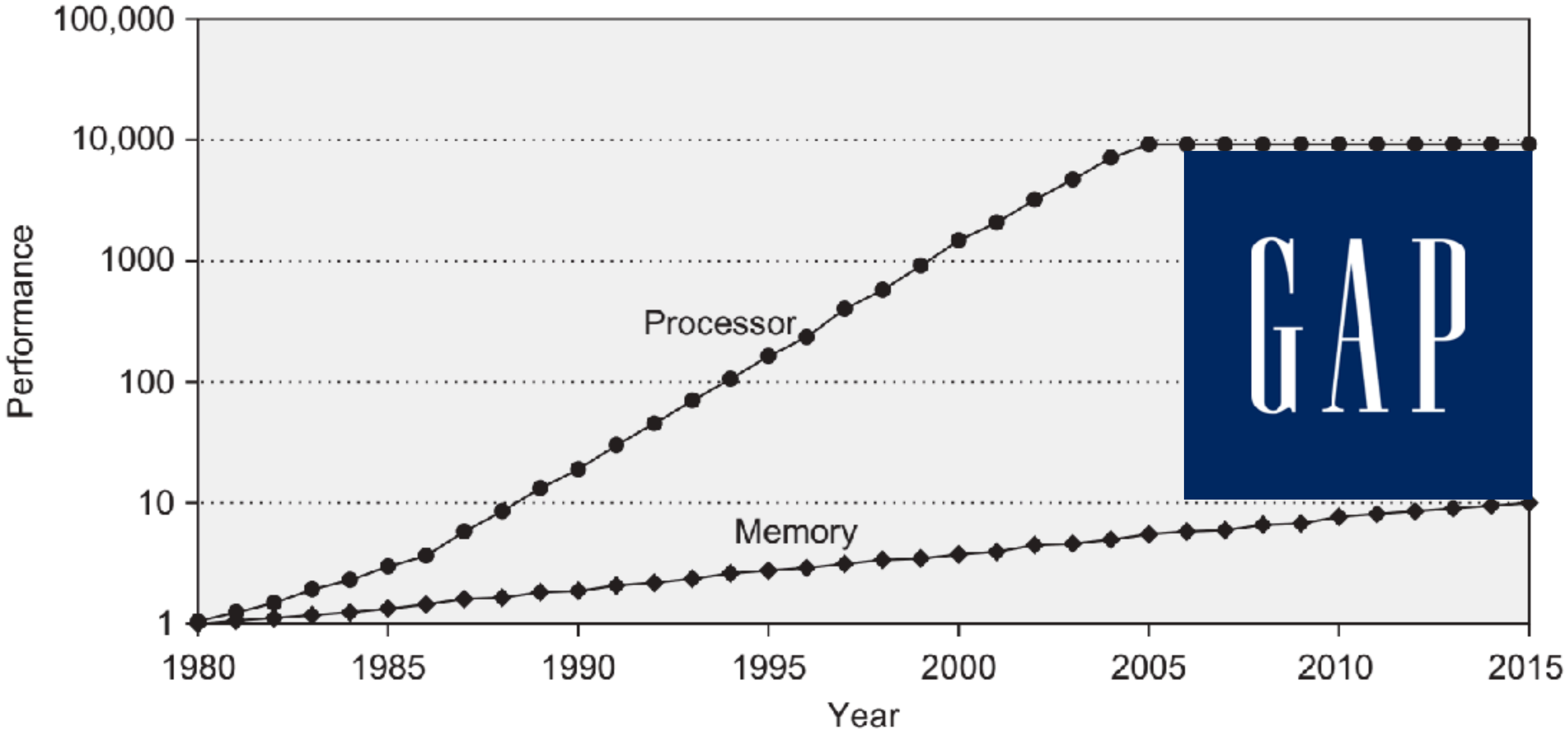
Memory Hierarchy (IV): Programming Techniques to Cache Performance & Basic Pipelined Processor Design

