CS107, Lecture 15

Architecture & Managing The Heap

Reading: B&O 9.9, 9.11

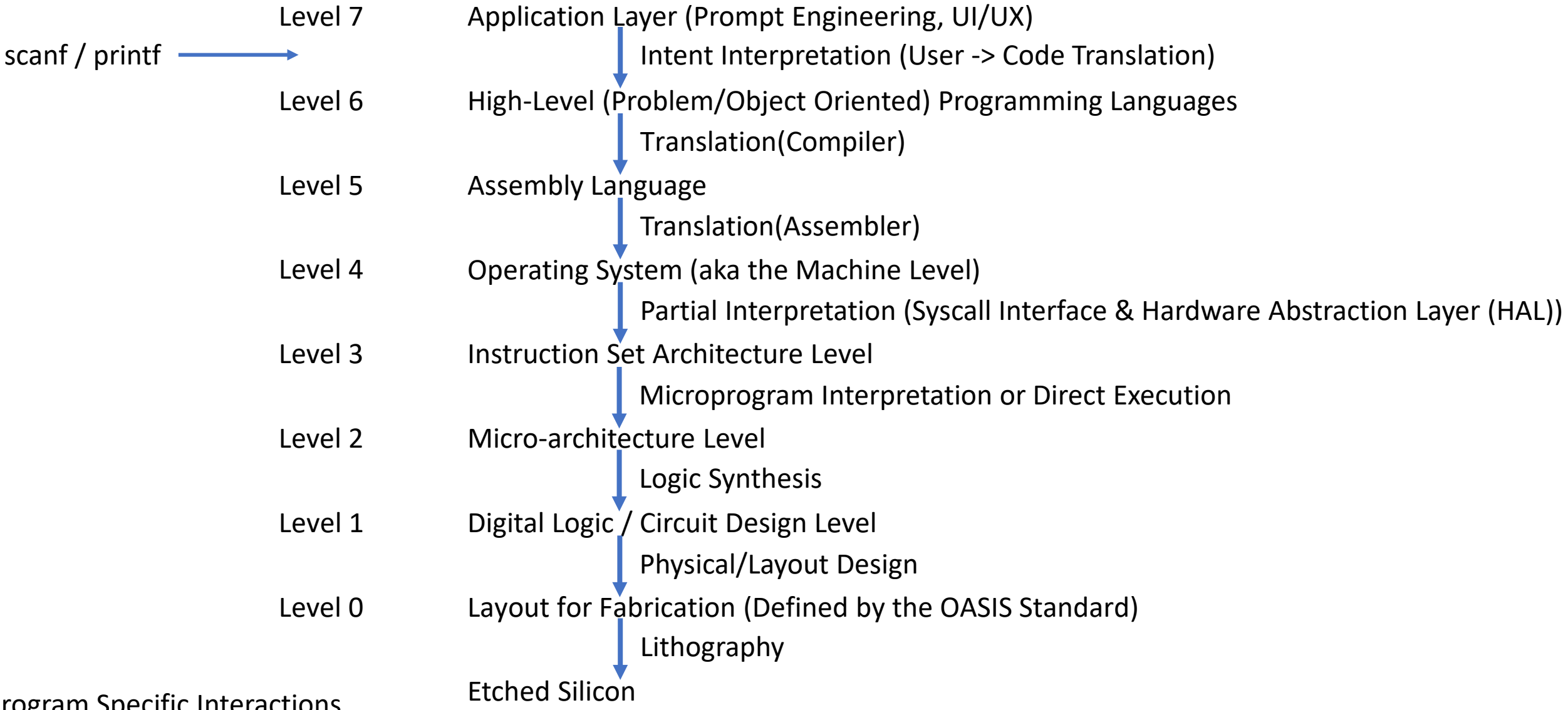
This document is copyright (C) Stanford Computer Science, Adam Keppler, and Olayinka Adekola licensed under Creative Commons Attribution 2.5 License. All rights reserved.
Based on slides created by Joel Ramirez, Nick Troccoli, Chris Gregg

