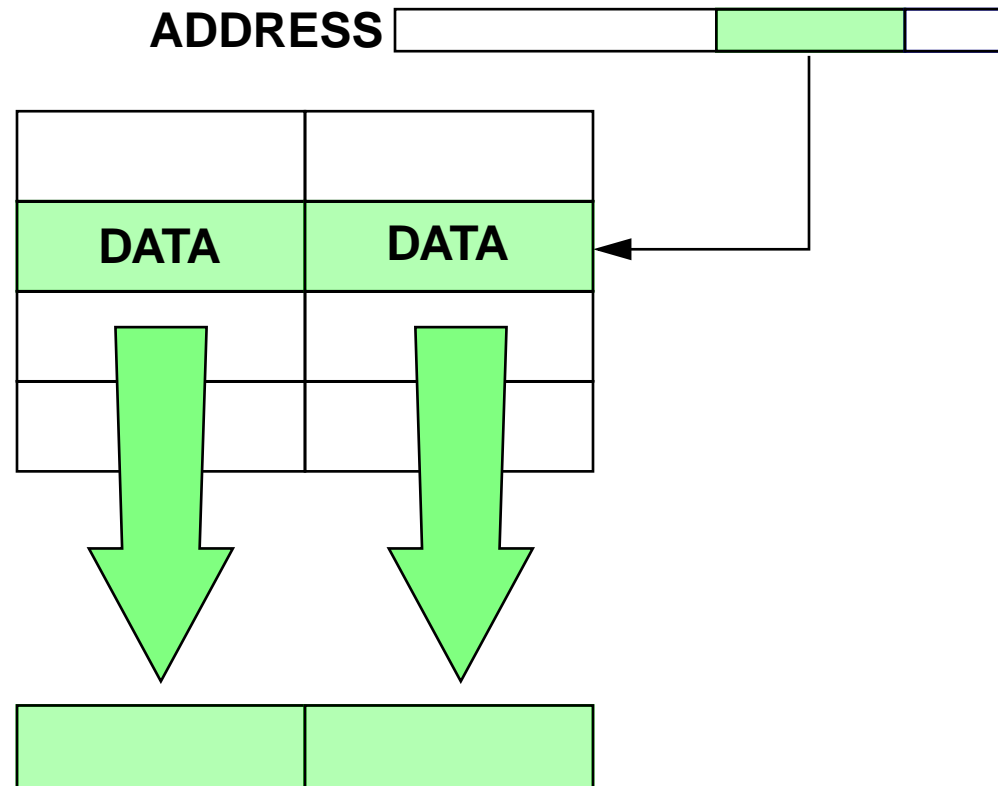
Cache Organizations



A Single CACHE COLUMN (or WAY)

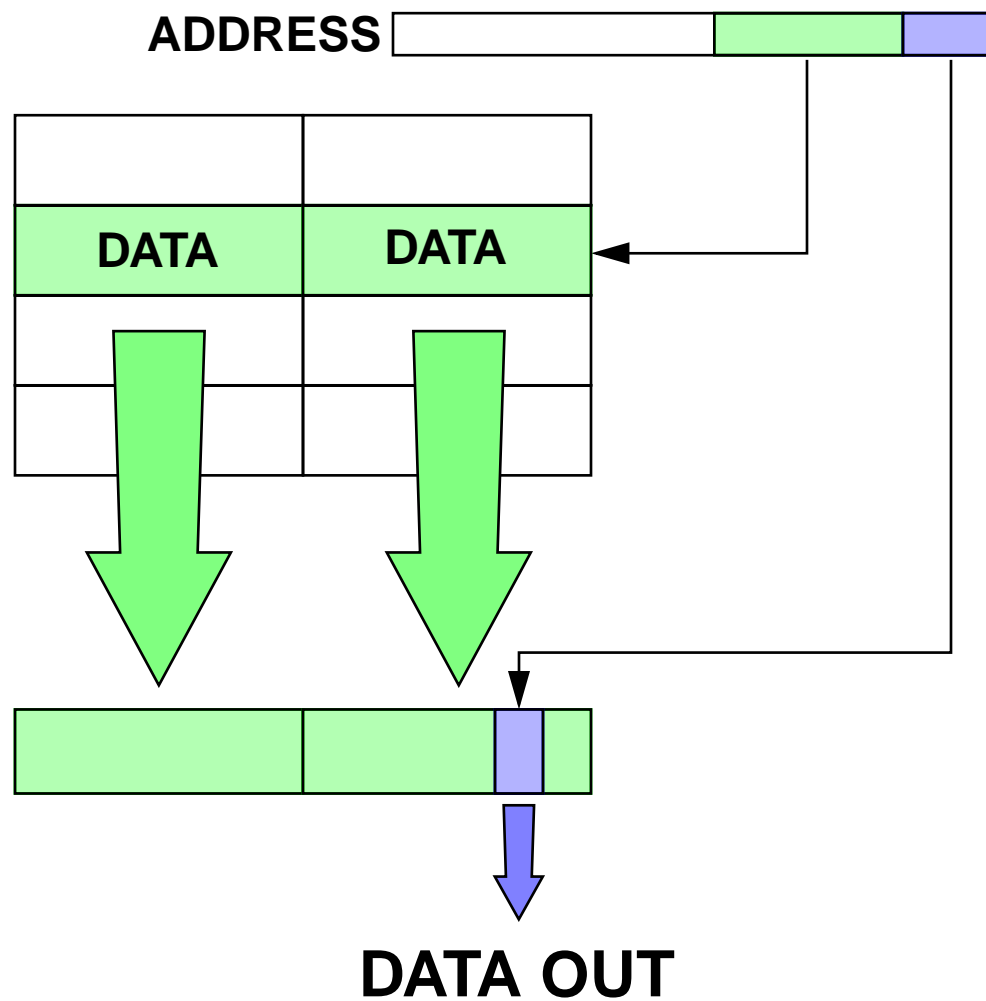
Cache Organizations

Associative Lookup



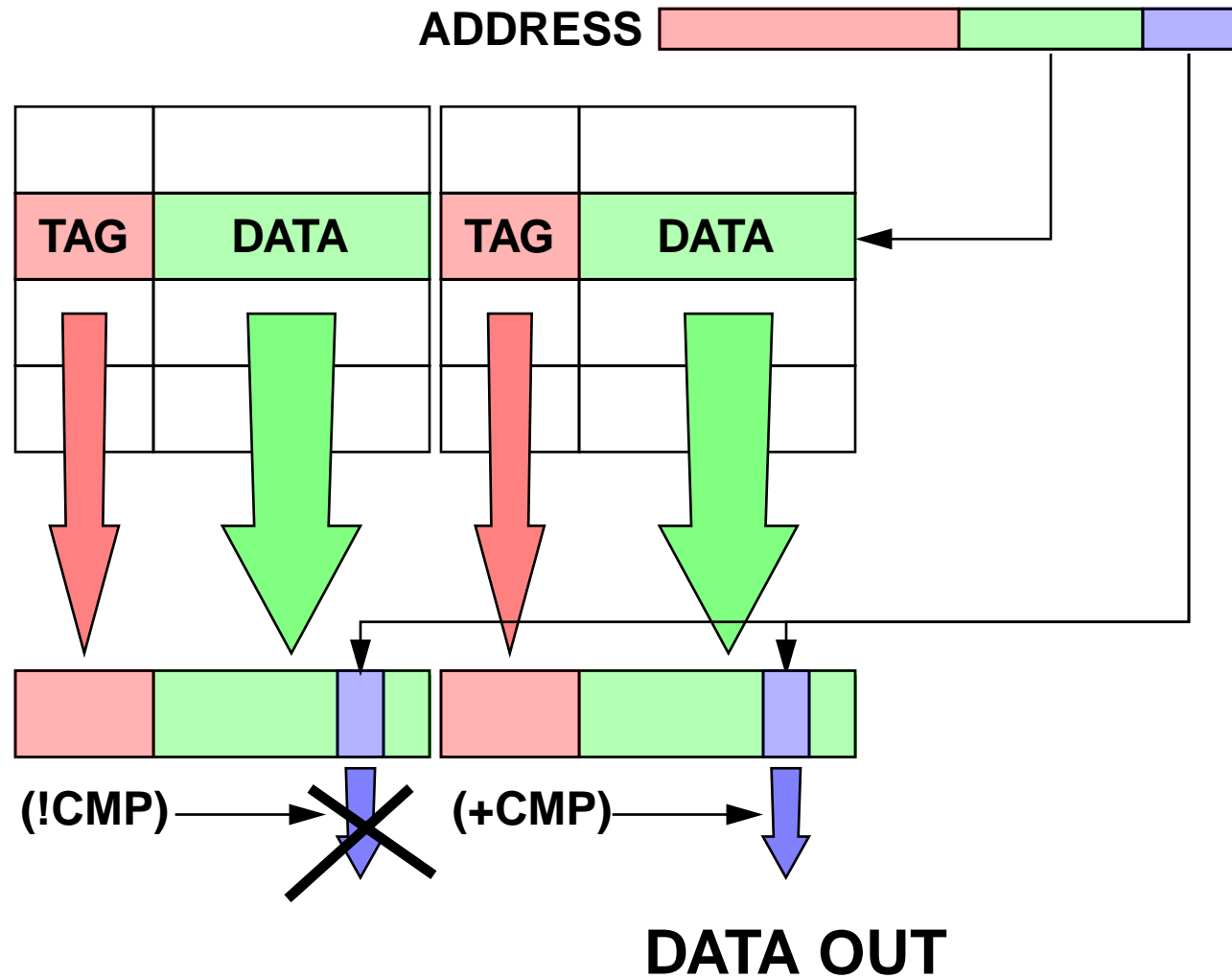
Cache Organizations

Associative Lookup



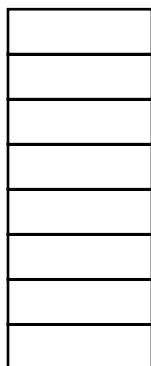
Cache Organizations

The Purpose of TAGS

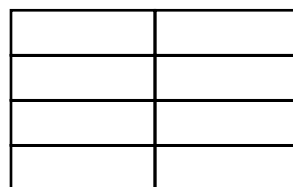


Cache Organizations

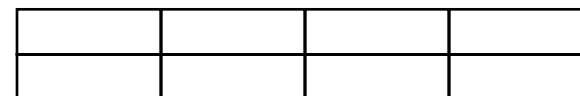
Given 8 cache blocks ...



