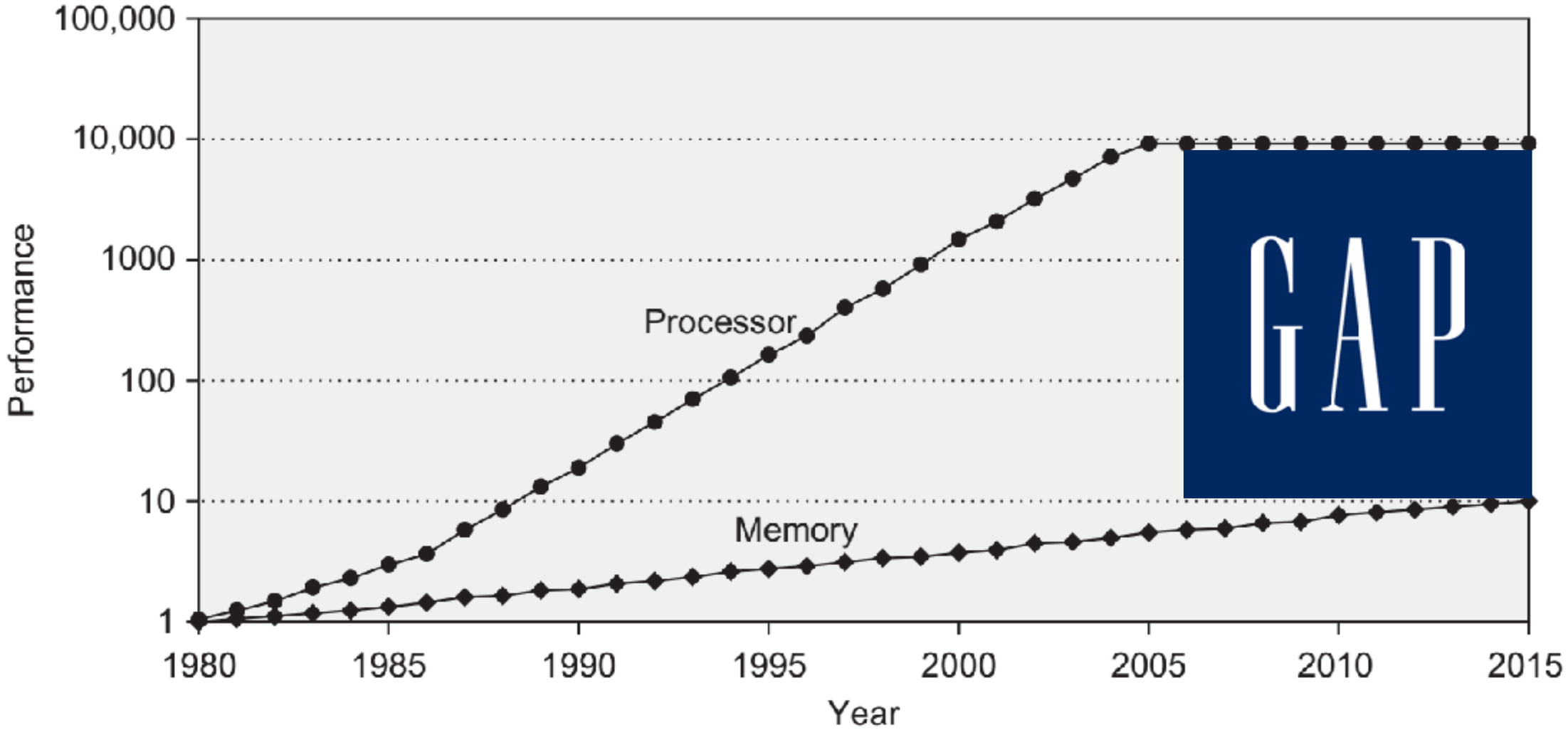
Optimizing cache performance — the soft way