Lecture Plan

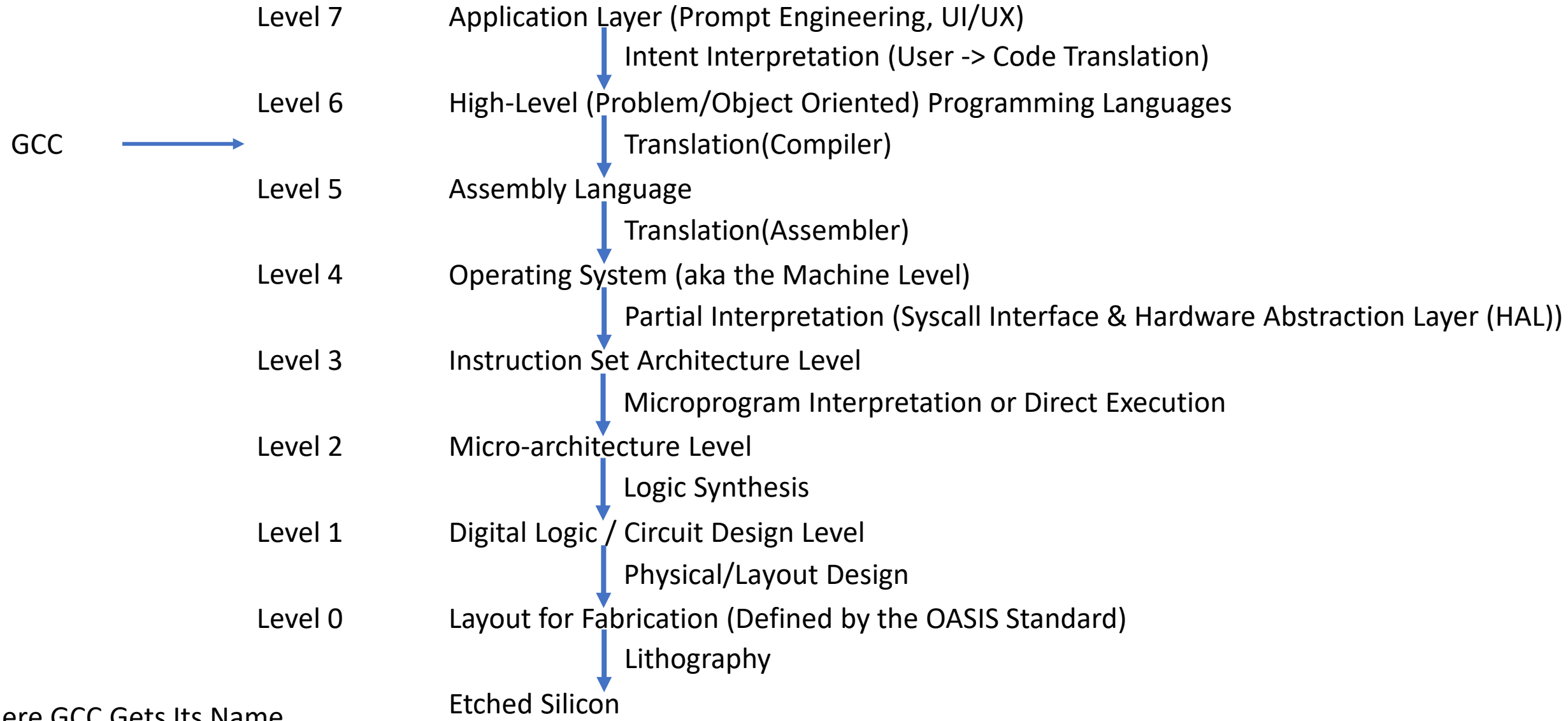
- **Into the Architecture!**
- Peeking at Memory
- Architecture & The ISA
- The heap so far
- What is a heap allocator?
- Heap allocator requirements and goals
- Method 0: Bump Allocator

Into the Architecture!