Hung-Wei Tseng

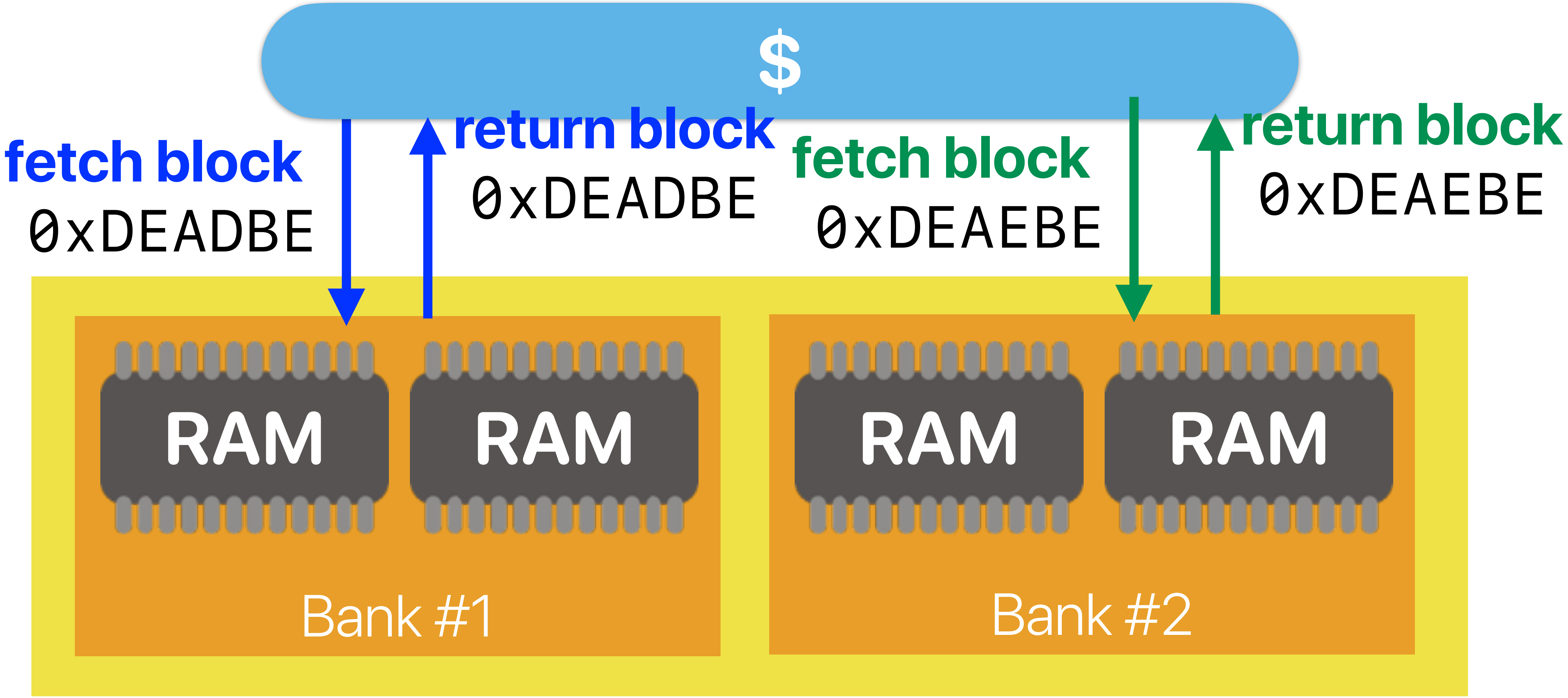
Performance gap between Processor/Memory



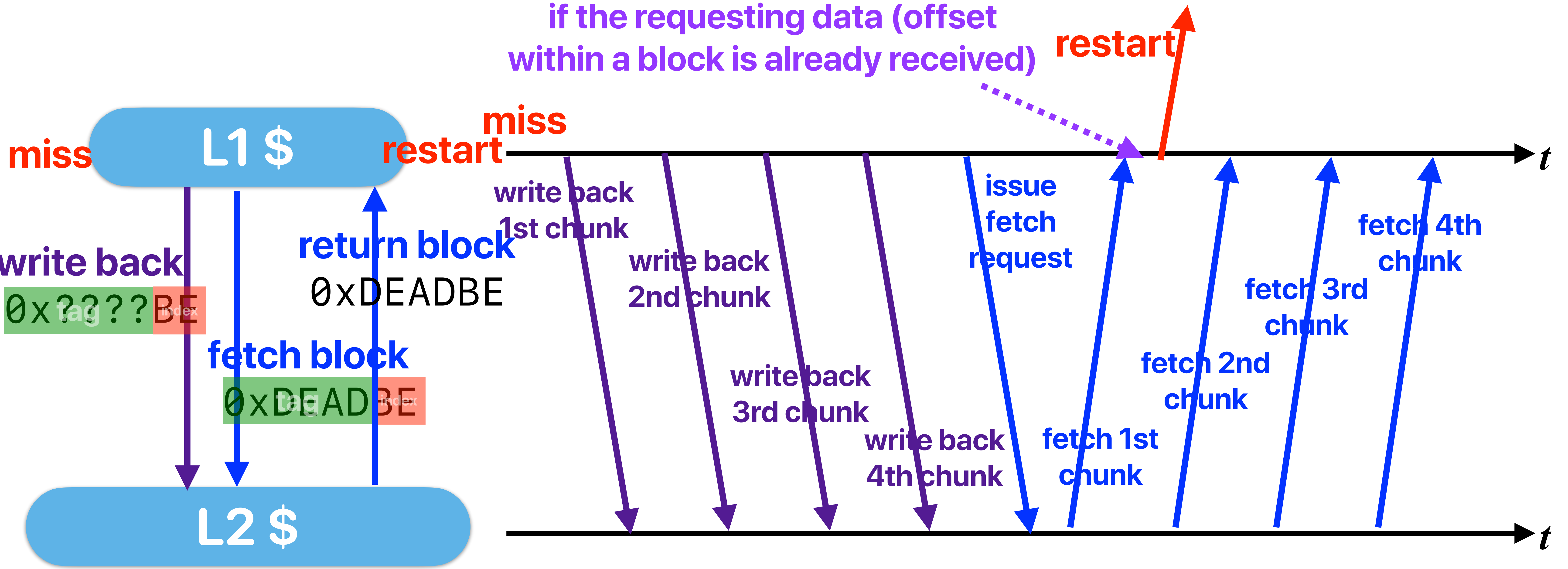
Which of the following schemes can help Athlon 64?

