Advanced Programming

Talk 5: Memory Hierarchy

Saumitra Joshi

Outline

- Concept of Memory
- Memory Hierarchy
- Memory Organization
    - Stack
    - Heap

Concept of Memory

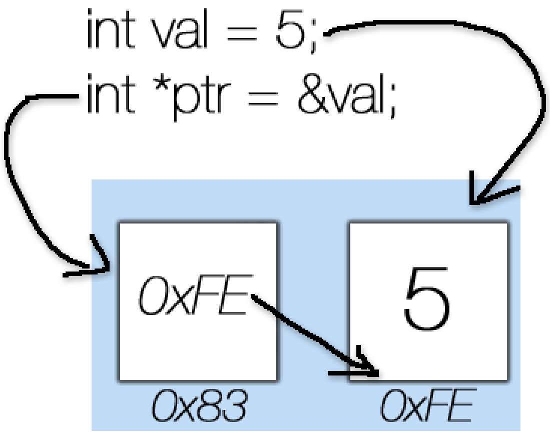
- Computer’s way of “remembering”
  - Instructions: “What to do?”
  - Data: “Whom to work on?”

Concept of Memory

- Computer’s way of “remembering”
  - Instructions: “What to do?”
  - Data: “Whom to work on?”
- Names?
  - `0x1f33`, `0x332d`, `0x4a3e` .. :/

Concept of Memory

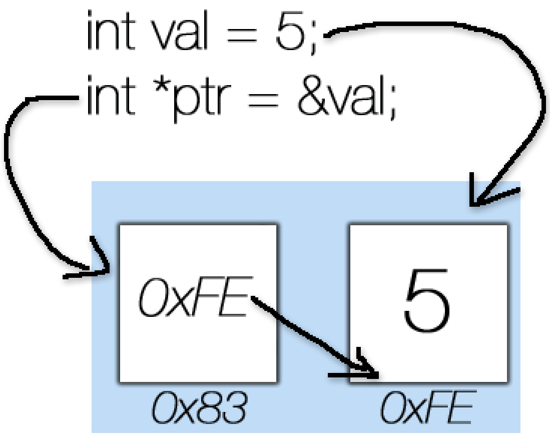
- Computer's way of "remembering"
  - Instructions: "What to do?"
  - Data: "Whom to work on?"
- Names?
  - `0x1f33`, `0x332d`, `0x4a3e` .. :/
  - Languages — map variables to memory
  - Use references to access data



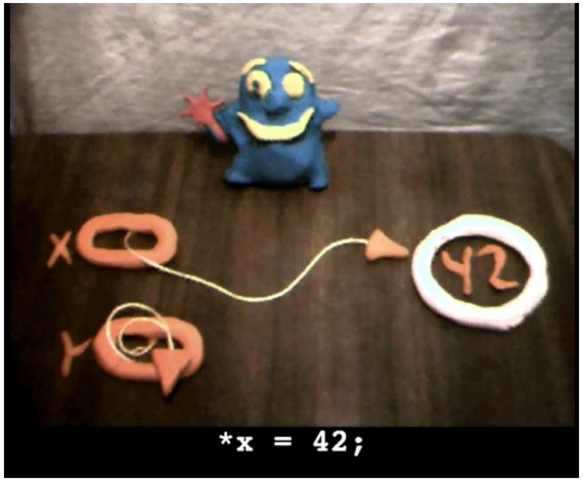
```
int val = 5;
int *ptr = &val;
```

Concept of Memory

- Computer's way of "remembering"
  - Instructions: "What to do?"
  - Data: "Whom to work on?"
- Names?
  - `0x1f33`, `0x332d`, `0x4a3e` .. :/
  - Languages — map variables to memory
  - Use references to access data
- Questions on pointers? [Ask Binky](#)!



```
int val = 5;
int *ptr = &val;
```