Hung-Wei Tseng

Recap: Performance gap between Processor/Memory



Memory Hierarchy

fastest

< 1ns

a few ns

tens of ns

us/ms

Processor

Processor Core

Registers

SRAM $

DRAM

Storage

TBs

larger

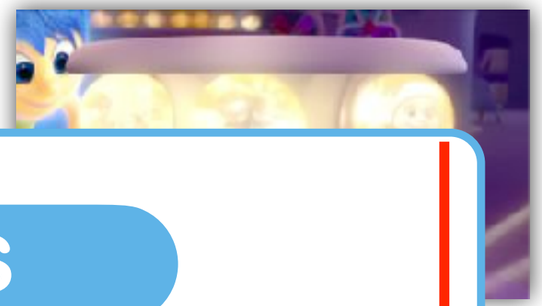
fastest

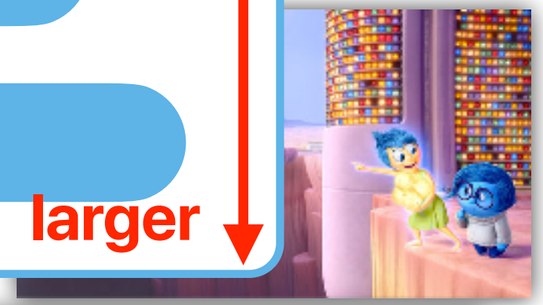
L1 $

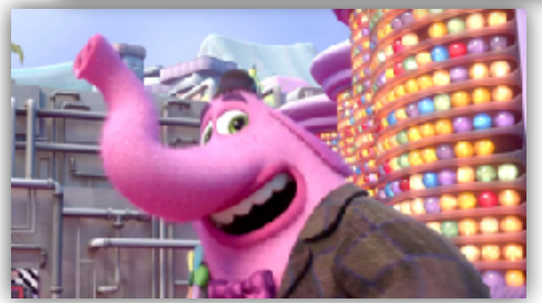
L2 $

L3 $

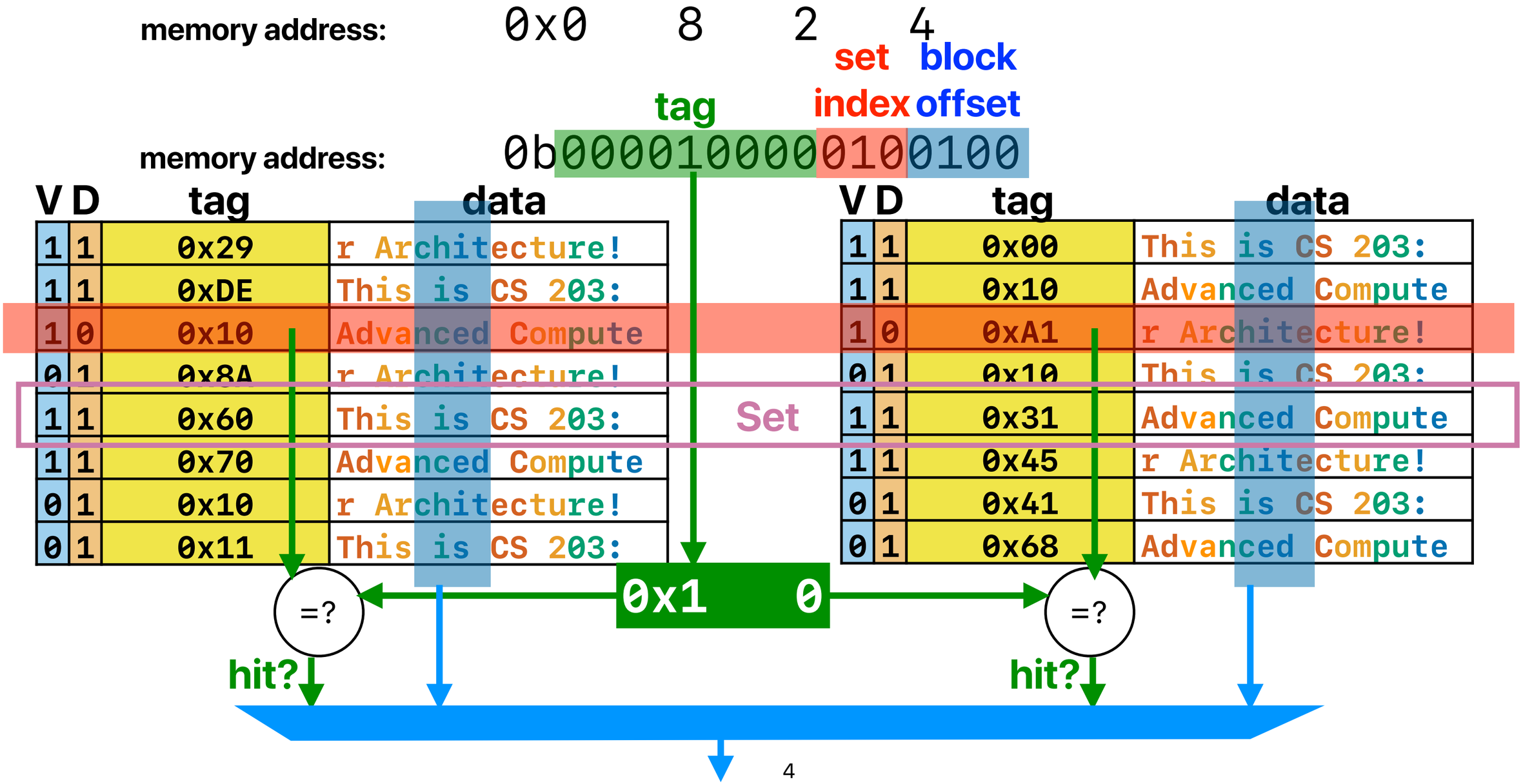
larger







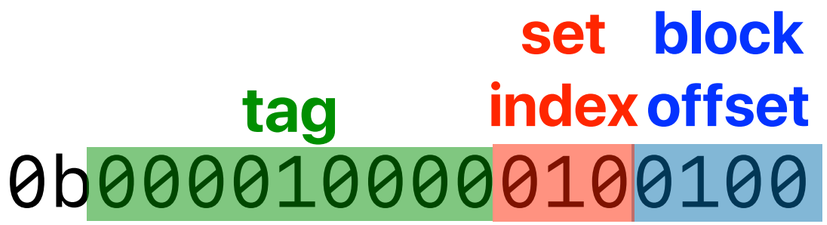
Way-associative cache



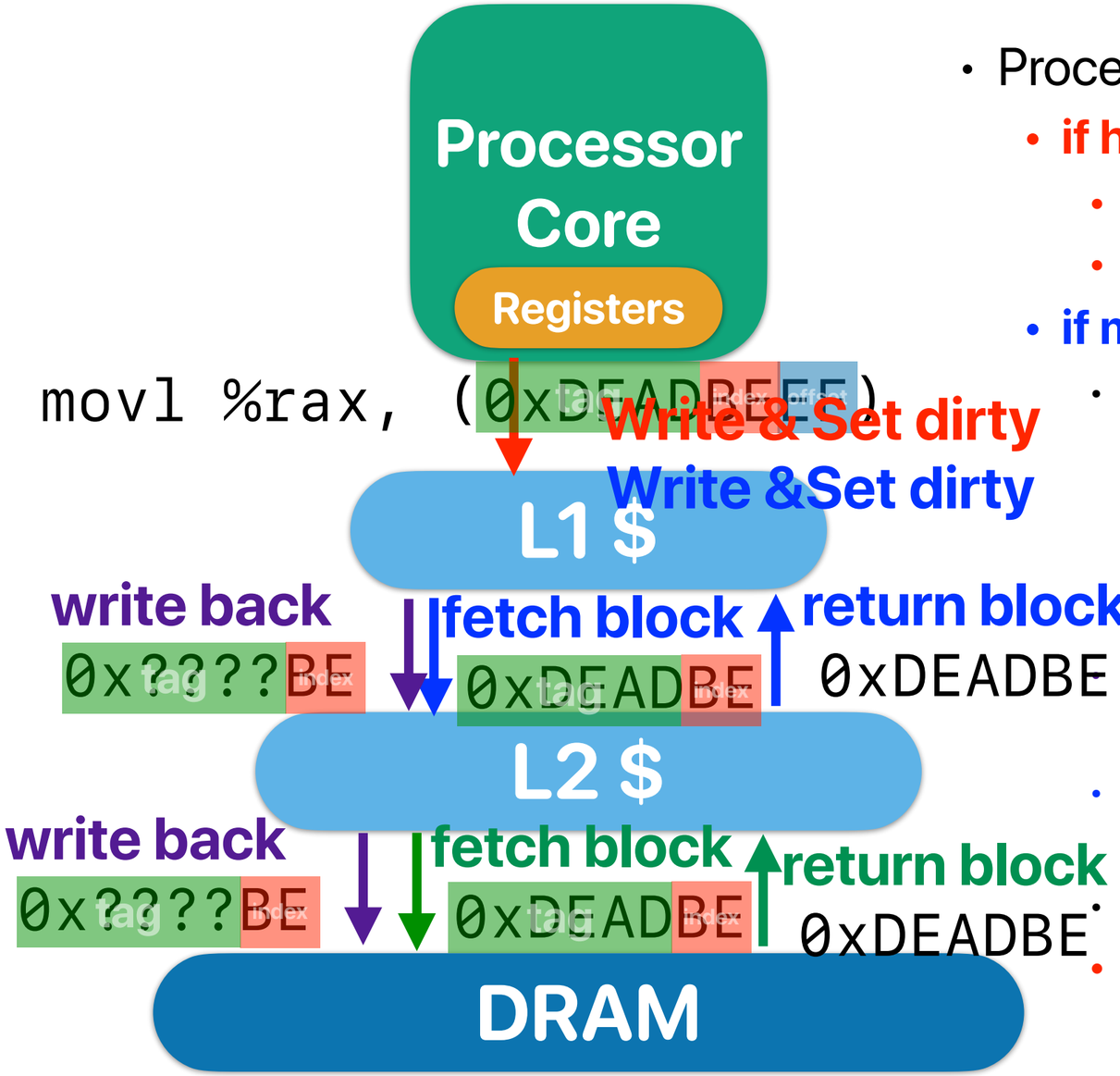
Review: C = ABS

- **C**: **C**apacity in data arrays
- **A**: Way-**A**ssociativity — how many blocks within a set
  - N-way: N blocks in a set, A = N
  - 1 for direct-mapped cache
- **B**: **B**lock Size (Cacheline)
  - How many bytes in a block
- **S**: Number of **S**ets:
  - A set contains blocks sharing the same index
  - 1 for fully associate cache
- number of bits in **b**lock offset — lg(**B**)
- number of bits in **s**et index: lg(**S**)
- tag bits: address_length - lg(S) - lg(B)
  - address_length is 64 bits for 64-bit machine
- $\frac{address}{block\_size} \pmod{S}$ = set index

memory address:



The complete picture



- Processor sends memory access request to L1-$
  - **if hit**
    - **Read - return data**
    - **Write - update & set DIRTY**
  - **if miss**
    - Select a victim block
      - If the target "set" is not full — select an empty/invalidated block as the victim block
      - If the target "set is full — select a victim block using some policy
      - LRU is preferred — to exploit temporal locality!
    - If the victim block is "dirty" & "valid"
      - **Write back** the block to lower-level memory hierarchy
    - Fetch the requesting block from lower-level memory hierarchy and place in the victim block
    - If write-back or fetching causes any miss, repeat the same process
  - **Present the write "ONLY" in L1 and set DIRTY**