Programming Levels

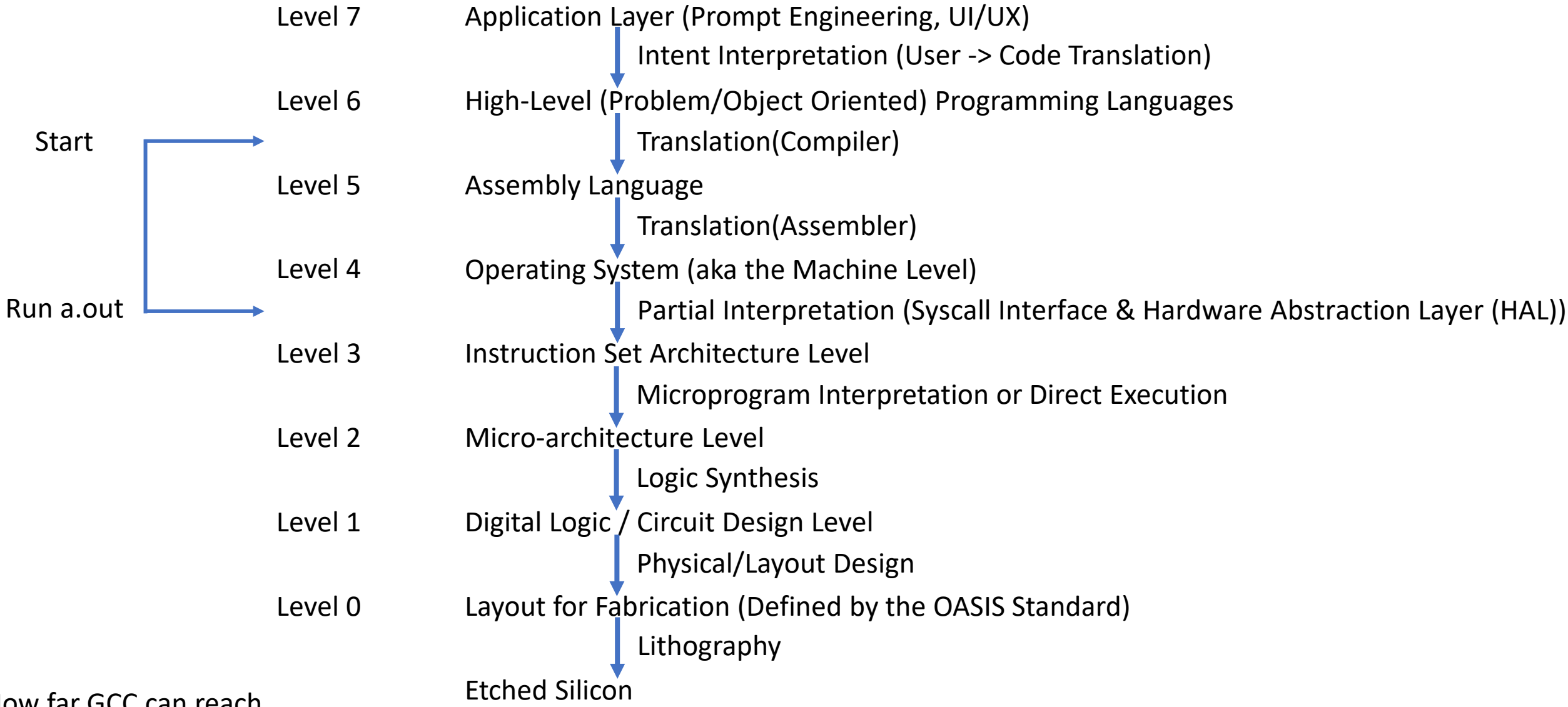


Programming Levels

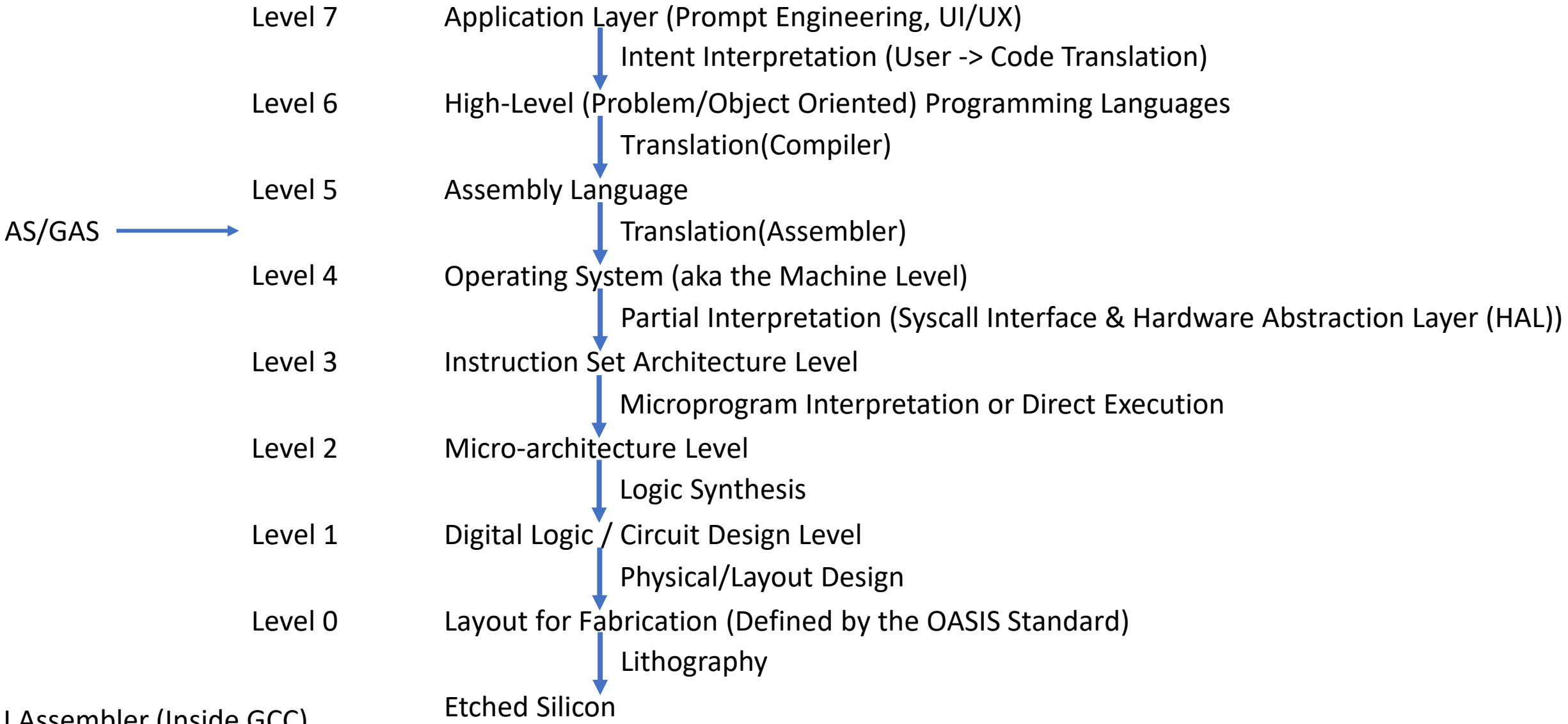