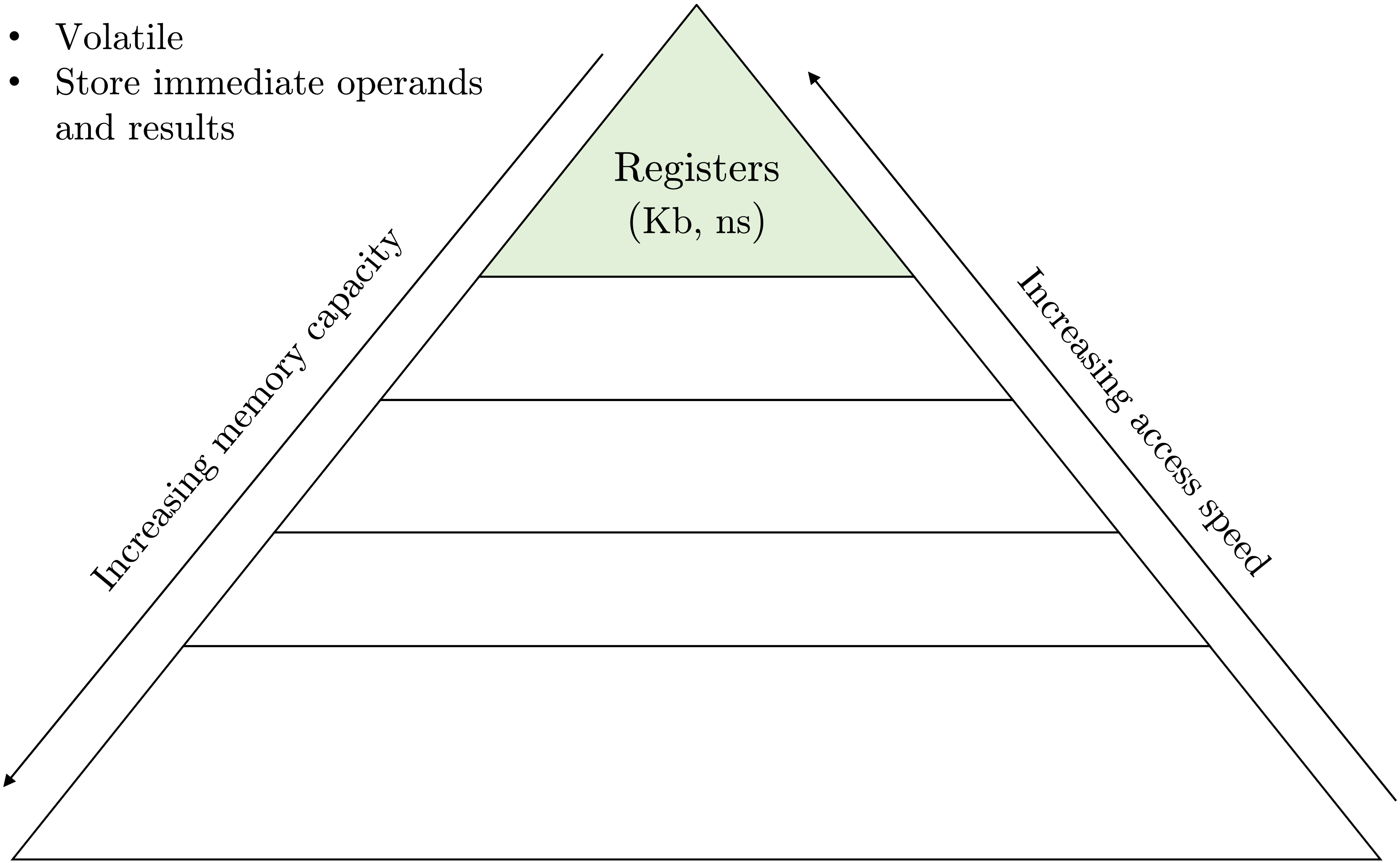
```
*x = 42;
```

Memory Hierarchy

- Categorized based on:
  - Size
  - Latency

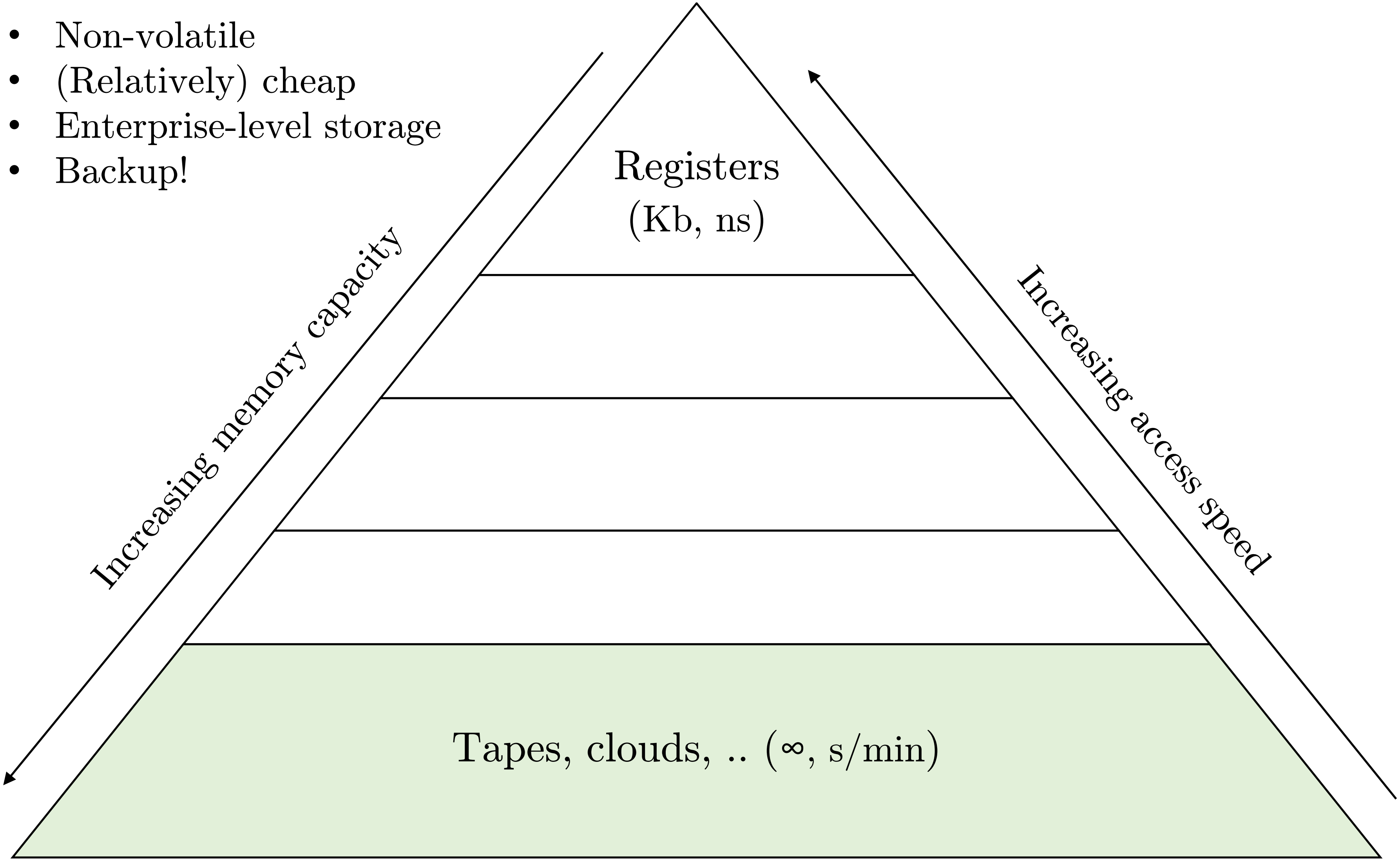
Memory Hierarchy: Registers

- Volatile