Takeaways: 3Cs of misses

- Compulsory miss
  - Cold start miss. First-time access to a block
- Capacity miss
  - The working set size of an application is bigger than cache size
- Conflict miss
  - Required data replaced by block(s) mapping to the same set
  - Similar collision in hash

Takeaways: Hardware Optimizations

- Hardware
  - Increase way-associativity — conflict miss, but increase hit time
  - Increase block-size — compulsory miss, but increase miss penalty
  - Increase capacity — capacity miss, but increase hit time/cost
  - Prefetch — compulsory miss
  - Stream buffer — avoid the conflict misses caused prefetching
  - Victim cache, miss cache — conflict miss

Outline

- Optimizing cache performance — the software approach

How can programmer improve memory performance?

Data structures

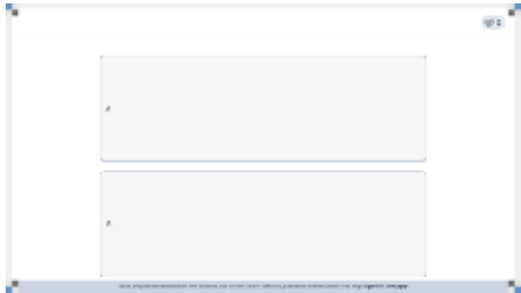
Column-store or row-store

- Considering your the most frequently used queries in your database system are similar to
  SELECT AVG(assignment_1) FROM table
  Which of the following would be a data structure that better implements the table supporting this type of queries?

Array of objects	object of arrays

`struct grades`
`{`
`  int id;`
`  double *homework;`
`  double average;`
`};`
`table = (struct grades *) \`
`malloc(num_of_students*sizeof(struct`
	`struct grades`
	`{`
	`  int *id;`
	`  double **homework;`
	`  double *average;`
	`};`
	
	`table =(struct grades *)malloc(sizeof(struct grades));`

A. Array of objects
B. Object of arrays



Column-store or row-store

- Considering your the most frequently used queries in your database system are similar to
  SELECT AVG(assignment_1) FROM table
  Which of the following would be a data structure that better implements the table supporting this type of queries?

Array of objects	object of arrays

`struct grades`
`{`
`  int id;`
`  double *homework;`
`  double average;`
`};`
`table = (struct grades *) \`
`malloc(num_of_students*sizeof(struct`
	`struct grades`
	`{`
	`  int *id;`
	`  double **homework;`
	`  double *average;`
	`};`
	
	`table =(struct grades *)malloc(sizeof(struct grades));`

A. Array of objects
B. Object of arrays

What if we want to calculate average scores for each student?

Array of structures or structure of arrays

Array of objects

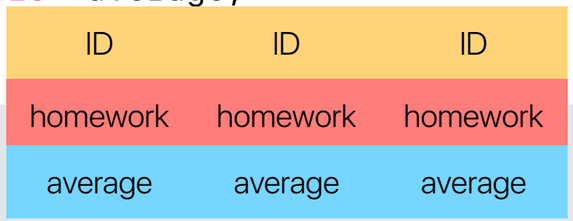
```
struct grades
{
  int id;
  double *homework;
  double average;
};
```

average of each homework

```
for(i=0;i<homework_items; i++)
{
gradesheet[total_number_students].homework[i] = 0.0;
   for(j=0;j<total_number_students;j++)
gradesheet[total_number_students].homework[i]
+=gradesheet[j].homework[i];
   gradesheet[total_number_students].homework[i] /=
(double)total_number_students;
}
```

object of arrays

```
struct grades
{
  int *id;
  double **homework;
  double *average;
};
```




```
for(i = 0;i < homework_items; i++)
{
  gradesheet.homework[i][total_number_students] = 0.0;
  for(j = 0; j <total_number_students;j++)
  {
      gradesheet.homework[i][total_number_students] +=
gradesheet.homework[i][j];
  }
      gradesheet.homework[i][total_number_students] /=
total_number_students;
}
```