- How many of the following schemes mentioned in "improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers" would help AMD Phenom II for the code in the previous slide?
  ① Missing cache — help improving conflict misses
  ② Victim cache — help improving conflict misses
  ③ Prefetch — can potentially hurt
  ④ Stream buffer — only help improving compulsory misses
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Multibanks & non-blocking caches

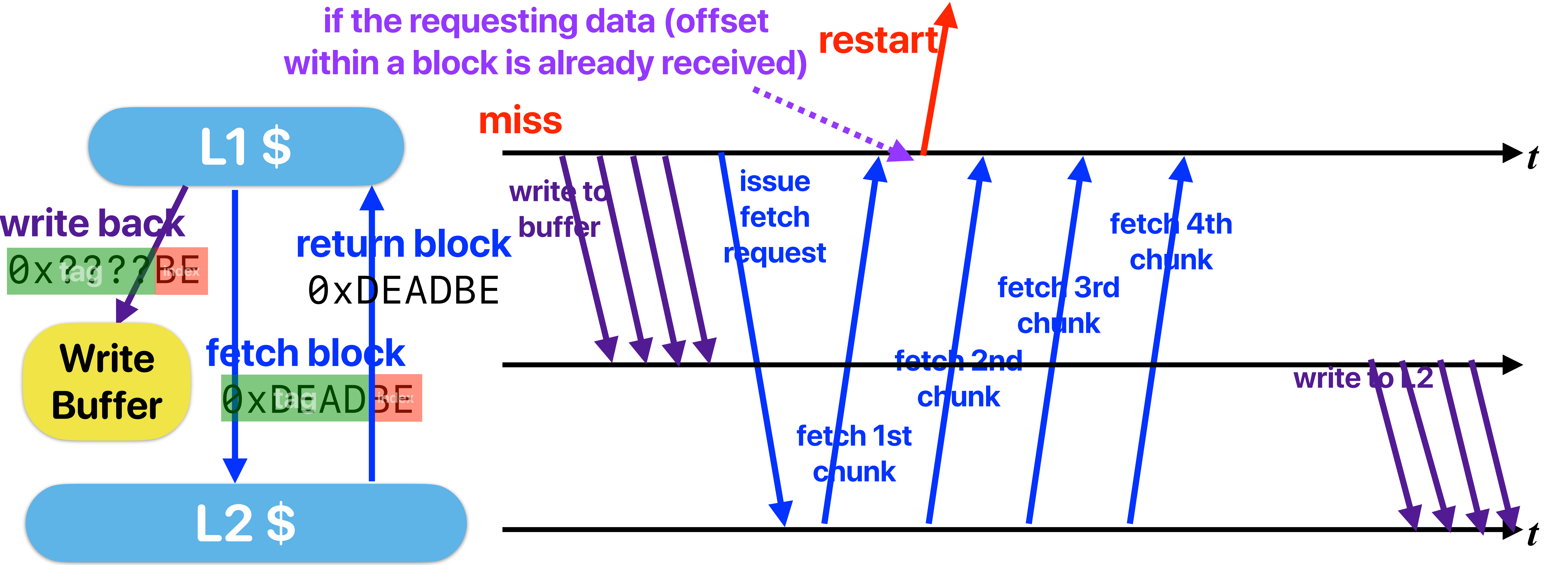


Early Restart and Critical Word First



assume the bus between L1/L2 only allows a quarter of the cache block go through it

Write buffer!



assume the bus between L1/L2 only allows a quarter of the cache block go through it

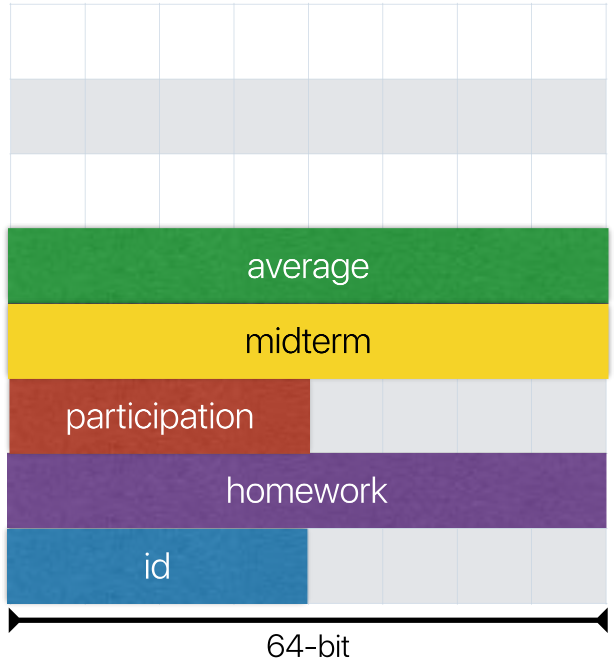
Summary of Optimizations

- Hardware
  - Prefetch — compulsory miss
  - Write buffer — miss penalty
  - Bank/pipeline — miss penalty
  - Critical word first and early restart — miss panelty