- Store immediate operands and results



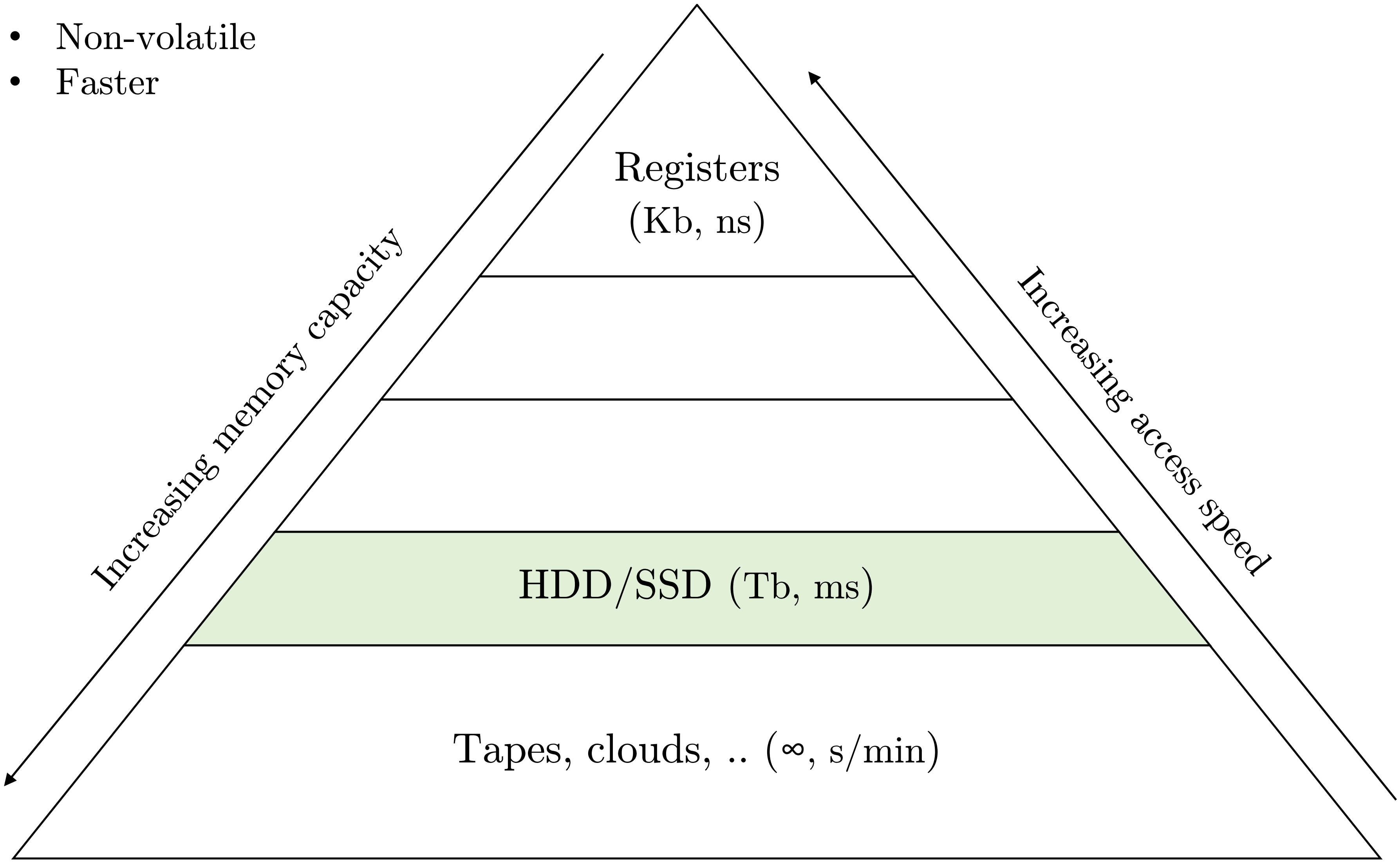
Memory Hierarchy: Tertiary Storage

- Non-volatile
- (Relatively) cheap
- Enterprise-level storage
- Backup!