Column-store or row-store

- If you're designing an in-memory database system, will you be using

RowId	EmpId	Lastname	Firstname	Salary
1	10	Smith	Joe	40000
2	12	Jones	Mary	50000
3	11	Johnson	Cathy	44000
4	22	Jones	Bob	55000

  - column-store — stores data tables column by column

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
40000:001,50000:002,44000:003,55000:004;
```

if the most frequently used query looks like —

```
select Lastname, Firstname from table
```

  - row-store — stores data tables row by row

```
001:10,Smith,Joe,40000;
002:12,Jones,Mary,50000;
003:11,Johnson,Cathy,44000;
004:22,Jones,Bob,55000;
```

Takeaways: Software Optimizations

- Data layout — capacity miss, conflict miss, compulsory miss

Loop interchange/fission/fusion

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

$O(n^2)$	Complexity	$O(n^2)$
Same	Instruction Count?	Same
Same	Clock Rate	Same
Better	CPI	Worse

Loop optimizations

Loop interchange

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

Takeaways: Software Optimizations

- Data layout — capacity miss, conflict miss, compulsory miss
- Loop interchange — conflict/capacity miss

NVIDIA Tegra X1

- D-L1 Cache configuration of NVIDIA Tegra X1
  - Size 32KB, 4-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384], b[16384], c[16384], d[16384], e[16384];
/* a = 0x10000, b = 0x20000, c = 0x30000, d = 0x40000, e = 0x50000 */
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
    //load a[i], b[i], c[i], d[i] and then store to e[i]
}
```

What's the data cache miss rate for this code?

A. 12.5%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

What if the code look like this?

- D-L1 Cache configuration of NVIDIA Tegra X1
  - Size 32KB, 4-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i]; //load a, b, c and then store to e
for(i = 0; i < 512; i++)
    e[i] /= d[i]; //load e, load d, and then store to e
```

What's the data cache miss rate for this code?

A. ~10%
B. ~20%
C. ~40%
D. ~80%
E. 100%

What if the code look like this?

- D-L1 Cache configuration of NVIDIA Tegra X1
  - Size 32KB, 4-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i]; //load a, b, c and then store to e
for(i = 0; i < 512; i++)
    e[i] /= d[i]; //load e, load d, and then store to e
```

What's the data cache miss rate for this code?

A. ~10%

B. ~20%

C. ~40%

D. ~80%

E. 100%

Loop optimizations

Loop interchange

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

Loop fission

B

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
}
```

A

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i];
for(i = 0; i < 512; i++)
    e[i] /= d[i];
```

Takeaways: Software Optimizations

- Data layout — capacity miss, conflict miss, compulsory miss
- Loop interchange — conflict/capacity miss
- Loop fission — conflict miss — when $ has limited way associativity

What if we change the processor?

- If we have an intel processor with a 48KB, 12-way, 64B-blocked L1 cache, which version of code performs better?
  A. Version A, because the code incurs fewer cache misses
  B. Version B, because the code incurs fewer cache misses
  C. Version A, because the code incurs fewer memory references
  D. Version B, because the code incurs fewer memory references
  E. They are about the same

A

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i];
for(i = 0; i < 512; i++)
    e[i] /= d[i];
```

B

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
}
```

What if we change the processor?

- If we have an intel processor with a 32KB, 8-way, 64B-blocked L1 cache, which version of code performs better?
  A. Version A, because the code incurs fewer cache misses
  B. Version B, because the code incurs fewer cache misses
  C. Version A, because the code incurs fewer memory references
  D. Version B, because the code incurs fewer memory references
  E. They are about the same

A

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i];
for(i = 0; i < 512; i++)
    e[i] /= d[i];
```

B

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
}
```

Loop optimizations

Loop interchange

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

Loop fission

B

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
}
```

A

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i];
for(i = 0; i < 512; i++)
    e[i] /= d[i];
```

Loop fusion

A

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i];
for(i = 0; i < 512; i++)
    e[i] /= d[i];
```

B

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
}
```