The result of `sizeof(struct student)`

- Consider the following data structure:

```
struct student {
    int id;
    double *homework;
    int participation;
    double midterm;
    double average;
};
```



What's the output of

`printf("%lu\n",sizeof(struct student))?`

A. 20
B. 28
C. 32
D. 36
E. 40

Memory addressing/alignment

- Almost every popular ISA architecture uses "byte-addressing" to access memory locations
- Instructions generally work faster when the given memory address is aligned
  - Aligned — if an instruction accesses an object of size n at address X, the access is **aligned** if X **mod** n **= 0**.
  - Some architecture/processor does not support aligned access at all
  - Therefore, compilers only allocate objects on "aligned" address
- **Compiler optimization cannot help!**

Team scores

4.5	5.5	4.5	4

Outline

- Programmer's optimizations for cache performance
- Basic Pipelined Processor Design

Programming and memory performance

Array of structures or structure of arrays

Array of objects

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

ID	*homework	average	ID	*homework	average

average of each homework

```c
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

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

ID	ID	ID
homework	homework	homework
average	average	average

average of each homework

```c
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;
}
```

Poll close in 1:30

What data structure is performing better

	Array of objects	object of arrays

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

- Considering your workload would like to calculate the average score of **one of the homework** for **all students**, which data structure would deliver better performance?
  - A. Array of objects
  - B. Object of arrays

Poll close in 1:30

1 — Instinct

What data structure is performing bet

<table>
<tr><th></th><th>Array of objects</th><th>object of arrays</th></tr>
<tr><td></td><td><pre>struct grades
{
  int id;
  double *homework;
  double average;
};</pre></td><td><pre>struct grades
{
  int *id;
  double **homework;
  double *average;
};</pre></td></tr>
<tr><td>average of each homework</td><td><pre>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;
}</pre></td><td><pre>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;
}</pre></td></tr>
</table>

- Considering your workload would like to calculate the average score of **one of the homework** for **all students**, which data structure would deliver better performance?
  A. Array of objects
  B. Object of arrays

What data structure is performing better

	Array of objects	object of arrays
	`struct grades { int id; double *homework; double average; };`	`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; }`	`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; }`

- Considering your workload would like to calculate the average score of **one of the homework** for **all students**, which data structure would deliver better performance? **What if we want to calculate average scores for each student?**
  A. Array of objects
  B. Object of arrays

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;
```

Loop interchange/fission/fusion

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];
  }
}
```

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

AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

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

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

A. 6.25%

B. 56.25%

C. 66.67%

D. 68.75%

E. 100%

C = ABS

64KB = 2 * 64 * S

S = 512

offset = lg(64) = 6 bits

index = lg(512) = 9 bits

tag = 64 - lg(512) - lg(64) = 49 bits

Poll close in 1:30

What if the code look like this?

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

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

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

A. 6.25%

B. 56.25%

C. 66.67%

D. 68.75%

E. 100%

Poll close in 1:30

What if the code look like this?

1 — Instinct

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

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

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

A. 6.25%

B. 56.25%

C. 66.67%

D. 68.75%

E. 100%

Loop fission

What if the code look like this?

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

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

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

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

Poll close in 1:30

What if the code look like this? — intel

A

```
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];
```

B

```
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
    {
        a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];
    }
```

- D-L1 Cache configuration of Intel Core i7
    - Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

Which version of code will perform better?

A. Version A

B. Version B

C. They're about the same

Poll close in 1:30

What if the code look like this? — int

1 — Instinct

A

```
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];
```

B

```
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
    {
        a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];
    }
```

- D-L1 Cache configuration of Intel Core i7
    - Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

Which version of code will perform better?

A. Version A

B. Version B

C. They're about the same

Loop Fusion

```
/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];


/* After */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
    {
        a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];
    }
```

2 misses per access to a & c vs. one miss per access

What if the code look like this? — intel

A