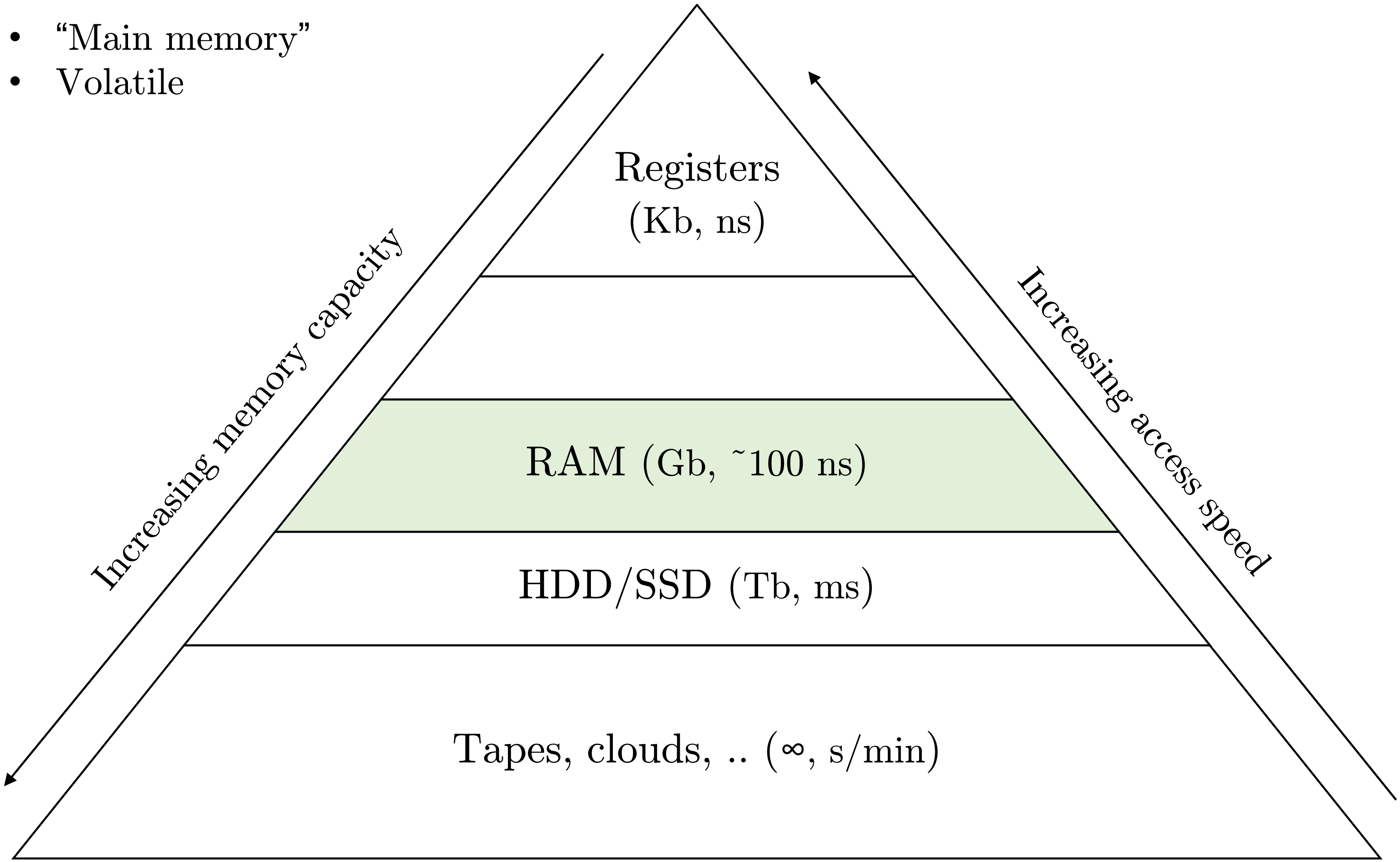
Memory Hierarchy: Secondary Storage

- Non-volatile
- Faster



Memory Hierarchy: RAM

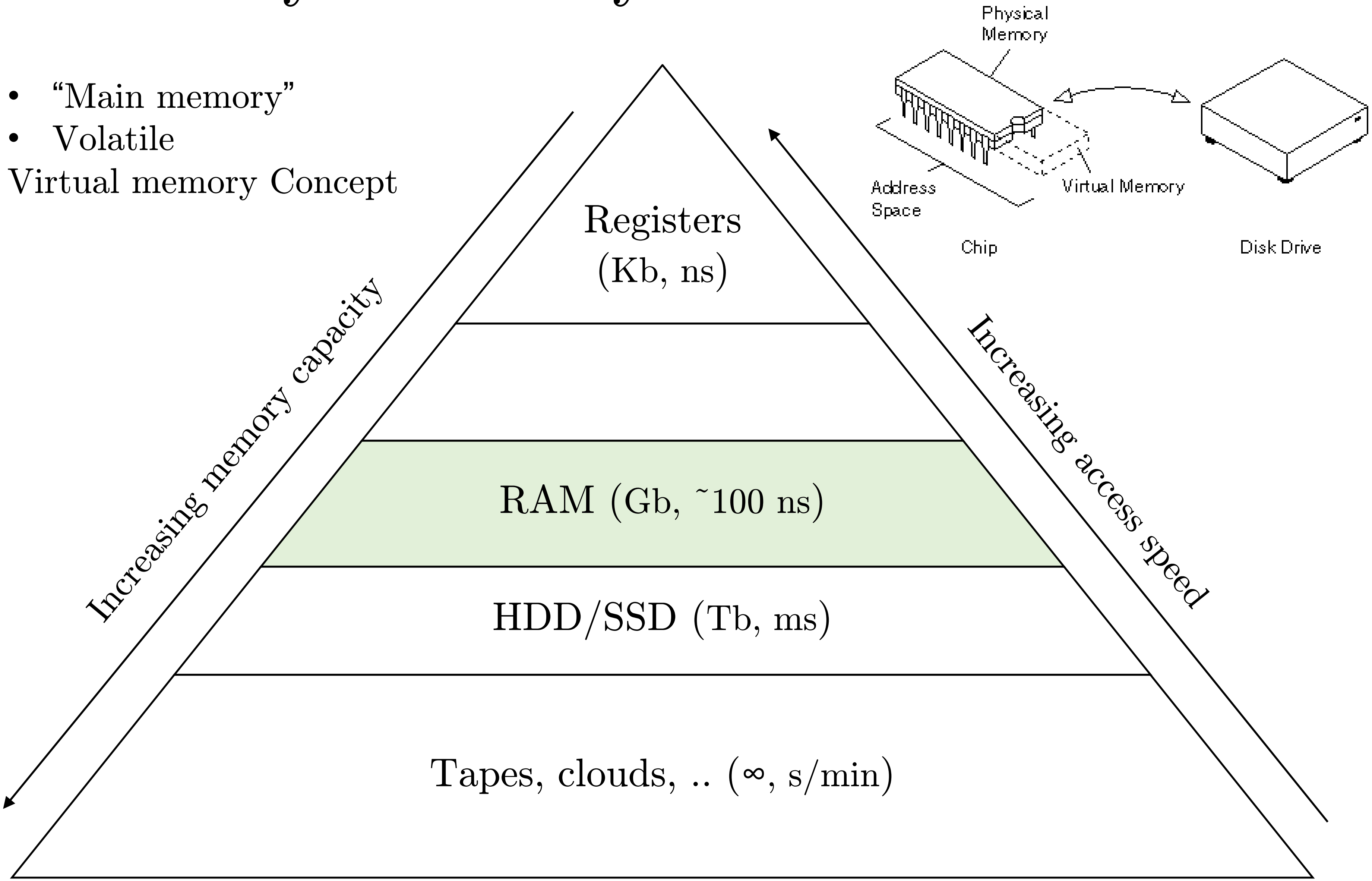
- "Main memory"
- Volatile