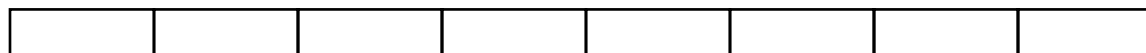
**Direct
Mapped**



**2-Way
Set Associative**



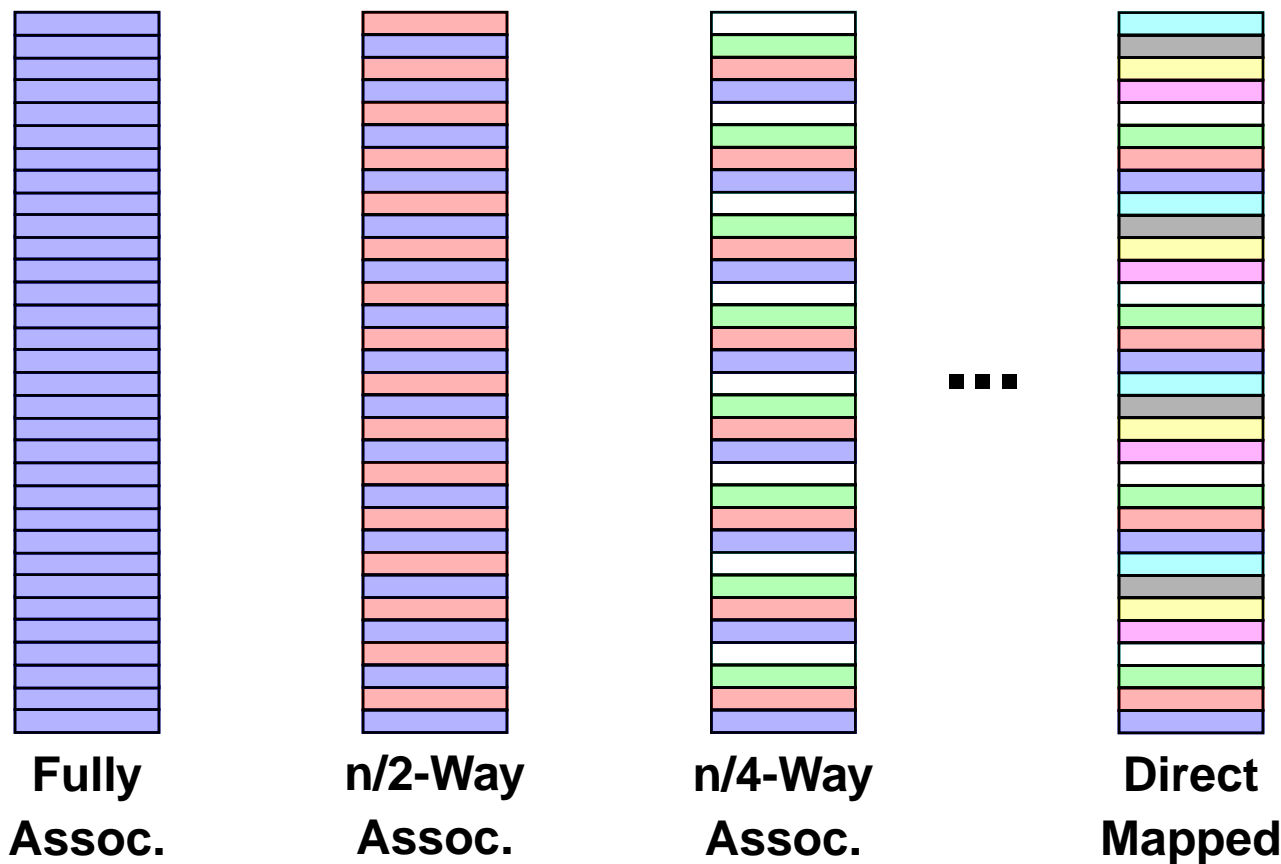
**4-Way
Set Associative**



**8-Way Set Associative
(Fully Associative, or Content-Addressable Memory)**

Cache Organizations

Associativity vs. the Memory Space



$n =$ blocks in cache

Memory Management

TRADITIONALLY:

The **MANAGEMENT** of
one's **USE** of **PHYSICAL MEMORY**

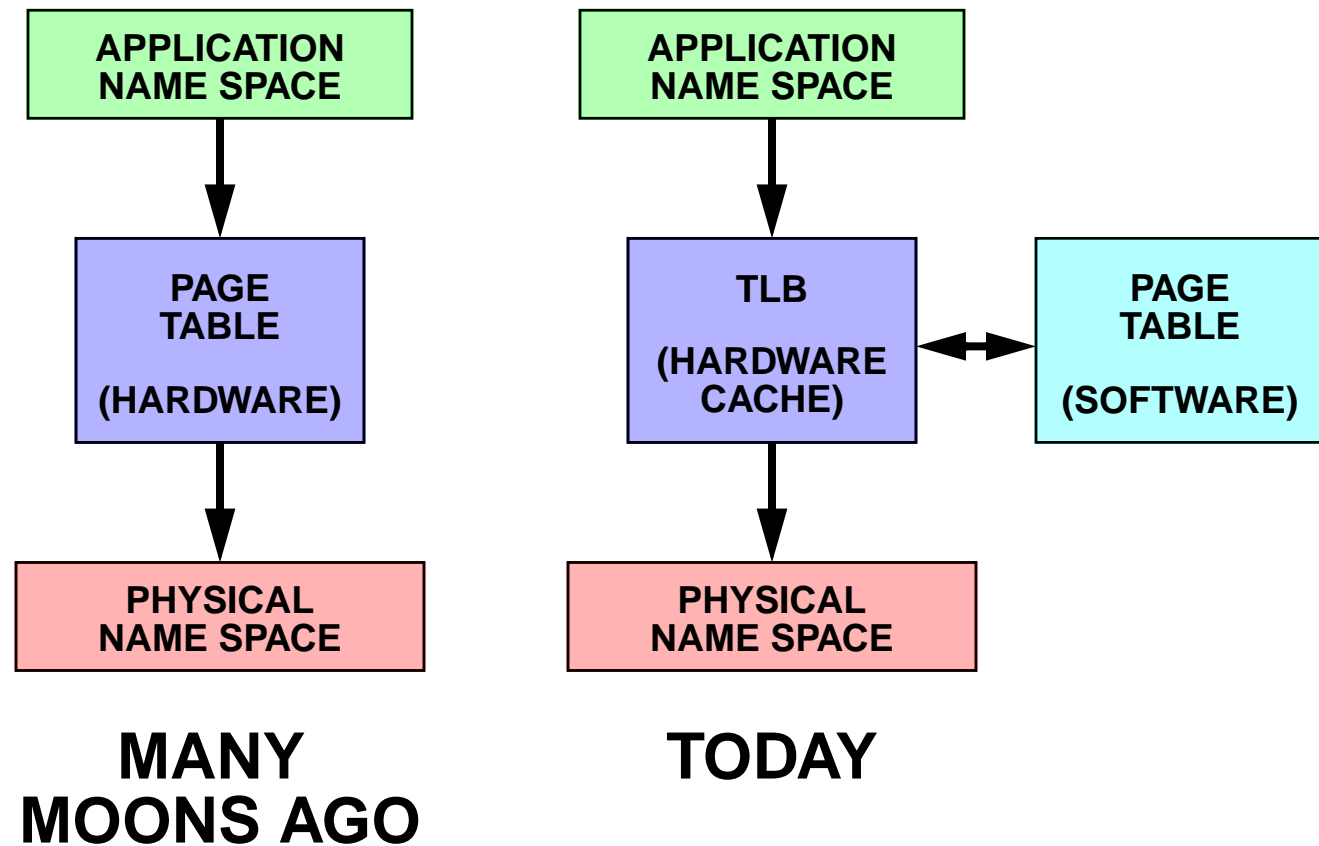
THIS TALK'S CONTEXT:

a **DIFFERENT NAMESPACE**
used for **ADDRESSING CACHES**

i.e. VIRTUAL ADDRESSING

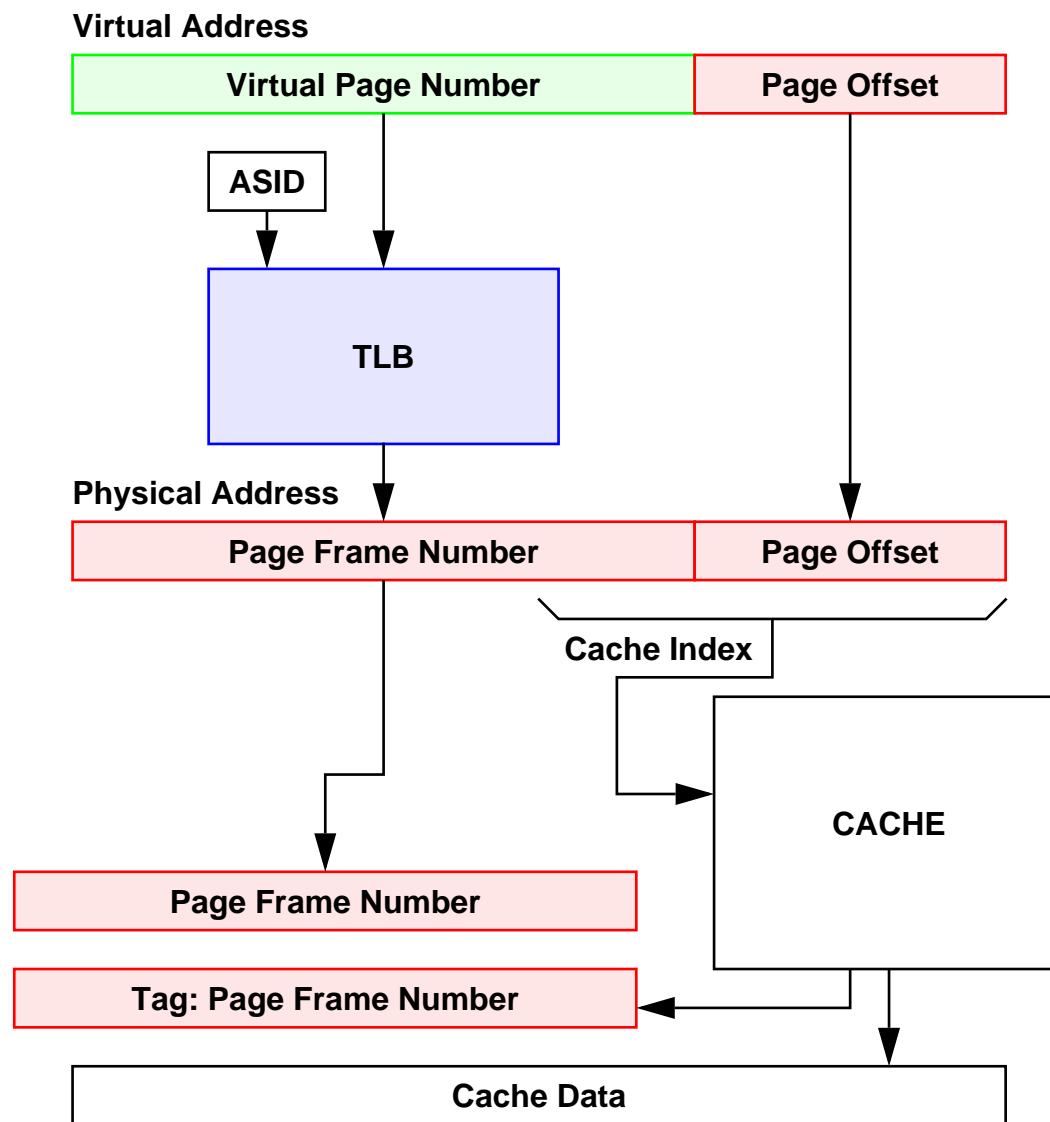
Memory Management

What is a TLB? (Translation Lookaside Buffer)