```
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];
```

B

```
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
    {
        a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];
    }
```

- D-L1 Cache configuration of Intel Core i7
  - Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

Which version of code will perform better?

A. Version A

B. Version B

C. They're about the same

Loop fusion

Blocking

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 $O(n^3)$

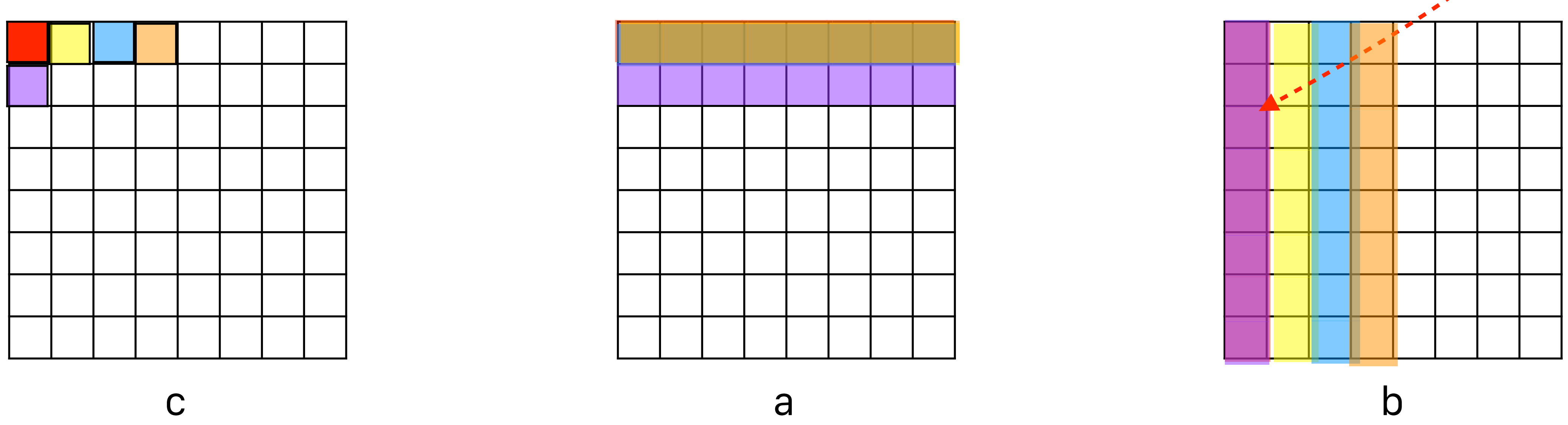
If n=1024, it takes about 1 sec

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

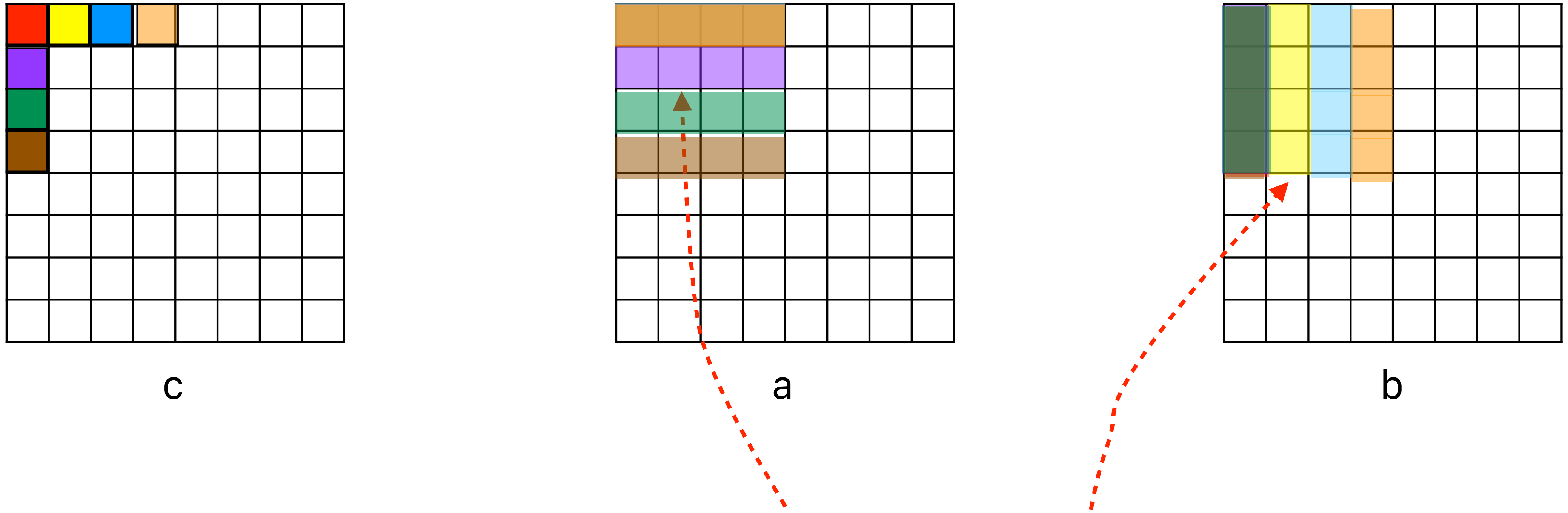


- If each dimension of your matrix is 2048
  - Each row takes 2048*8 bytes = 16KB
  - The L1 $ of intel Core i7 is 32KB, 8-way, 64-byte blocked
  - You can only hold at most 2 rows/columns of each matrix!
  - You need the same row when j increase!

Block 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];
    }
  }
}
```



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

How do you know it's better?