Memory Hierarchy: RAM

- “Main memory”
- Volatile

Virtual memory Concept



Registers (Kb, ns)

RAM (Gb, ~100 ns)

HDD/SSD (Tb, ms)

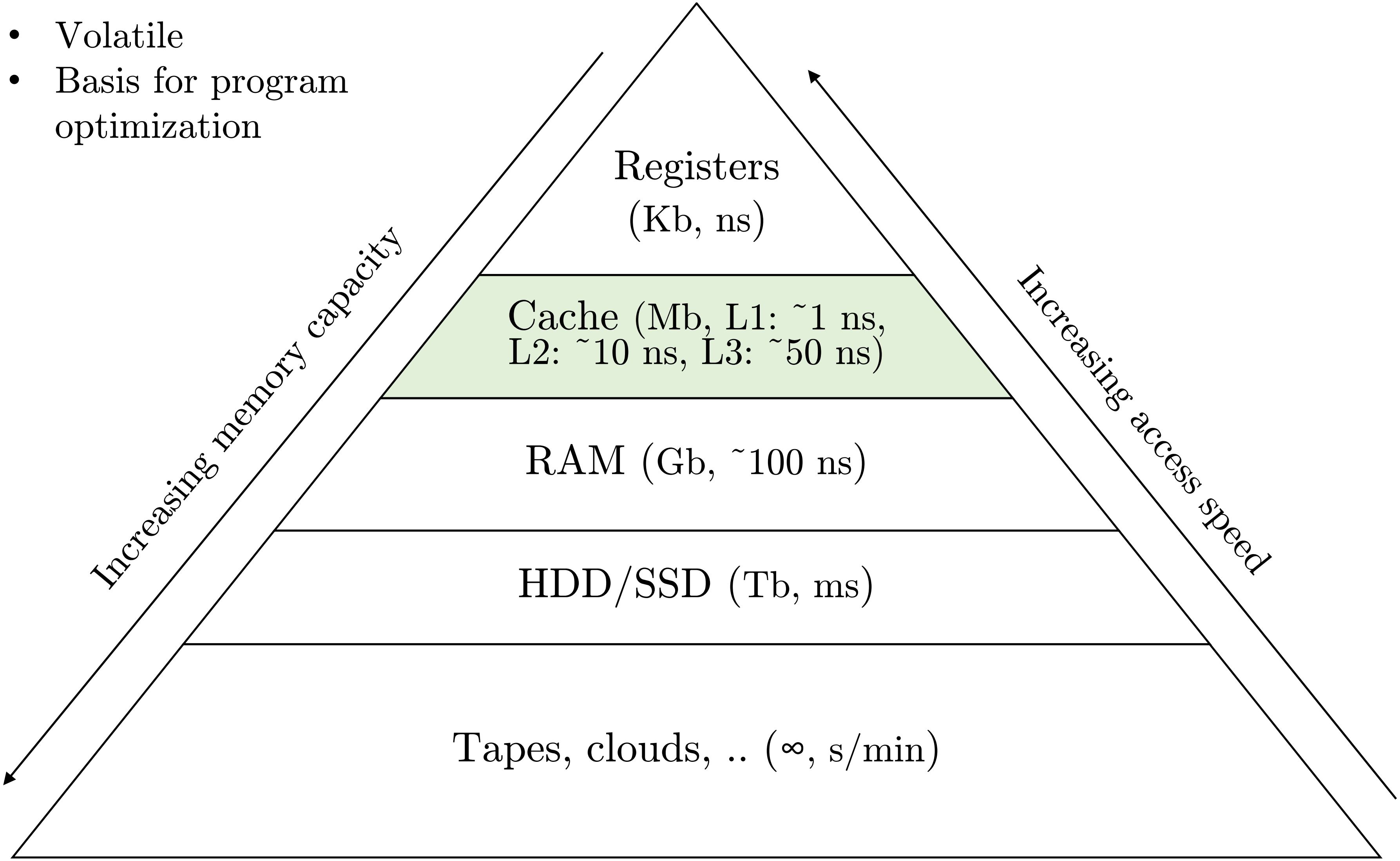
Tapes, clouds, .. (∞, s/min)