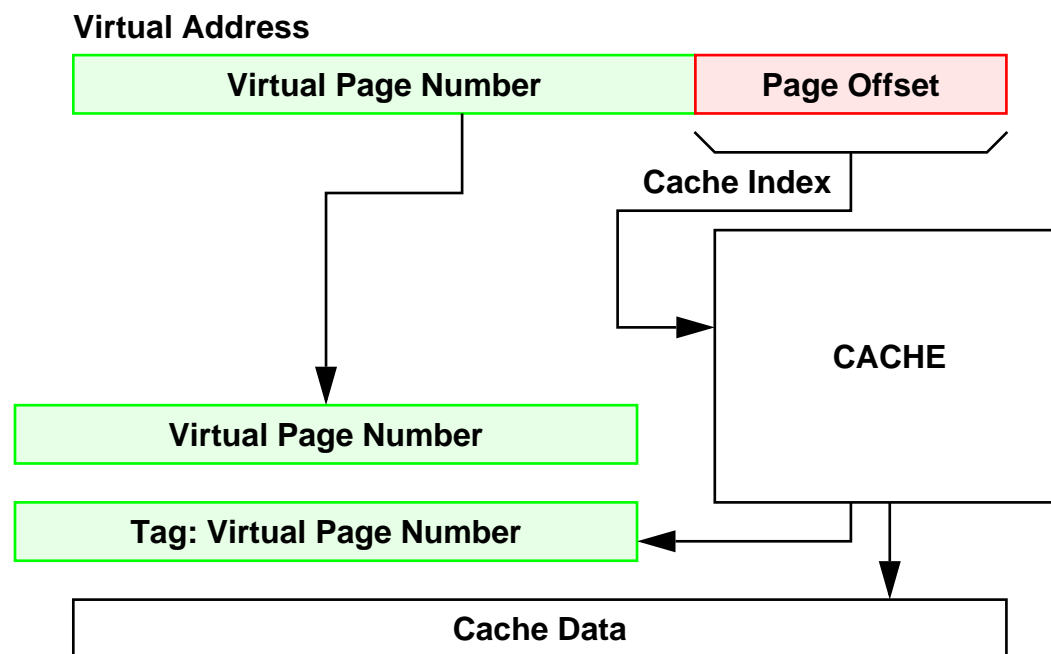
Cache Addressing

Physically Indexed, Physically Tagged



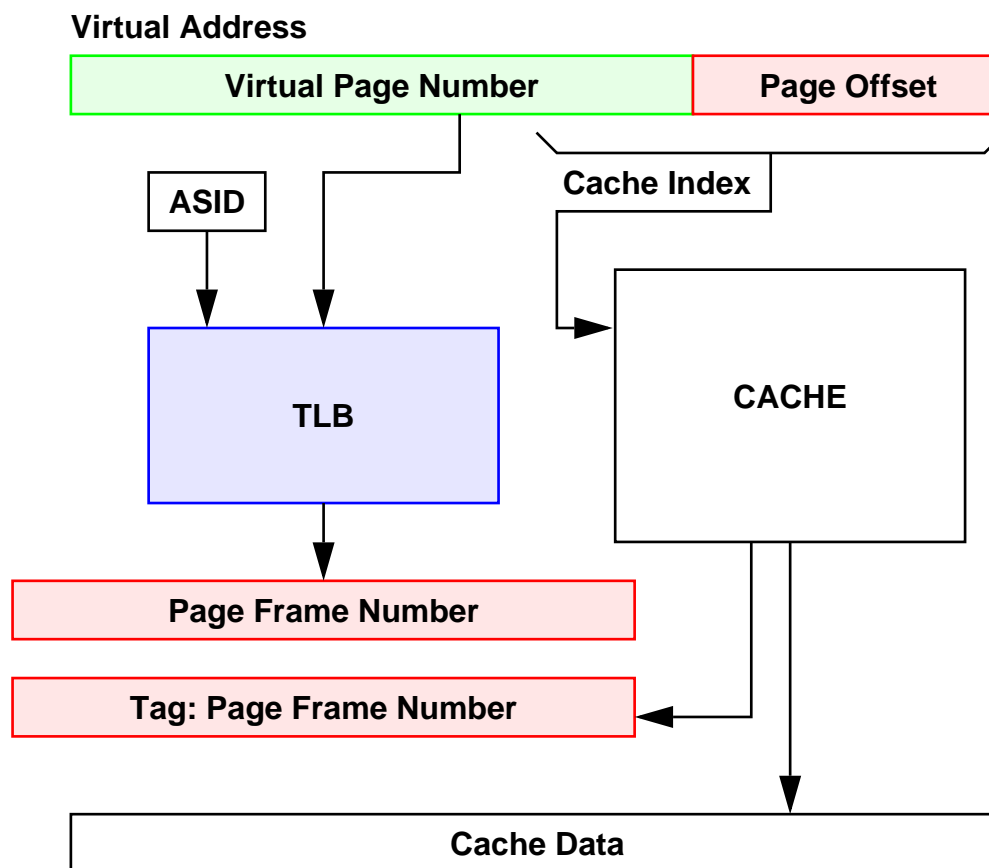
Cache Addressing

Physically Indexed, Virtually Tagged



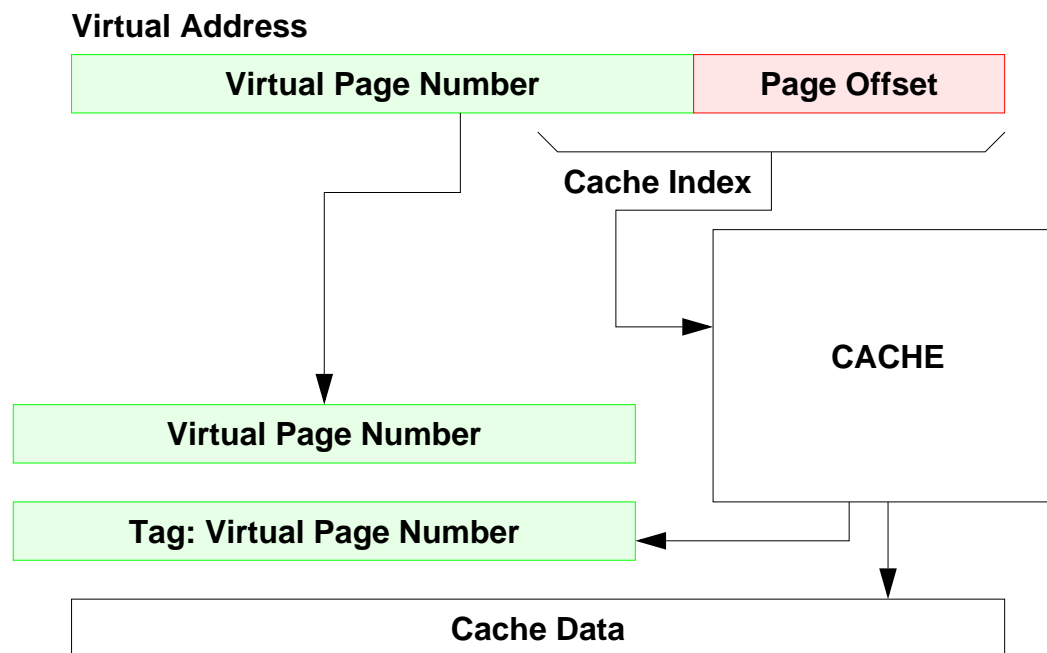
Cache Addressing

Virtually Indexed, Physically Tagged

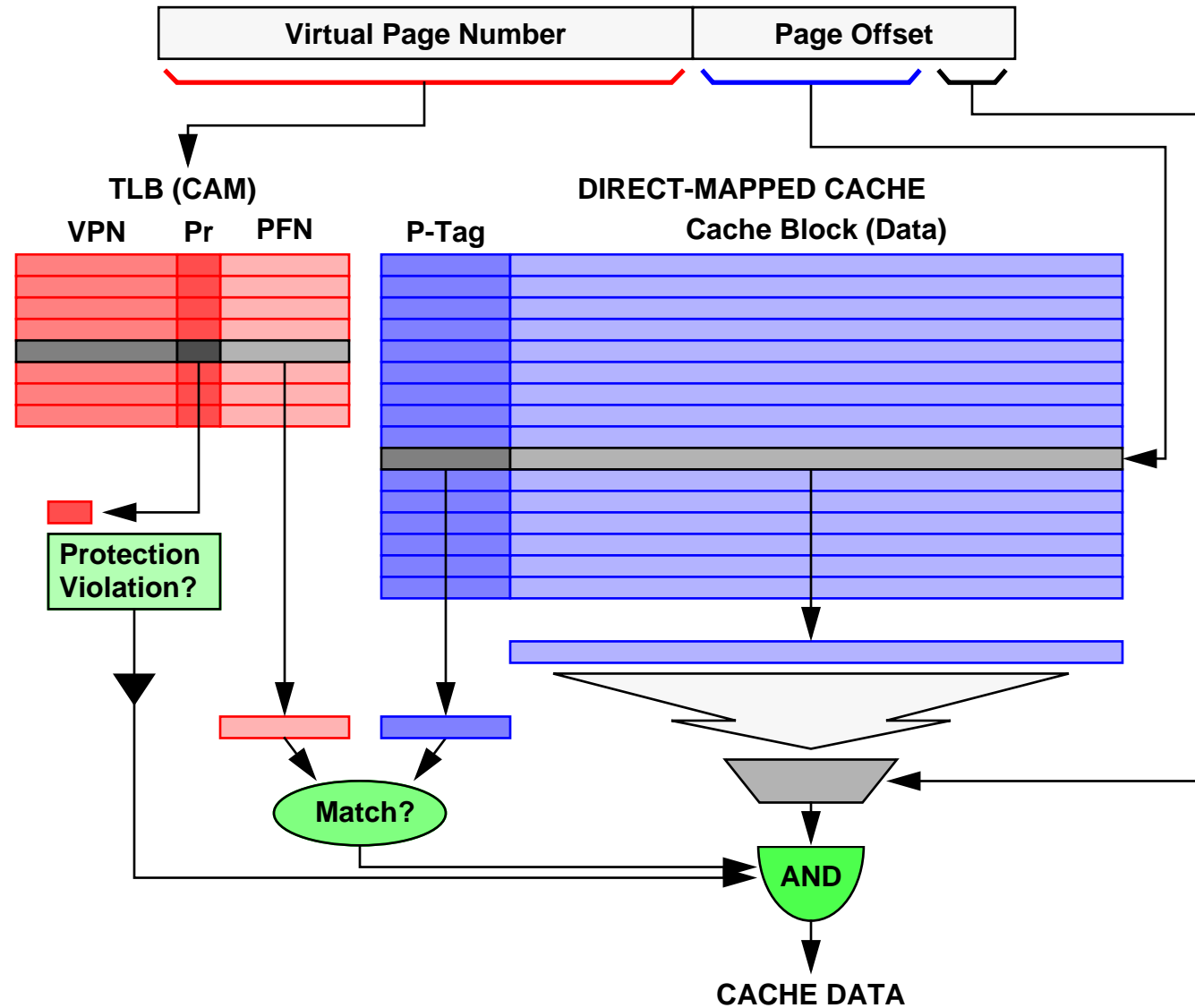


Cache Addressing

Virtually Indexed, Virtually Tagged



A Little More Detail



Memory Management

TRADITIONALLY:

The **MANAGEMENT** of
one's **USE** of **PHYSICAL MEMORY**

NEW DEFINITION:

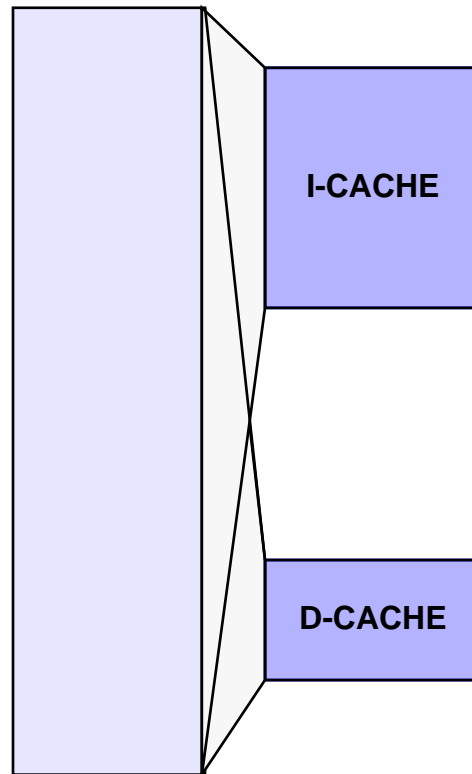
The **MANAGEMENT** of **ALL STRUCTURES**
associated with **MEMORY**

WHAT WE WILL SEE:

VIRTUAL ADDRESSING can help
SIMPLIFY memory management

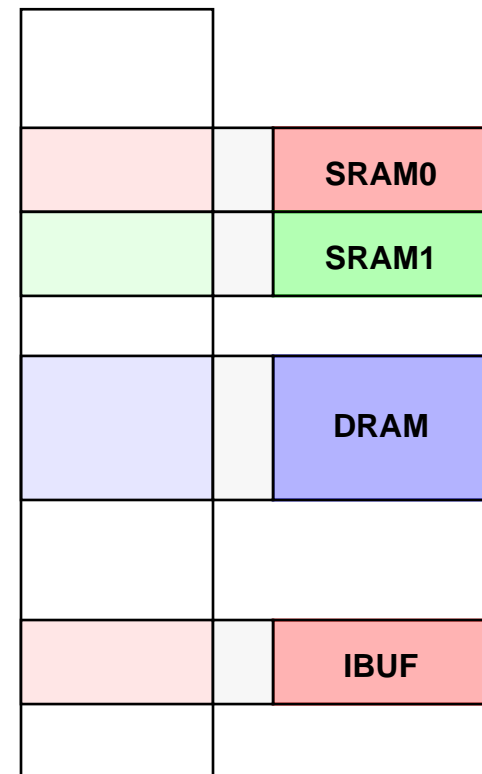
Cache vs. Scratch-pad RAM

UNIFORM
ADDRESS
SPACE



Traditional Caches

NON-UNIFORM
ADDRESS
SPACE



Scratch-Pad RAMs

**MAIN DIFFERENCE: scratch-pad requires
EXPLICIT MANAGEMENT**

Why Traditional Caches Suck



NON-DETERMINISM

Example #1

Refs: 0133ABABC01 3 34

Time →



HIT or **MISS**?