- 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/

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];
    }
  }
}
```

Poll 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];
    }
  }
}
```

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];
    }
  }
}
```

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

Poll close in 1:30

1 — Instinct

What kind(s) of misses can matrix transpose rem

- 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. $\boxed{\text{Conflict miss}}$

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];
    }
  }
}
```

Summary of Optimizations

- Software
  - Data layout — capacity miss, conflict miss, compulsory miss
  - Blocking — capacity miss
  - Transpose — conflict miss
  - Loop fission — conflict miss — when $ has limited way associativity
  - Loop fusion — capacity miss — when $ has enough way associativity
  - Loop interchange — conflict/capacity miss
- Hardware
  - Prefetch — compulsory miss
  - Write buffer — miss penalty
  - Bank/pipeline — miss penalty
  - Critical word first and early restart — miss panelty

Poll close in 1:30

Which version is faster?

A

```
for(i=0;i<1000000000;i++)
{
     sum+=data[(i*15) & 131071];
}
```

B

```
for(i=0;i<1000000000;i++)
{
     sum+=data[((i << 4) - i) & 131071];
}
```

- Both version A and B produces the same output. Without compiler optimization, which version of code would have better performance?
  A. Version A
  B. Version B
  C. They are about the same (less than 5% difference)

Poll close in 1:30

Which version is faster?

1 — Instinct

A

```
for(i=0;i<1000000000;i++)
{
    sum+=data[(i*15) & 131071];
}
```

B

```
for(i=0;i<1000000000;i++)
{
    sum+=data[((i << 4) - i) & 131071];
}
```

- Both version A and B produces the same output. Without compiler optimization, which version of code would have better performance?
  A. Version A
  B. Version B
  C. They are about the same (less than 5% difference)