Takeaways: Software Optimizations

- Data layout — capacity miss, conflict miss, compulsory miss
- Loop interchange — conflict/capacity miss
- Loop fission — conflict miss — when $ has limited way associativity
- Loop fusion — capacity miss — when $ has enough way associativity

Tiling

Case study: Matrix Multiplication

```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Algorithm class tells you it's $O(n^3)$

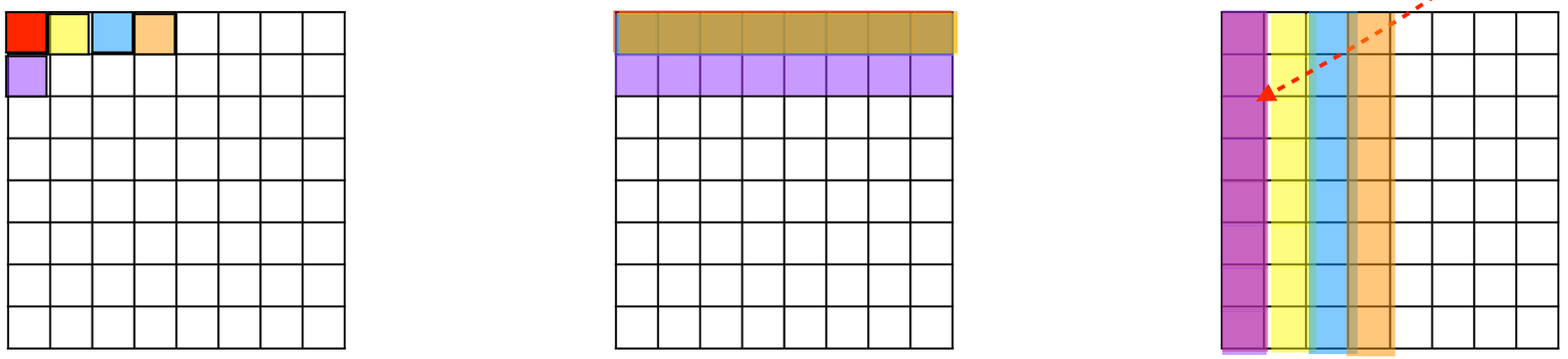
If n=1024, it takes about 1 sec

How long is it take when n=2048?

Matrix Multiplication

```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Very likely a miss if array is large



c a b

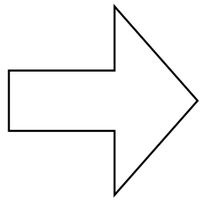
- If each dimension of your matrix is 2048
  - Each row takes 2048*8 bytes = 16KB
  - The L1 $ of intel Core i7 is 48 KB, 12-way, 64-byte blocked
  - You can only hold at most 3 rows/columns of each matrix!
  - You need the same row when j increase!

Tiling algorithm for matrix multiplication

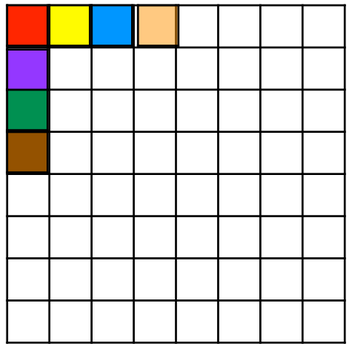
- Discover the cache miss rate
  - valgrind --tool=cachegrind cmd
    - cachegrind is a tool profiling the cache performance
  - Performance counter
    - Intel® Performance Counter Monitor http://www.intel.com/software/pcm/

Tiling algorithm for matrix multiplication

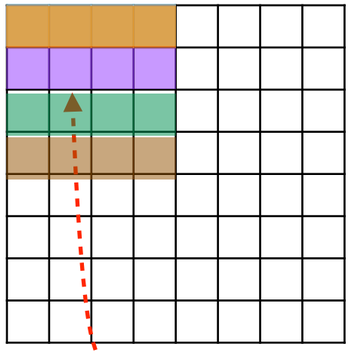
```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```



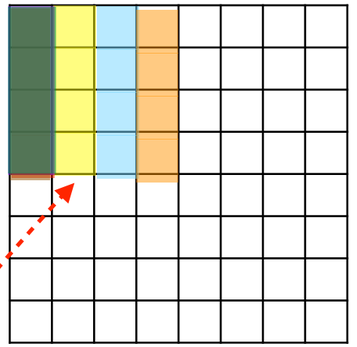
```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```



c



a



b

You only need to hold these sub-matrices in your cache

What kind(s) of misses can tiling algorithm remove?