Increasing memory capacity

Increasing access speed

Memory Hierarchy: Cache

- Volatile
- Basis for program optimization



Registers (Kb, ns)

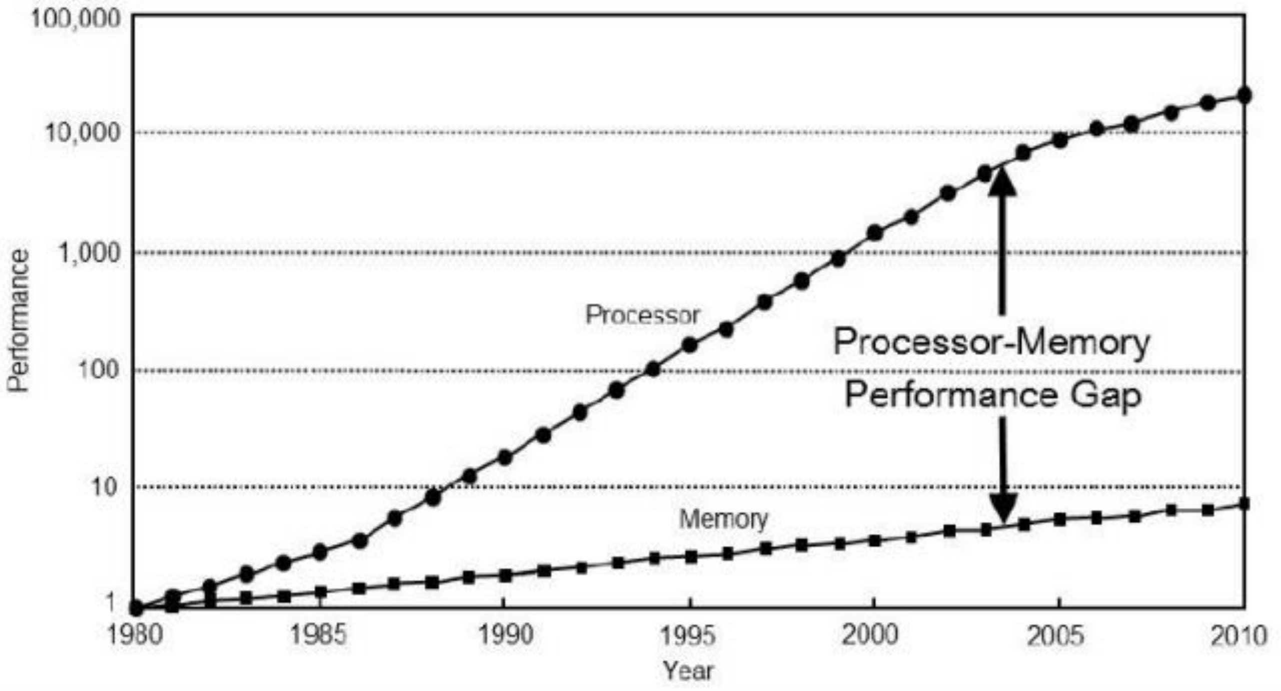
Cache (Mb, L1: ~1 ns, L2: ~10 ns, L3: ~50 ns)

RAM (Gb, ~100 ns)

HDD/SSD (Tb, ms)