Basic Pipelined Processor

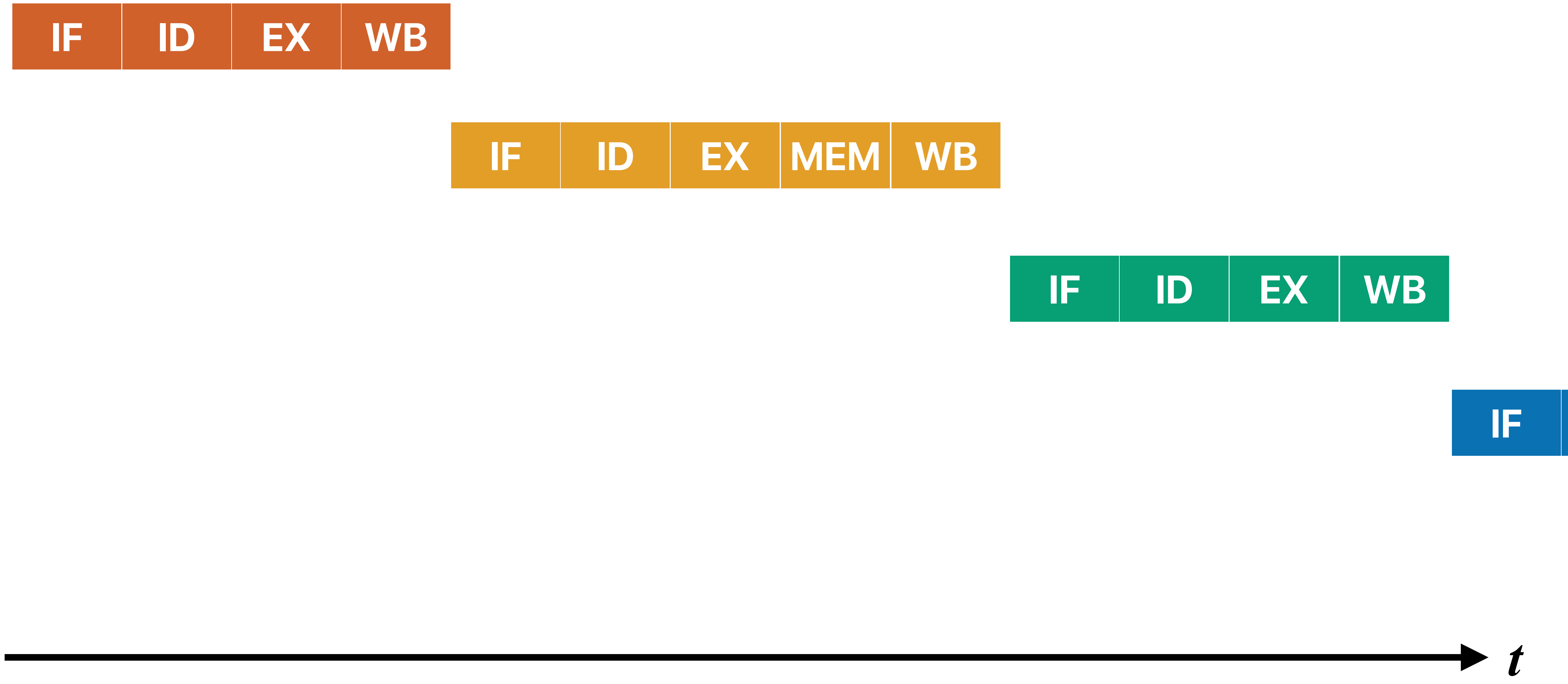
Hung-Wei Tseng

Tasks in RISC-V ISA

- Instruction Fetch (**IF**) — fetch the instruction from memory
- Instruction Decode (**ID**)
  - Decode the instruction for the desired operation and operands
  - Reading source register values
- Execution (**EX**)
  - ALU instructions: Perform ALU operations
  - Conditional Branch: Determine the branch outcome (taken/not taken)
  - Memory instructions: Determine the effective address for data memory access
- Data Memory Access (**MEM**) — Read/write memory
- Write Back (**WB**) — Present ALU result/read value in the target register
- Update PC
  - If the branch is taken — set to the branch target address
  - Otherwise — advance to the next instruction — current PC + 4

Simple implementation w/o branch

```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9,x10,x11
sd  x1, 0(x12)
```