- Comparing the naive algorithm and tiling algorithm on matrix multiplication, what kind of misses does tiling algorithm help to remove? (assuming an intel Core i7)

Naive

```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

What kind(s) of misses can tiling algorithm remove?

- Comparing the naive algorithm and tiling algorithm on matrix multiplication, what kind of misses does tiling algorithm help to remove? (assuming an intel Core i7)

Naive

```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

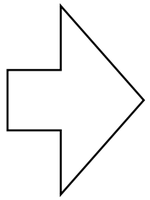
Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

Matrix Transpose

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```



```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}
```

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

https://www.pollev.com/hungweitseng close in 1:30

What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

Block + Transpose

```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}


for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

Block + Transpose

```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}


for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

Takeaways: Software Optimizations

- Data layout — capacity miss, conflict miss, compulsory miss
- Loop interchange — conflict/capacity miss
- Loop fission — conflict miss — when $ has limited way associativity
- Loop fusion — capacity miss — when $ has enough way associativity
- Blocking/tiling — capacity miss, conflict miss

Announcement

- Assignment #3 is released, due **this Thursday**
  - Please check our course website https://www.escalab.org/classes/cs203-2024sp/
  - Start early to get feedback from the autograder
- Midterm next Tuesday 9:30a-10:50a
  - Closed book, closed note
  - In-person only
  - Please bring a calculator. We don't allow calculators on any mobile device
  - Will release a sample midterm and review session this Thursday
- Hung-Wei's office hour this week is Wednesday 2p-4p
- Nurlan Nazaraliyev office hour this week is Friday 2p-4p
- CEN Talk this Friday at 12:30p — Zhi Guo from Fortinet
- Regarding autograder
  - Please be as detailed as possible — we do word count, we do want to find key words within your answers — you will need to do the same for midterm. Don't just show the final result
  - Please make sure you include the exact numbers you measured

UC RIVERSIDE Computer Engineering

CEN TECH TALK

Join Us to Learn How to Shape Tech Careers:
Insights and Guidance from Industry Experts

Zhi Guo

Director of ASIC Design at Fortinet

John Cortes

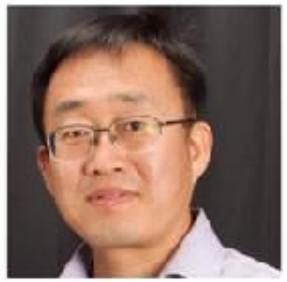
Manager of Software Development at Fortinet

Friday, May 3, 2024
12:30 pm - 1:30 pm

Winston Chung Hall
205/206

Food will be provided!

Guest speaker biography: Zhi Guo is a Director of ASIC Design at Fortinet, a global leader of cybersecurity solutions. Zhi got his Ph.D. from UCR supervised by Professor Walid Najjar. He received his B.E. and M.E. from Tsinghua University and Zhejiang University, respectively. At Fortinet, Zhi and his team work on the algorithms, architectures and verification systems of multiple generations of market-proven network security processors, SoCs and FPGAs. Prior to working at Fortinet, Zhi worked for Huawei Santa Clara R&D Center and Brocade Communications Systems. Zhi holds 10 granted and pending patents.



Guest speaker biography: John Cortes is a Manager of Software Development at Fortinet, a cybersecurity company headquartered in Sunnyvale. John received his MS in CS working under Walid Najjar. John received his BS in EE and CS from UCR as well. At Fortinet, John

Computer
Science &
Engineering

203

つづく