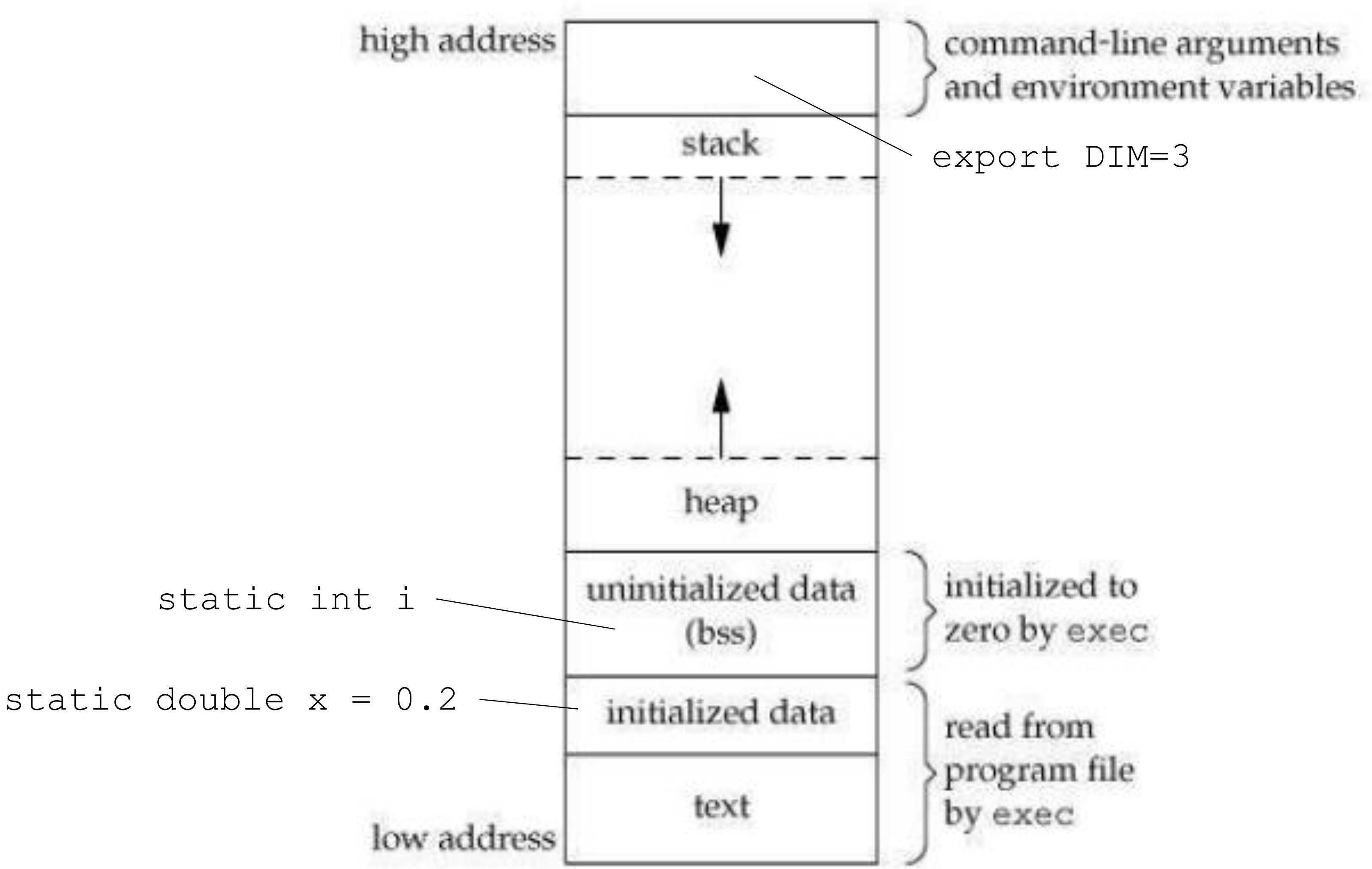
Tapes, clouds, .. (∞, s/min)

Memory Performance Gap



Source: Hennessy, Patterson, Arpaci-Dusseau: Computer Architecture: A Quantitative Approach

Organization of Memory


high address
command-line arguments and environment variables
stack
heap
uninitialized data (bss)
initialized to zero by exec
initialized data
text
read from program file by exec
low address

`export DIM=3`

`static int i`

`static double x = 0.2`

Stack

```
double x = 42.0;
int *p = &q;
```

- LIFO
  - Fast memory access
- Automatic de-allocation after end of scope

- Stack-overflow may cause undefined behavior
- No security against out-of-bounds access!

Stack: Example

```
int bump(int num) {
    num += 1;
    return num;
}

int main() {
    int a = 2;
    int b = bump(a);
    return 0;
}
```

0x15		
0x16		
0x17		
0x18		
0x19		
0x20		
0x21		
0x22		

Stack: Example

```
int bump(int num) {
     num += 1;
     return num;
}

int main() {
     int a = 2;
     int b = bump(a);
     return 0;
}
```

0x15	a	2
0x16		
0x17		
0x18		
0x19		
0x20		
0x21		
0x22		

Stack: Example

```
int bump(int num) {
    num += 1;
    return num;
}

int main() {
    int a = 2;
    int b = bump(a);
    return 0;
}
```

0x15	a	2
0x16		
0x17		
0x18		
0x19		
0x20		
0x21		
0x22		

Stack: Example

```
int bump(int num) {
    num += 1;
    return num;
}

int main() {
    int a = 2;
    int b = bump(a);
    return 0;
}
```

0x15	a	2
0x16	num	2
0x17		
0x18		
0x19		
0x20		
0x21		
0x22		

Stack: Example

```
int bump(int num) {
    num += 1;
    return num;
}

int main() {
    int a = 2;
    int b = bump(a);
    return 0;
}
```

0x15	a	2
0x16	num	3
0x17		
0x18		
0x19		
0x20		
0x21		
0x22		

Stack: Example

```
int bump(int num) {
    num += 1;
    return num;
}

int main() {
    int a = 2;
    int b = bump(a);
    return 0;
}
```

0x15	a	2
0x16		
0x17		
0x18		
0x19		
0x20		
0x21		
0x22		

Stack: Example

```
int bump(int num) {
    num += 1;
    return num;
}

int main() {
    int a = 2;
    int b = bump(a);
    return 0;
}
```

0x15	a	2
0x16	b	3
0x17		
0x18		
0x19		
0x20		
0x21		
0x22		

Stack: Example

```
int bump(int num) {
    num += 1;
    return num;
}

int main() {
    int a = 2;
    int b = bump(a);
    return 0;
}
```

0x15		
0x16		
0x17		
0x18		
0x19		
0x20		
0x21		
0x22		

Heap

```
double *x = new double;  // C++
int *p = malloc int[10]; // C
```

- Linked list of used and free blocks
- On-demand memory allocation
- Size can grow during runtime

- Slower allocation
- Memory not freed automatically!
  - Memory leakage

Heap: Example

```
int bump(int num) {
      int *ans = new int;
      *ans = num + 1;
      return *ans;
}

int main() {
      int *a = new int;
      *a = 2;
      int b = bump(*a);
      return 0;
}
```

0x15		
0x16		
0x17		

0x22		
0x23		
0x24		
0x25		
0x26		

Heap: Example

```
int bump(int num) {
      int *ans = new int;
      *ans = num + 1;
      return *ans;
}

int main() {
      int *a = new int;
      *a = 2;
      int b = bump(*a);
      return 0;
}
```