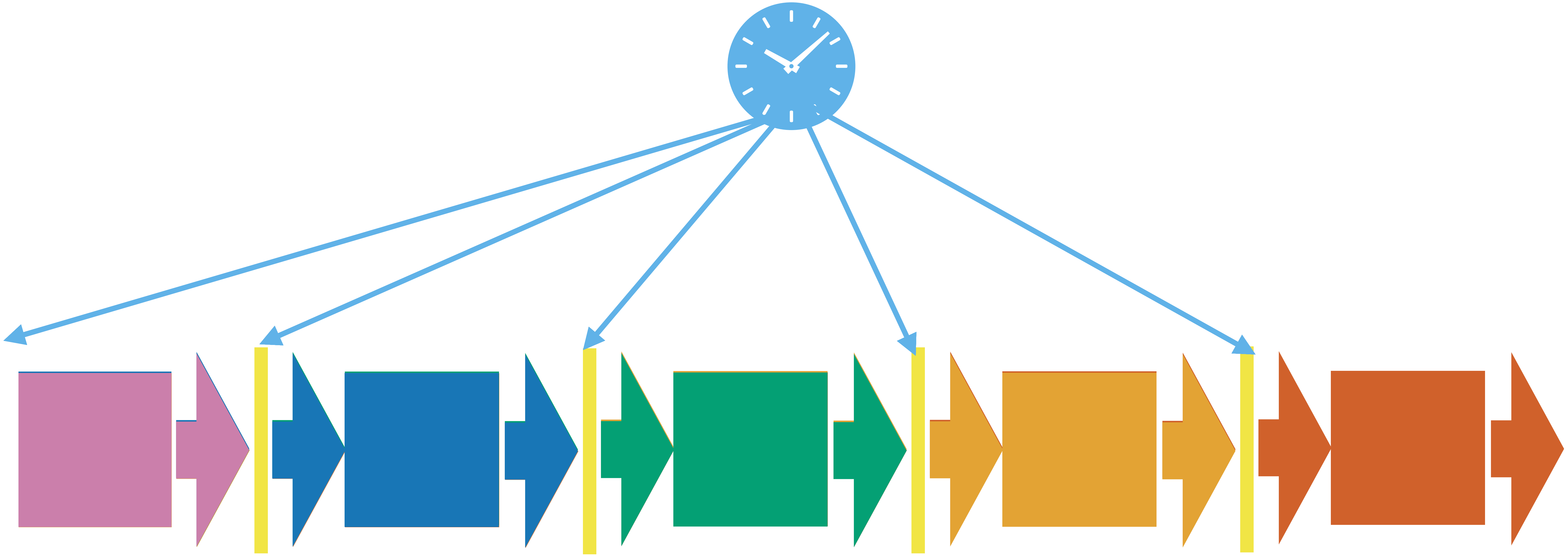
Pipelining

Pipelining

Pipelining

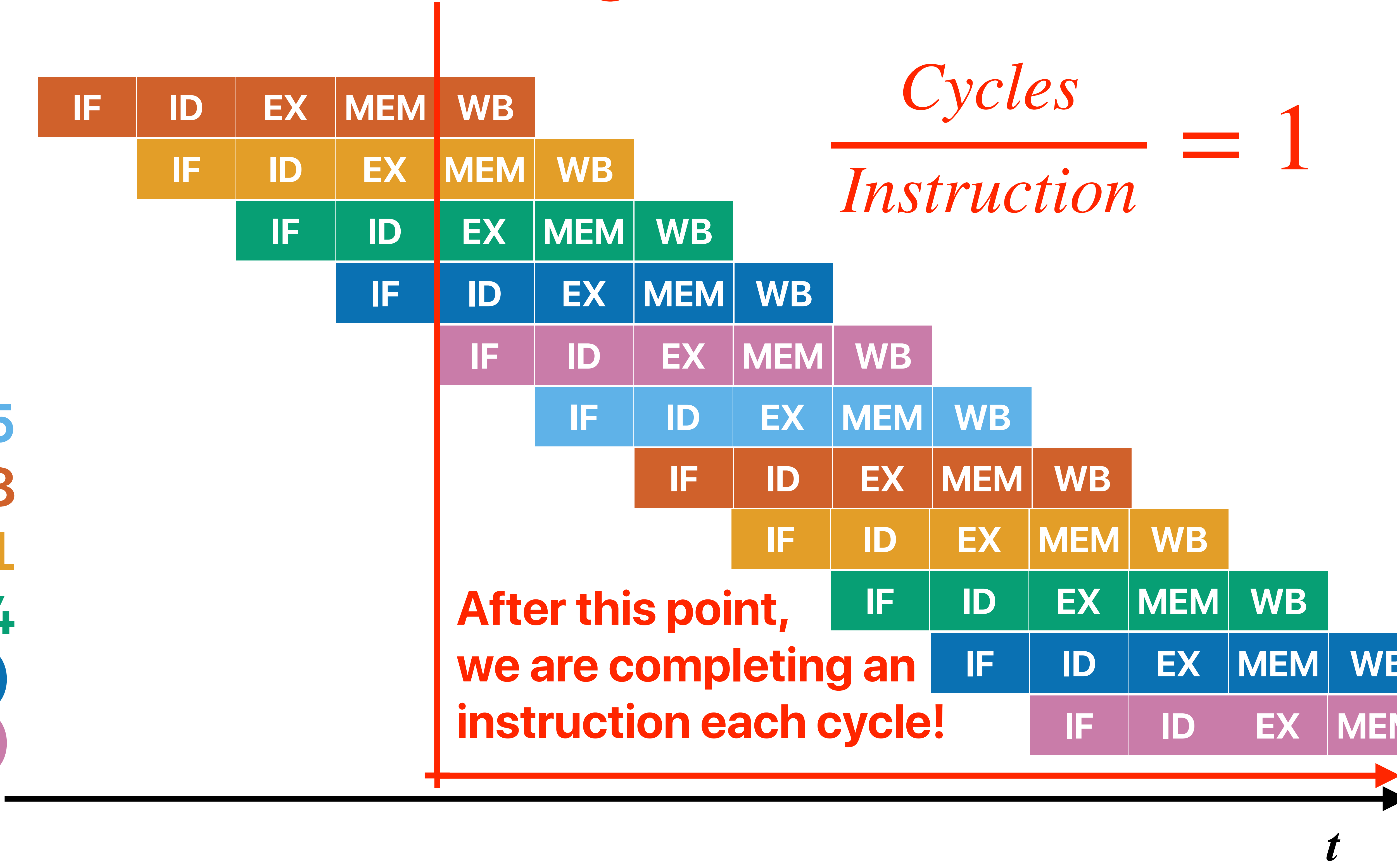
- Different parts of the processor works on different instructions simultaneously
- A clock signal controls and synchronize the beginning and the end of each part of the work
- A pipeline register between different parts of the processor to keep intermediate results necessary for the upcoming work

Pipelining



Pipelining

```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9,x10,x11
sd  x1, 0(x12)
xor x13,x14,x15
and x16,x17,x18
add x19,x20,x21
sub x22,x23,x24
ld  x25, 4(x26)
sd  x27, 0(x28)
```



$$\frac{Cycles}{Instruction} = 1$$

Which version is faster?

A

```
for(i=0;i<1000000000;i++)
{
    sum+=data[(i*15) & 131071];
}
```

B

```
for(i=0;i<1000000000;i++)
{
    sum+=data[((i << 4) - i) & 131071];
}
```

- Both version A and B produces the same output. Without compiler optimization, which version of code would have significantly better performance?
  A. Version A
  B. Version B
  C. They are about the same (less than 10% difference)

  — Because we have pipelined instructions, the CPI of one instruction doesn't matter as long as we can keep the pipeline busy

Announcement

- Project is up — check the website
- Assignment #3 due next Monday
- Midterm
  - Release next Tuesday (11/10) 12:00am, turn in before next Friday (11/13) 11:59pm
  - You can only open it once and you have to finish a total of 30 questions within 80 minutes.
  - You may open book, but you have to bare the risks of not being able to finish them
- Attendance
  - The attendance throughout the quarter count as one assignment
  - You only need to answer 50% of the Zoom polls to receive full credits
    - Please don't email me for absence — we count only 50% to give you flexibility
    - If you just login but never answer questions, you won't receive any.
- Reading Quizzes — 2 attempts, average
- Office Hours on Zoom (the office hour link, not the lecture one)
  - Hung-Wei/Prof. Usagi: M 8p-10p (make up for the last week), W 2p-3p
  - Quan Fan: F 1p-3p

Computer
Science &
Engineering

つづく