Set 3:

--

Set 2:

--

Set 1:

--

Set 0:

--

Example #1

Refs: **0**133ABABC01 3 34



Set 3:

Set 2:

Set 1:

Set 0:

0

Example #1

Refs: 0**1**33ABABC01 3 34



Set 3:

Set 2:

Set 1:

Set 0:

1
0

Example #1

Refs: 01**3**3ABABC01 3 34



Set 3:

3

Set 2:

Set 1:

1

Set 0:

0

Example #1

Refs: 0133ABABC01 3 34



Set 3:

3

Set 2:

Set 1:

1

Set 0:

0

Example #1

Refs: 0133**A**BABC01 3 34



Set 3:

3

Set 2:

A

Set 1:

1

Set 0:

0

Example #1

Refs: 0133A**B**ABC01 3 34



Set 3:

B

Set 2:

A

Set 1:

1

Set 0:

0

Example #1

Refs: 0133ABABC01 3 34



Set 3:	B
Set 2:	A
Set 1:	1
Set 0:	0

Example #1

Refs: 0133ABA**B**C01 3 34



Set 3:

B

Set 2:

A

Set 1:

1

Set 0:

0

Example #1

Refs: 0133ABAB**C**01 3 34



Set 3:

B

Set 2:

A

Set 1:

1

Set 0:

C

Example #1

Refs: 0133ABABC**0**1 3 34



Set 3:

B

Set 2:

A

Set 1:

1

Set 0:

0

Example #1

Refs: 0133ABABC0**1** 3 34



Set 3:

B

Set 2:

A

Set 1:

1

Set 0:

C

Example #1

Refs: 0133ABABC01 **3** 34



Set 3:

3

Set 2:

A

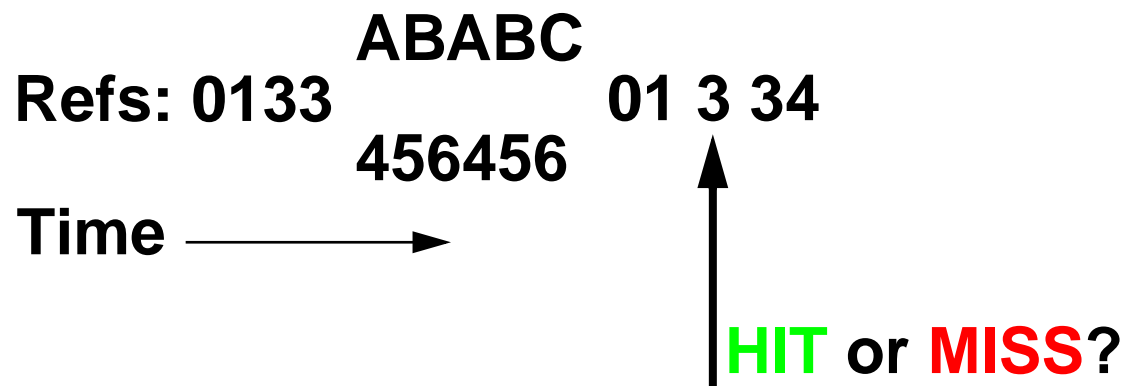
Set 1:

1

Set 0:

C

Example #2



Set 3:

Set 2:

Set 1:

Set 0:

Example #2

Refs: **0**133 ABABC 01 3 34
 ↑ 456456

Set 3:

Set 2:

Set 1:

Set 0:

0

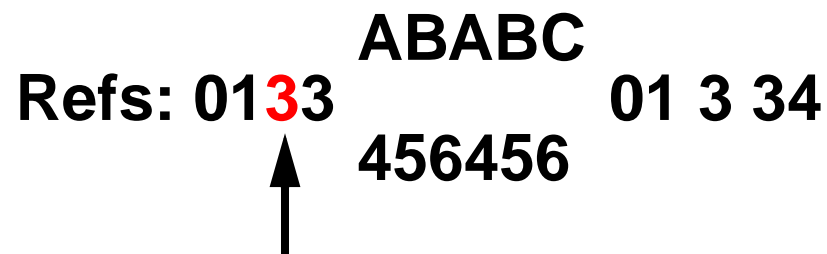
Example #2

Refs: 0**1**33 ABABC 01 3 34
 ↑ 456456

Set 3:	
Set 2:	
Set 1:	1
Set 0:	0

Example #2

Refs: 01**33** ABABC 01 3 34
456456



Set 3:

3

Set 2:

Set 1:

1

Set 0:

0

Example #2

Refs: 0133 ABABC 01 3 34
 456456

Set 3:	3
Set 2:	
Set 1:	1
Set 0:	0

Example #2

Refs: 0133 **ABABC** 01 3 34
 456456
 ↑

Set 3:	3
Set 2:	
Set 1:	1
Set 0:	4

Example #2

Refs: 0133 **ABABC** 01 3 34
 4**5**6456
 ↑

Set 3:	3
Set 2:	
Set 1:	5
Set 0:	4

Example #2

Refs: 0133 **ABABC** 01 3 34
 45**6**456
 ↑

Set 3:	3
Set 2:	6
Set 1:	5
Set 0:	4

Example #2

Refs: 0133 **ABABC** 01 3 34
 456**4**56
 ↑

Set 3:

3

Set 2:

6

Set 1:

5

Set 0:

4

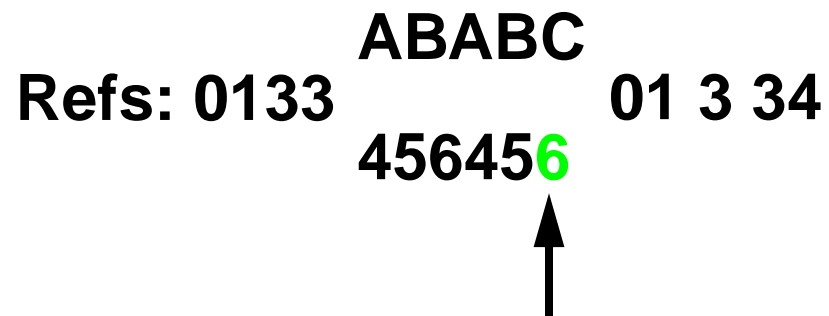
Example #2

Refs: 0133 **ABABC** 01 3 34
 4564**5**6
 ↑

Set 3:	3
Set 2:	6
Set 1:	5
Set 0:	4

Example #2

Refs: 0133 **ABABC** 01 3 34
 45645**6**



Set 3:

3

Set 2:

6

Set 1:

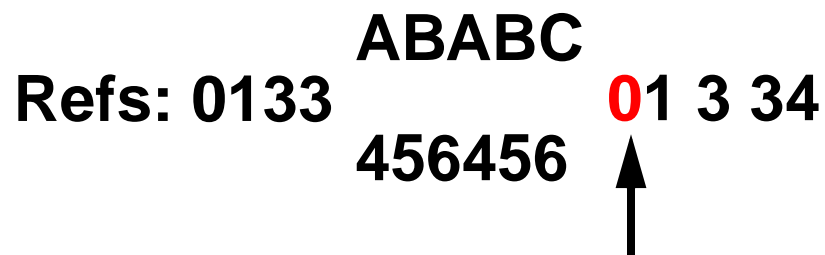
5

Set 0:

4

Example #2

Refs: 0133 ABABC
456456 01 3 34



Set 3:

3

Set 2:

6

Set 1:

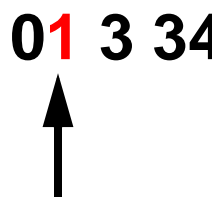
5

Set 0:

0

Example #2


Refs: 0133 ABABC
456456 01 3 34



Set 3:	3
Set 2:	6
Set 1:	1
Set 0:	0

Example #2

Refs: 0133 ABABC
456456 01 **3** 34



Set 3:

3

Set 2:

6

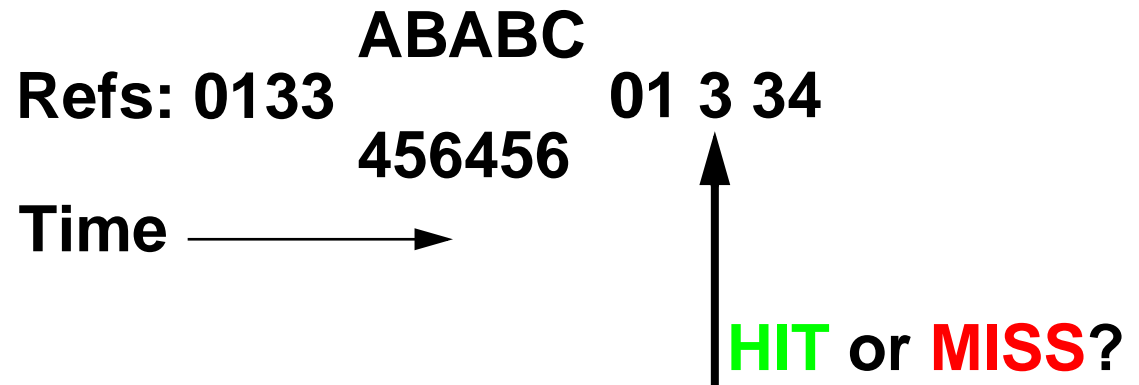
Set 1:

1

Set 0:

0

Example #3



Set 3:

--	--

Set 2:

--	--

Set 1:

--	--

Set 0:

--	--

Example #3

Refs: **0**133 ABABC 01 3 34
 ↑ 456456

Set 3:		
Set 2:		
Set 1:		
Set 0:	0	

Example #3

Refs: 0133 ABABC 01 3 34
 ↑ 456456

Set 3:		
Set 2:		
Set 1:	1	
Set 0:	0	

Example #3

Refs: 01**33** ABABC 01 3 34
456456



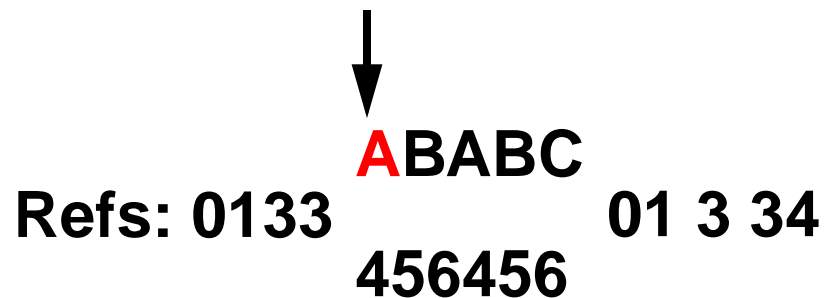
Set 3:	3	
Set 2:		
Set 1:	1	
Set 0:	0	

Example #3

Refs: 0133 ABABC 01 3 34
456456

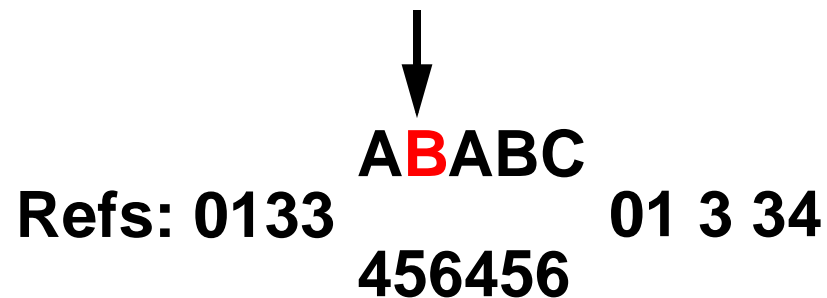
Set 3:	3	
Set 2:		
Set 1:	1	
Set 0:	0	

Example #3



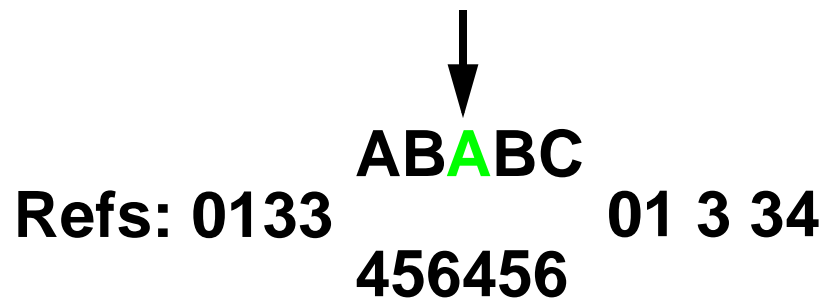
Set 3:	3	
Set 2:		A
Set 1:	1	
Set 0:	0	

Example #3



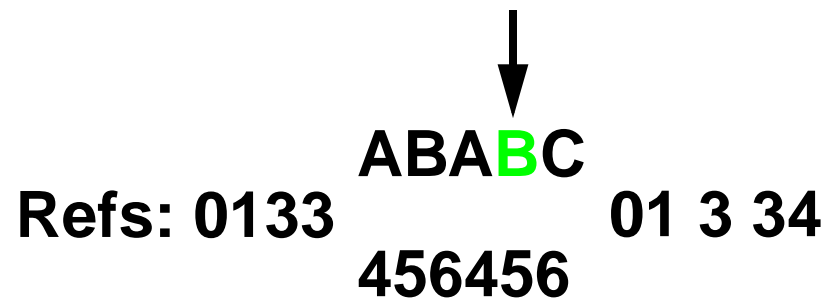
Set 3:	3	B
Set 2:		A
Set 1:	1	
Set 0:	0	

Example #3



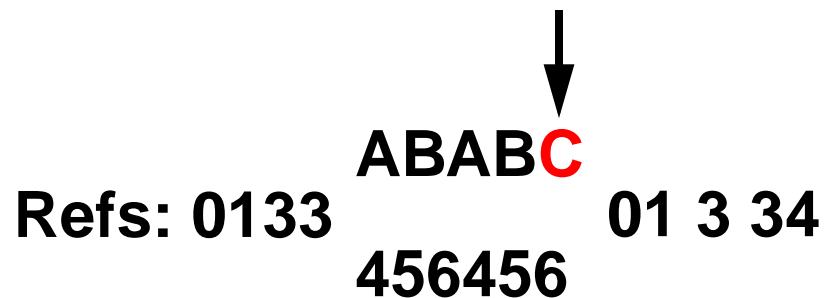
Set 3:	3	B
Set 2:		A
Set 1:	1	
Set 0:	0	

Example #3



Set 3:	3	B
Set 2:		A
Set 1:	1	
Set 0:	0	

Example #3



Set 3:	3	B
Set 2:		A
Set 1:	1	
Set 0:	0	C

Example #3

Refs: 0133 ABABC
456456 01 3 34
↑

Set 3:	3	B
Set 2:		A
Set 1:	1	
Set 0:	0	C


Example #3

Refs: 0133 ABABC
456456 01 3 34
↑

Set 3:	3	B
Set 2:		A
Set 1:	1	
Set 0:	0	C

Example #3

Refs: 0133 ABABC
456456 01 **3** 34



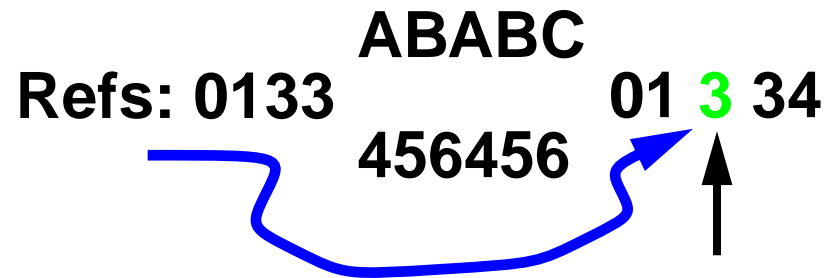
Set 3:	3	B
Set 2:		A
Set 1:	1	
Set 0:	0	C

Example #3



Set 3:	3	B
Set 2:		A
Set 1:	1	
Set 0:	0	C

Example #3



Set 3:	3	
Set 2:		6
Set 1:	1	5
Set 0:	0	4

Example #3



HIT?
or MISS?

Set 3:	3	-/B
Set 2:		6/A
Set 1:	1	5/-
Set 0:	0	4/C

Traditional Caches

Require TAGS

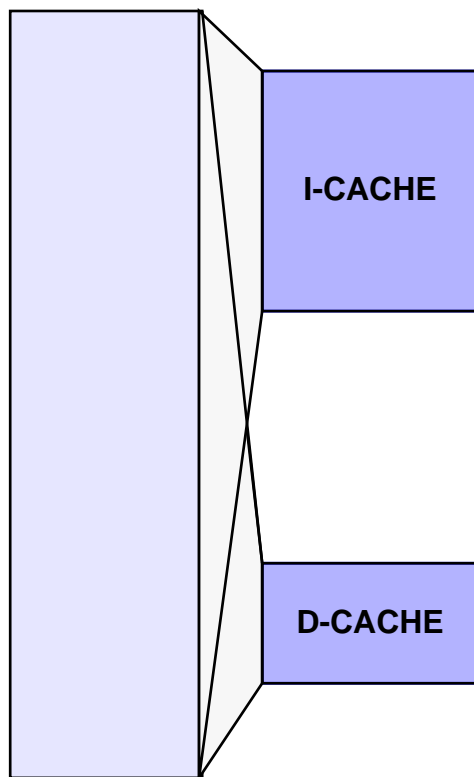
**Soon into program execution,
contents of cache are indeterminate
(thus the term “hit rate” for performance)**

**Set associativity delays problems,
but only to a point**

**Associativity > 2 does not implement
TRUE Least-Recently-Used**

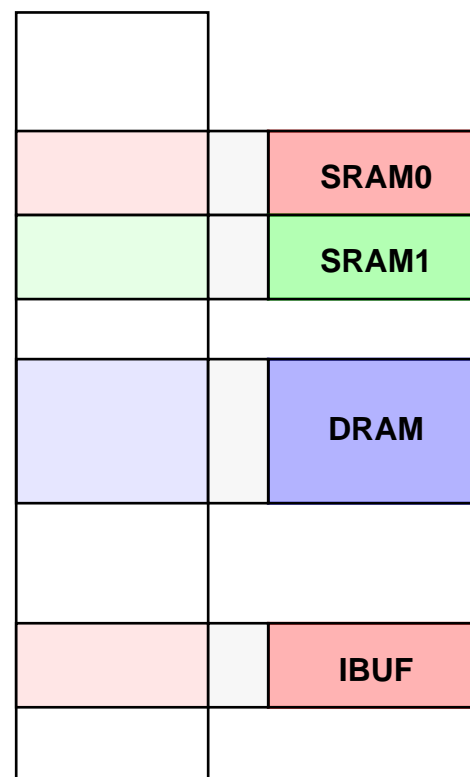
Scratch-pad RAMs (again)

UNIFORM
ADDRESS
SPACE



Traditional Caches

NON-UNIFORM
ADDRESS
SPACE



Scratch-Pad RAMs

Require **EXPLICIT MANAGEMENT**

Scratch-Pad RAMs

No TAGS (save die area)

As long as everything fits, GREAT!

Otherwise, addressing is impediment:

CONTIGUITY must be preserved

DISTANCE BETWEEN OBJECTS

must be preserved

DSPs go one step further:

Multiply-accumulate requires

TWO DISJOINT DATA SPACES

Scratch-Pad RAMs

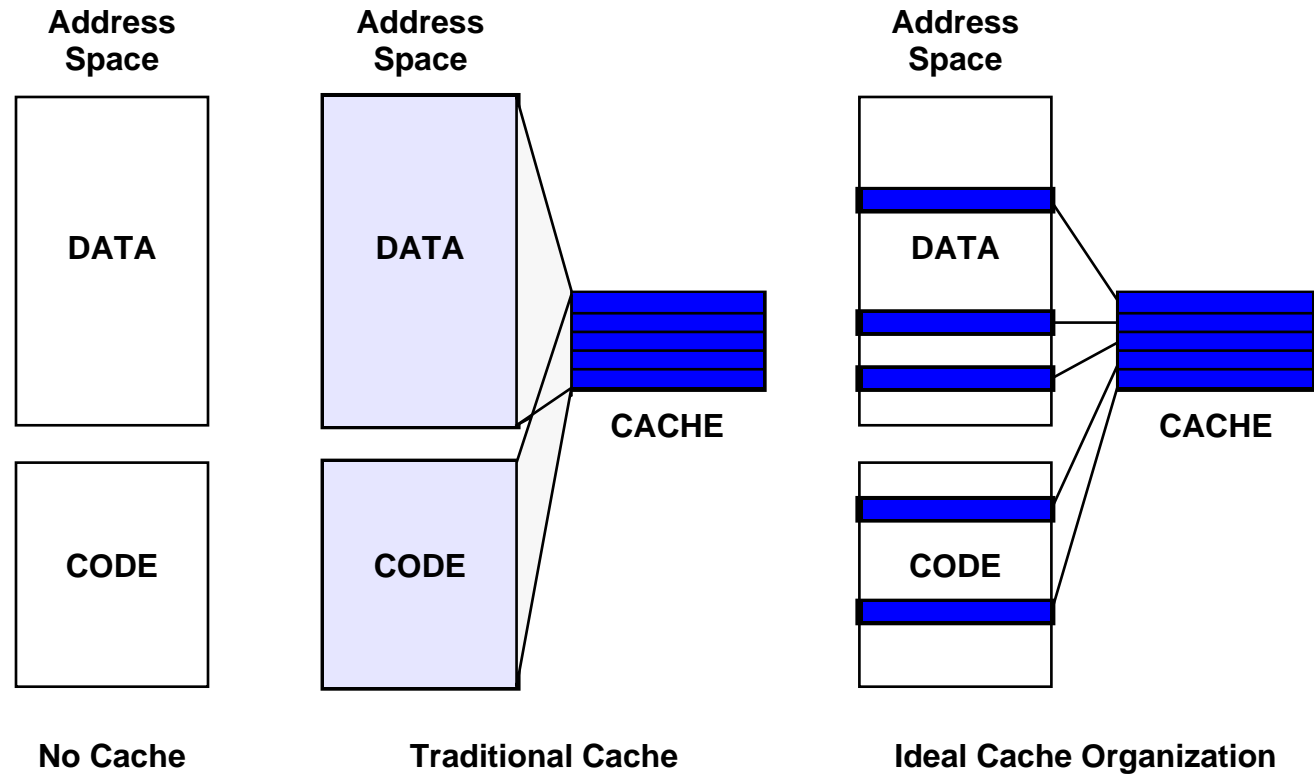
Access to memory is NON-ORTHOGONAL
Separate spaces are DISJOINT