0x15	a	0x23
0x16		
0x17		

0x22		
0x23	*a	-
0x24		
0x25		
0x26		

Heap: Example

```
int bump(int num) {
    int *ans = new int;
    *ans = num + 1;
    return *ans;
}

int main() {
    int *a = new int;
    *a = 2;
    int b = bump(*a);
    return 0;
}
```

0x15	a	0x23
0x16		
0x17		

0x22		
0x23	*a	2
0x24		
0x25		
0x26		

Heap: Example

```
int bump(int num) {
      int *ans = new int;
      *ans = num + 1;
      return *ans;
}

int main() {
      int *a = new int;
      *a = 2;
      int b = bump(*a);
      return 0;
}
```

0x15	a	0x23
0x16		
0x17		

0x22		
0x23	*a	2
0x24		
0x25		
0x26		

Heap: Example

```
int bump(int num) {
    int *ans = new int;
    *ans = num + 1;
    return *ans;
}

int main() {
    int *a = new int;
    *a = 2;
    int b = bump(*a);
    return 0;
}
```

0x15	a	0x23
0x16	num	2
0x17		

0x22		
0x23	*a	2
0x24		
0x25		
0x26		

Heap: Example

```
int bump(int num) {
      int *ans = new int;
      *ans = num + 1;
      return *ans;
}

int main() {
      int *a = new int;
      *a = 2;
      int b = bump(*a);
      return 0;
}
```

0x15	a	0x23
0x16	num	2
0x17	ans	0x26

0x22		
0x23	*a	2
0x24		
0x25		
0x26	*ans	-

Heap: Example

```
int bump(int num) {
    int *ans = new int;
    *ans = num + 1;
    return *ans;
}

int main() {
    int *a = new int;
    *a = 2;
    int b = bump(*a);
    return 0;
}
```

0x15	a	0x23
0x16	num	2
0x17	ans	0x26

0x22		
0x23	*a	2
0x24		
0x25		
0x26	*ans	3

Heap: Example

```
int bump(int num) {
      int *ans = new int;
      *ans = num + 1;
      return *ans;
}

int main() {
      int *a = new int;
      *a = 2;
      int b = bump(*a);
      return 0;
}
```

0x15	a	0x23
0x16		
0x17		

0x22		
0x23	*a	2
0x24		
0x25		
0x26	*ans	3

Heap: Example

```
int bump(int num) {
    int *ans = new int;
    *ans = num + 1;
    return *ans;
}

int main() {
    int *a = new int;
    *a = 2;
    int b = bump(*a);
    return 0;
}
```

0x15	a	0x23
0x16	b	3
0x17		

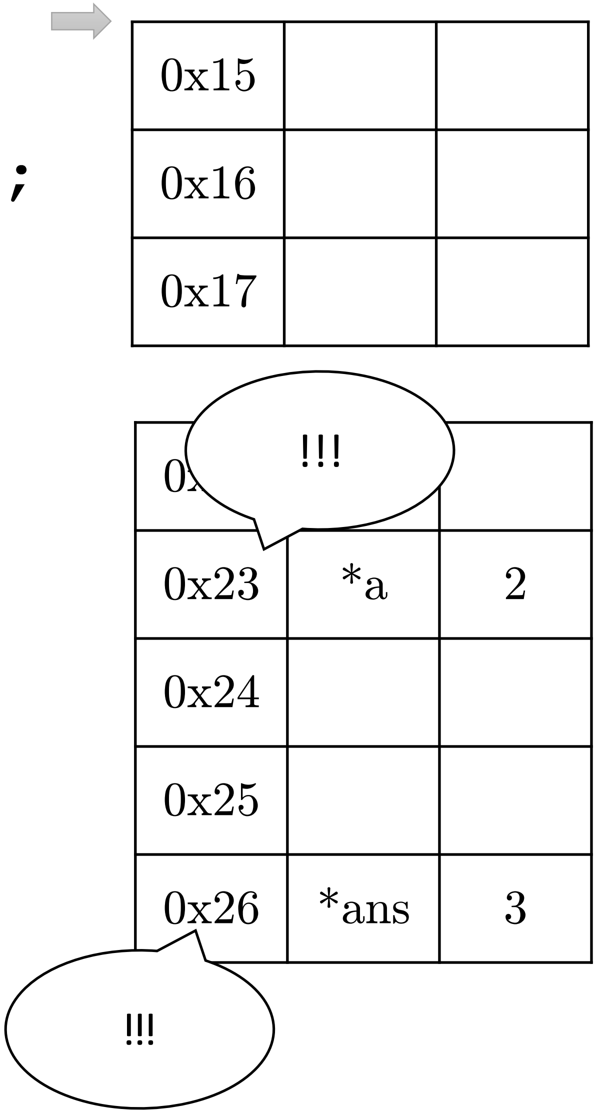
0x22		
0x23	*a	2
0x24		
0x25		
0x26	*ans	3

!!!

Heap: Example

```
int bump(int num) {
    int *ans = new int;
    *ans = num + 1;
    return *ans;
}

int main() {
    int *a = new int;
    *a = 2;
    int b = bump(*a);
    return 0;
}
```



0x15		
0x16		
0x17		

0x		
0x23	*a	2
0x24		
0x25		
0x26	*ans	3

Always de-allocate dynamic memory!

Use smart pointers!

That’s all :)

Questions?