Bottom Line: COMPILATION IS HARD

Trend: UNIFORM ADDRESS SPACES

(i.e. more like traditional caches)

WHAT WE WANT



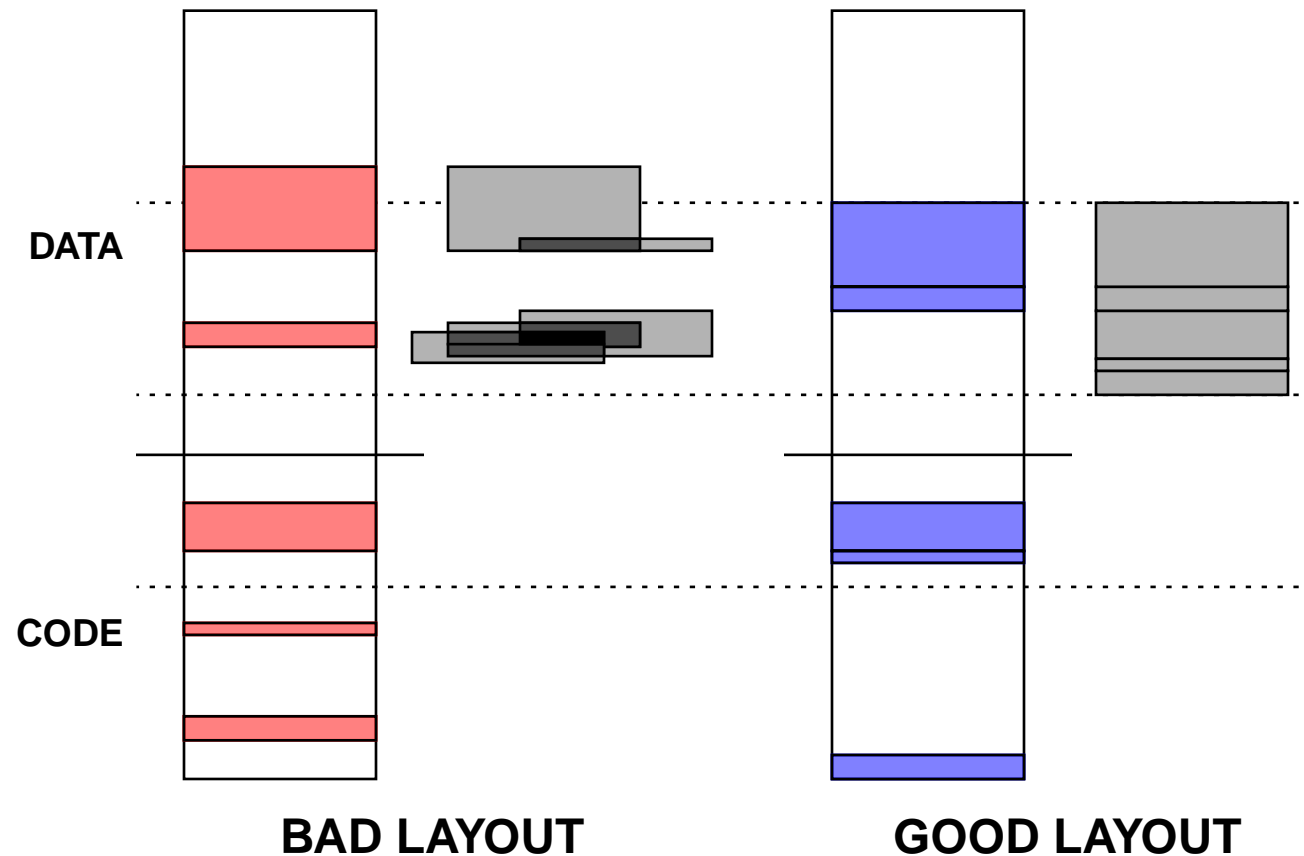
-  Guaranteed slow access-time
-  Statistically fast access-time
-  Guaranteed fast access-time

WHY IT'S DIFFICULT

DATA NAME => DATA PLACEMENT

Must Group Data & Instructions

So as to Minimize Cache Conflicts



Data Placement

DATA SPACE

- **Relatively easy to rearrange items ...**
- **... Unless part of a LARGER ITEM
(cannot rearrange array elements)**

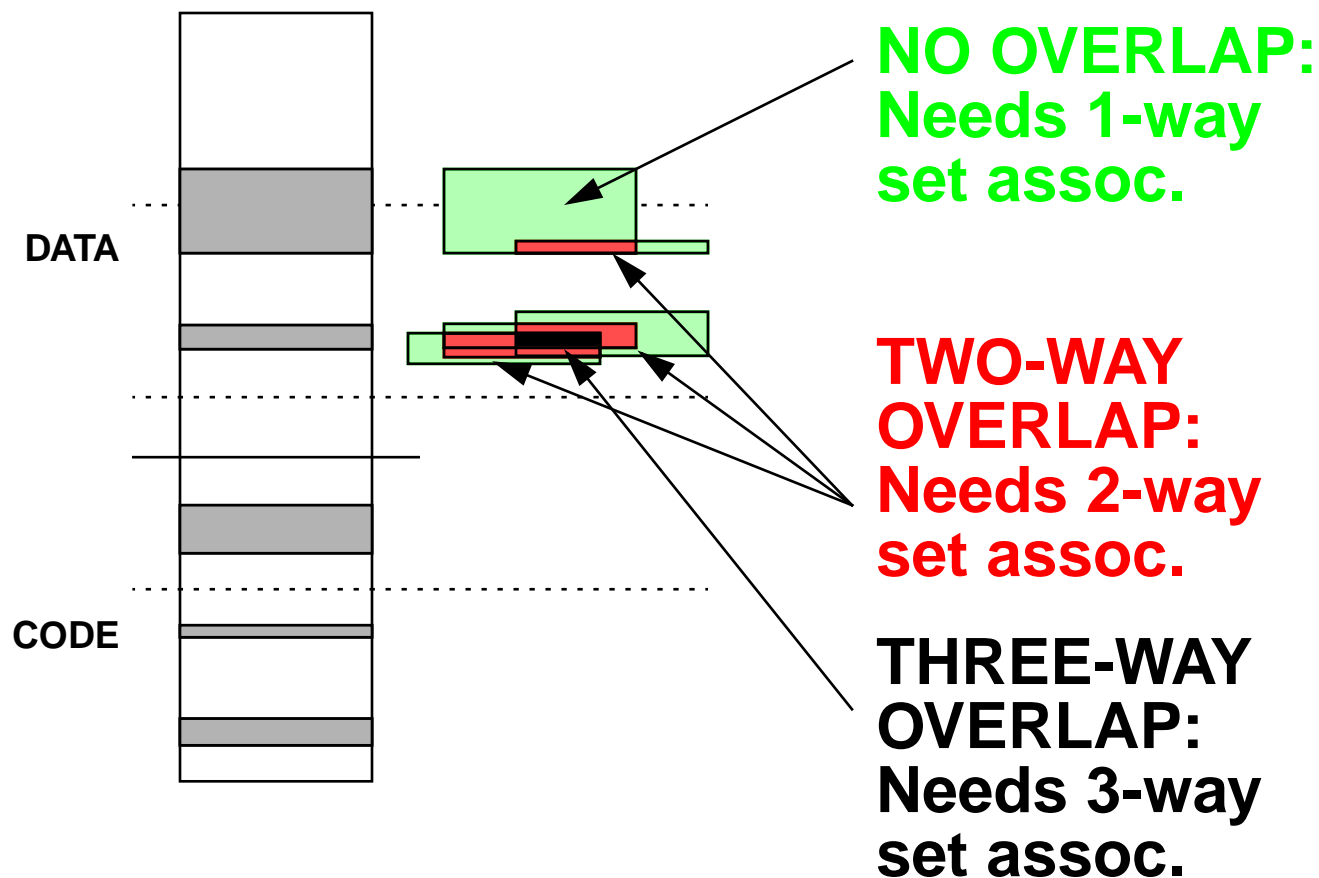
CODE SPACE

- **Can move FUNCTIONS around easily**
- **PORTIONS of code is another matter ...**

THERE IS A FAMILIAR SOLUTION ...

Solution #1

A BIG, HIGHLY ASSOCIATIVE CACHE
+ ability to PIN DOWN CACHE LINES



Solution #1

- **Choose items to cache,
Bring each into the cache,
Pin each down**
- **Can CACHE/NOT-CACHE adjacent items**
- **Must know CACHE ORGANIZATION
at COMPILE TIME
(not huge issue for embedded systems)**
- **SIMPLEST, but perhaps
MOST EXPENSIVE solution**

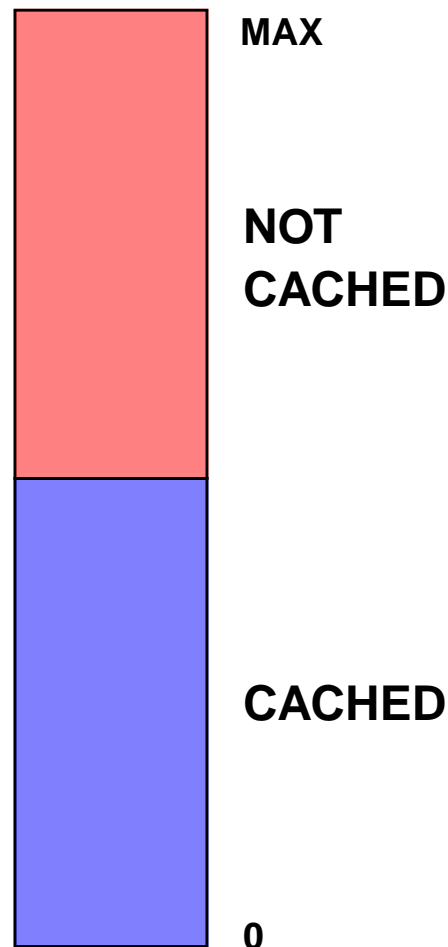
Solution #2 (var. on #1)

Software-Managed Caches

Top bits determine
memory-access behavior
(CACHED/NON-CACHED)

Other possibilities:

- Physical/virtual
- Faulting/non-faulting
- Which cache or
memory structure



Enables on-the-fly decision-making

Re: memory behavior

Application Behavior

```
int *array = malloc (N * sizeof int); // YES
```

```
int *stream = malloc (N * sizeof int); // NO
```

```
int *mix = malloc (N * sizeof int); // MAYBE
```

```
for (i=0; i<N; i++)
```

```
    x = array[i]; // CACHED REFERENCE
```

```
stream |= MIN_NEG_INT; // 0x80000000
```

```
for (i=0; i<N; i++)
```

```
    x = stream[i]; // NON-CACHED
```

```
for (i=0; i<N; i++) // DEPENDS ON cache_it
```

```
    x = (cache_it (i)) ? mix[i]
```

```
        : (mix | MIN_NEG_INT)[i];
```

Solution #2

Advantages over Solution #1:

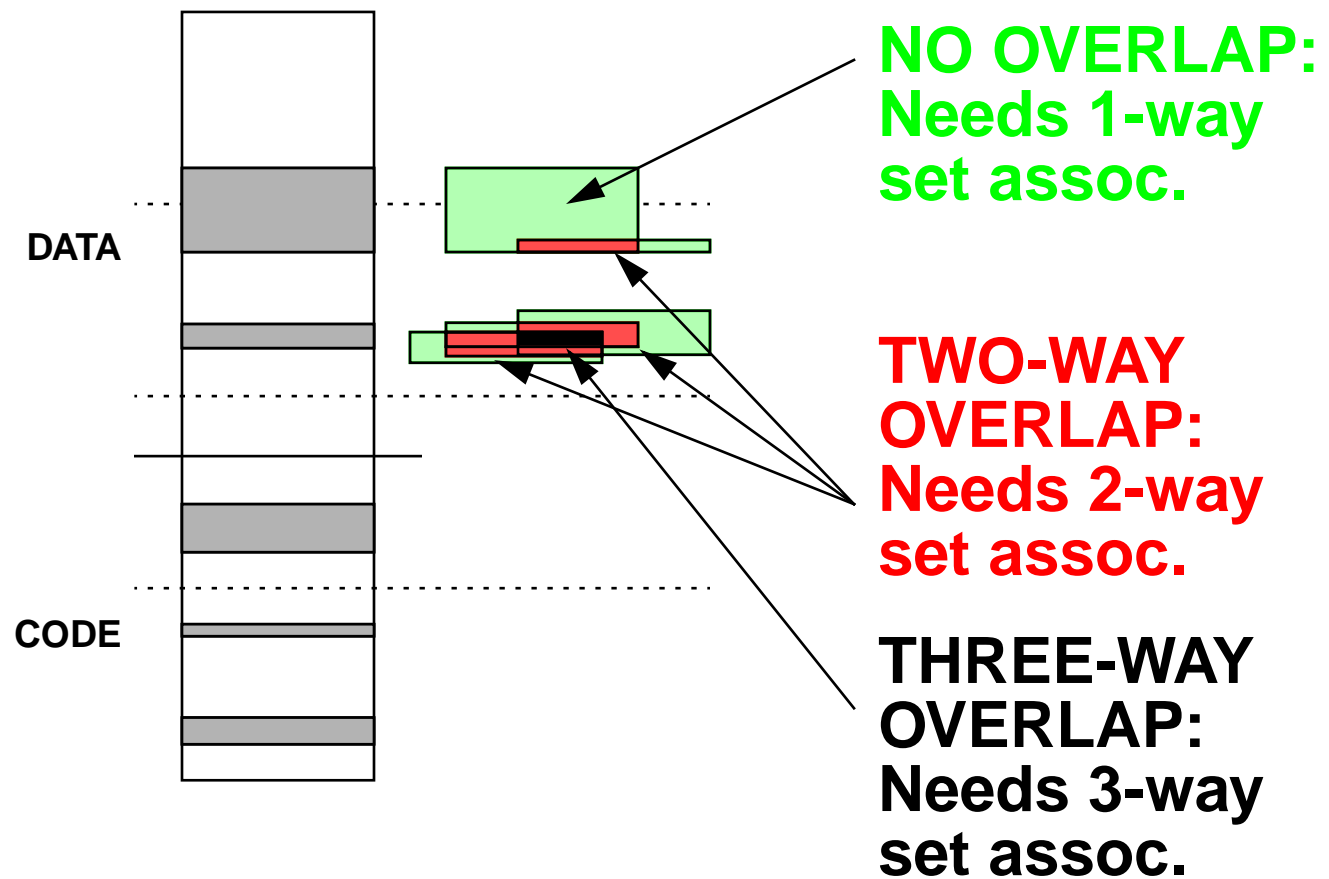
Allows **DYNAMIC CACHE DECISIONS**
LESS TIED to **CACHE ORGANIZATION**

Many of the same weaknesses:

Requires **BIG CACHE**
Requires **SET ASSOCIATIVE CACHE**
Have to deal with **DATA PLACEMENT ...**

Issue: Data Placement

DATA NAME => DATA PLACEMENT



TO MINIMIZE OVERLAP, RELOCATE DATA

Issue: Data Placement

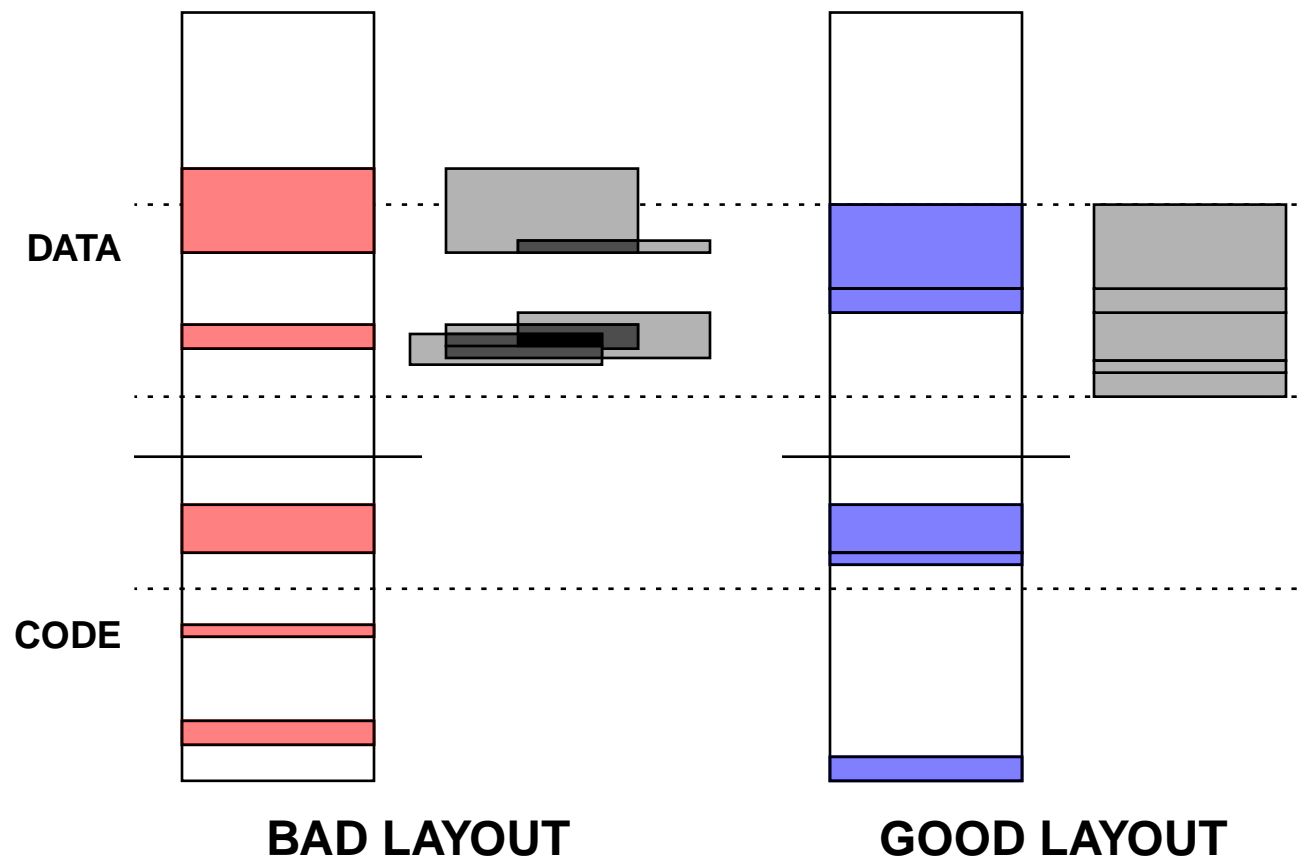
GOALS:

- Disassociate **NAME** and **PLACEMENT**
- Fine-grained code/data relocation at granularity of **TLB page** or (better) **cache line**

Enter Virtual Memory

Disassociates NAME from PLACE

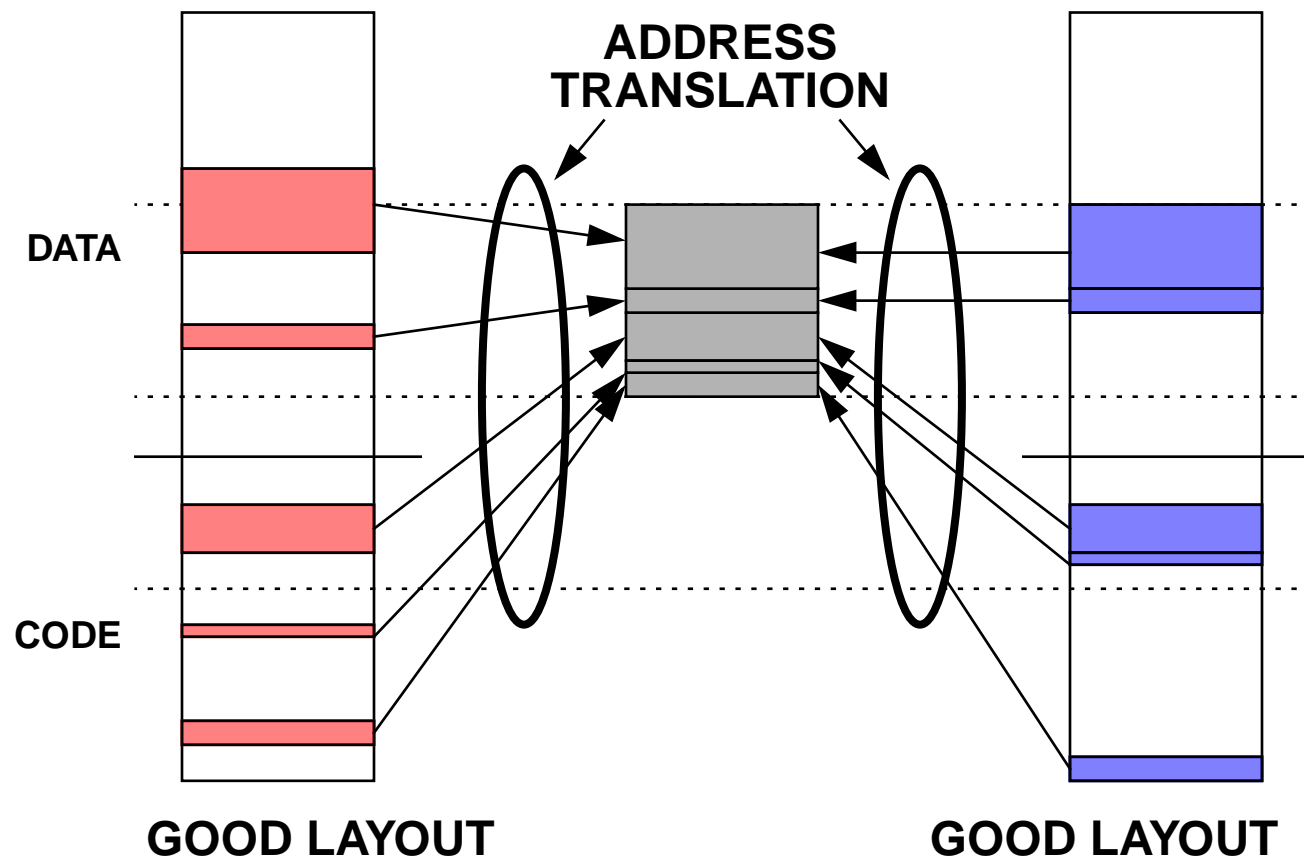
Allows you to go from **THIS**:



Enter Virtual Memory

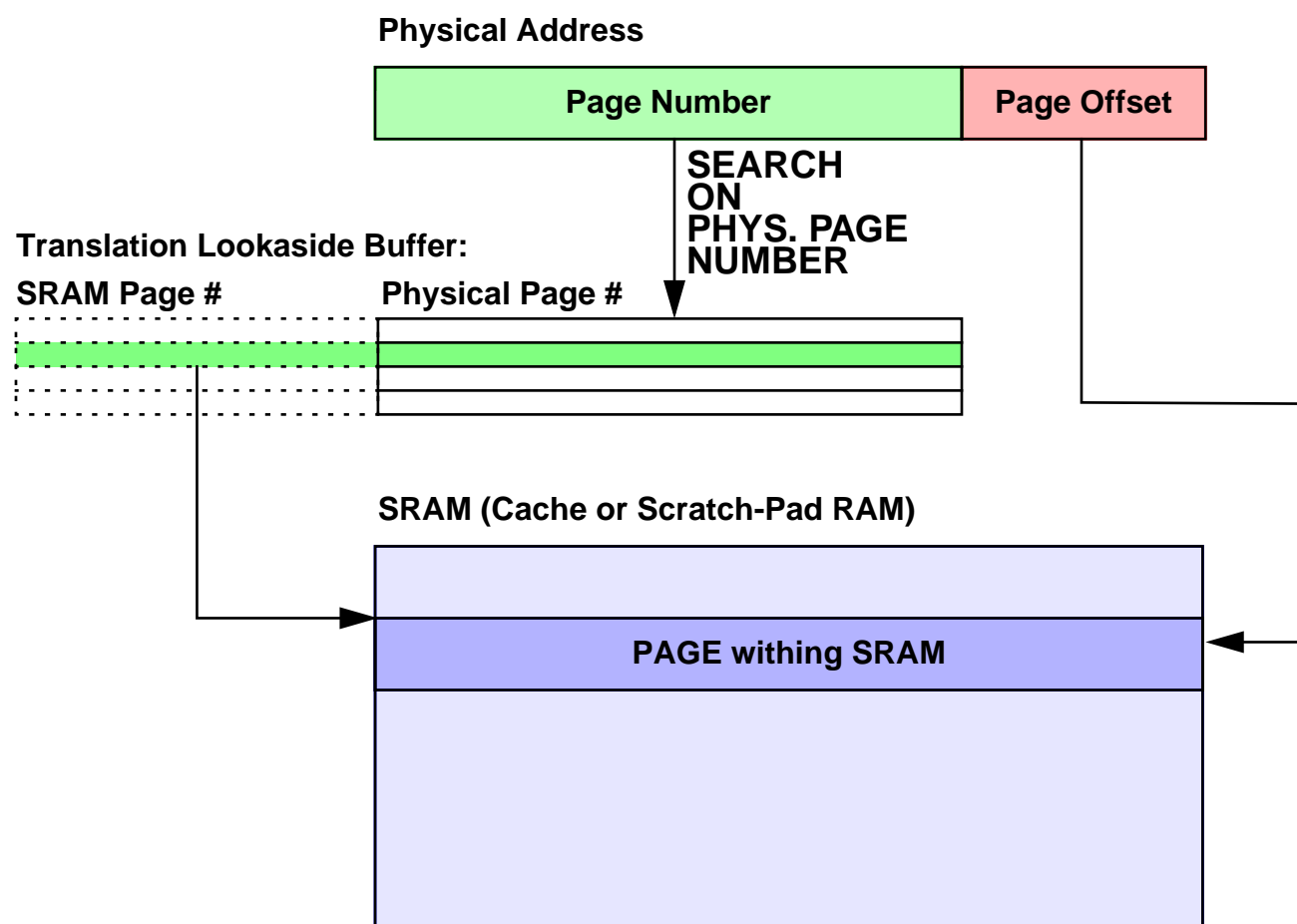
Disassociates NAME from PLACE

... to **THIS**:



Real-Time TLB Organization

Works with either **CACHE** or **SCRATCH-PAD**



Solution #3

Fully-Associative Real-Time TLB + Direct-Mapped SRAM

- TLB must fully map SRAM
(8KB SRAM, 256-byte page => 32 entries)
- Can place ANY 256-byte page
ANYWHERE in the SRAM
- Benefit: **simple SRAM design**
- Drawback: **fully assoc. TLB**

Variations on Solution #3

WANT A **LARGER CACHE?**

- Larger TLB
- Larger Page Size

WANT A **SMALLER TLB?**

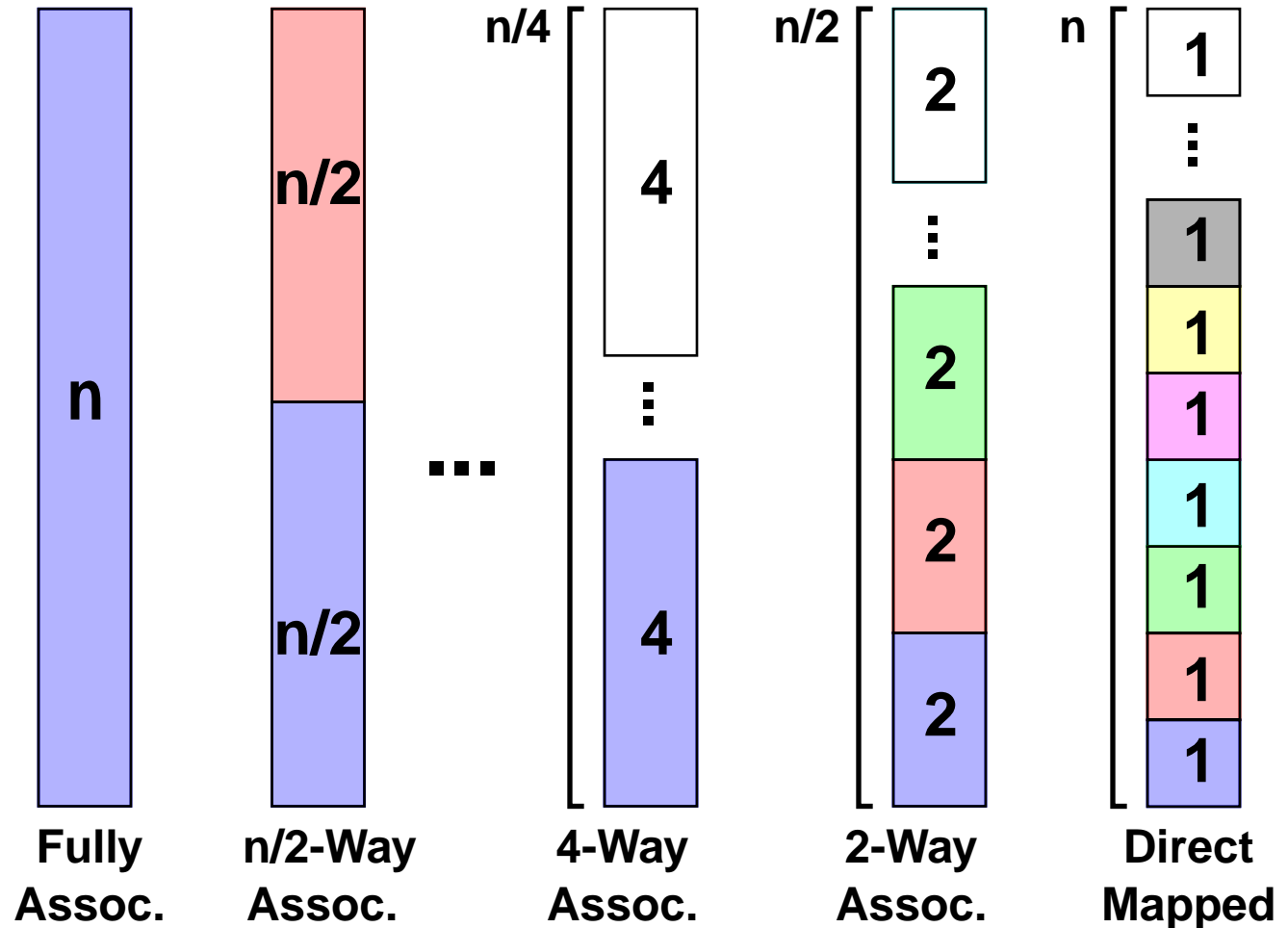
- Smaller Cache
- Larger Page Size

WANT **LESS ASSOCIATIVITY?**

- That's a little more involved ...

Set-Associative RT-TLBs

Associativity vs. the Memory Space



$n =$ entries in TLB

Set-Associative RT-TLBs

LIMITING CASE:

Direct-Mapped TLB

Direct-Mapped SRAM

Same set of data placement problems
we had with NO TLB ...

EXCEPT: contiguity restriction lifted

Bottom Line: **PROBABLY NOT WORTH IT**

INTERMEDIATE SOLUTIONS:

Obvious Trade-Offs Exist

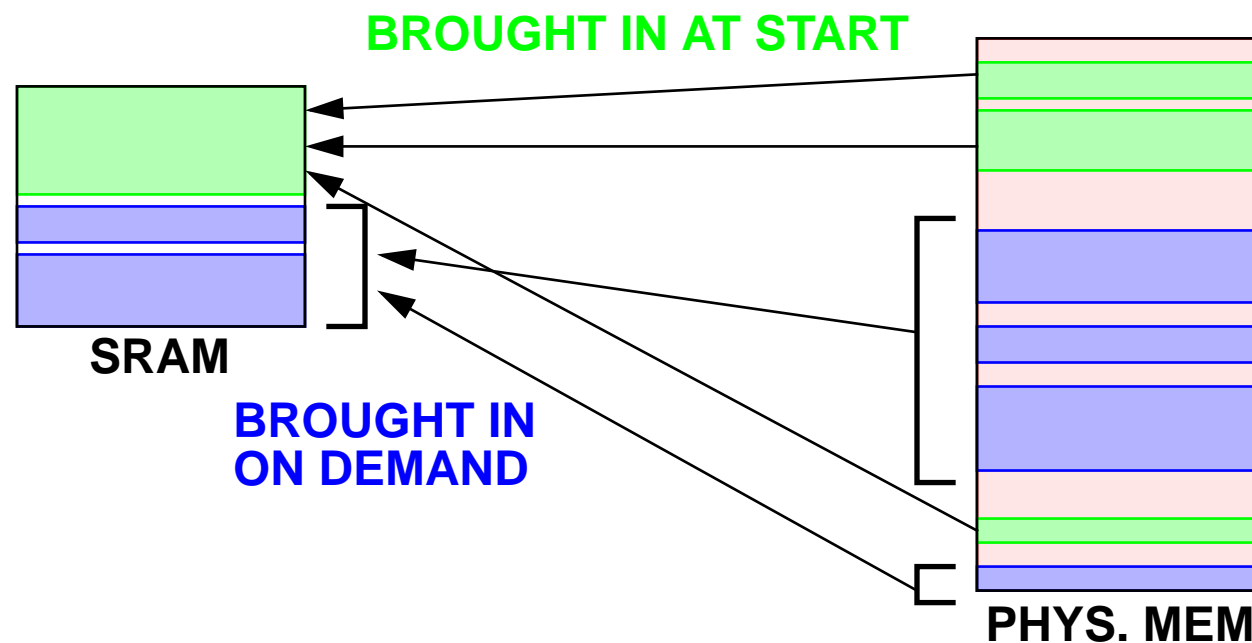
NEED MORE INVESTIGATION

Solution #4

What if SRAM Still Too Small?

(i.e. — previous solution reduces **CONFLICT** problems, not **CAPACITY** problems)

Real-Time SRAM-Management



Real-Time SRAM Management

CLASSIFY ALL CODE & DATA:

- **MUST ALWAYS REMAIN CACHED**
- **MUST NEVER BE CACHED**
- **EXHIBITS PERIODIC LOCALITY**
(i.e. loop code & data)

FOR PERIODIC ITEMS:

- Add code at beginning to set up TLB
- Add code at end to unmap TLB
and write out any dirty values

Real-Time SRAM Management

RESULTS:

- **VM-style extending of SRAM space into DRAM space via demand-paging**
- **PROACTIVE demand-paging, not REACTIVE demand-paging**
- **Deterministic memory performance for all references**
- **Slight overhead in code size & execution**

Summary

UNIFORM MEMORY SPACES:

- **Provide Orthogonal Look at Memory**
- **Cache Architectures Exhibit Non-Deterministic Performance**

NON-UNIFORM MEMORY SPACES:

- **Non-Orthogonal Memory Map**
- **Caches Offer Deterministic Performance (at the Price of Explicit Management)**

TREND IS TOWARD UNIFORM SPACES

- **Easier to Program & Compile for ...**

Summary, cont'd

REAL-TIME CACHE ARCHITECTURES:

- Really Big, Highly Associative Caches
- Software-Managed Caches
- Virtual Addressing w/ RT-TLB
- Real-Time SRAM Management

VIRTUAL MEMORY:

- Nice Programming Paradigm
- Separates NAMING from LOCATION
- Like Tang[®], Not Just for Breakfast ...

slides at <http://www.ece.umd.edu/~blj/talks/>