Where GCC Gets Its Name

Programming Levels

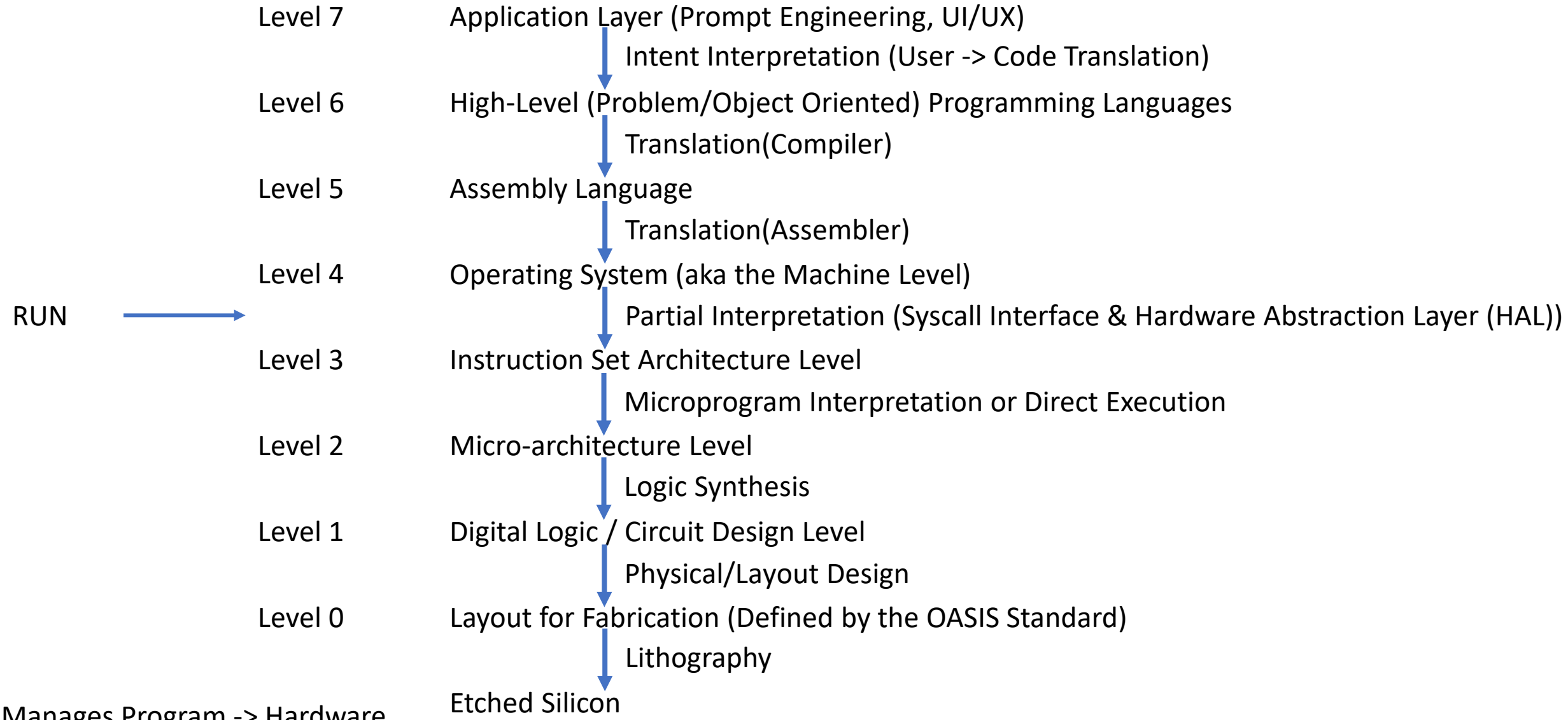


Programming Levels



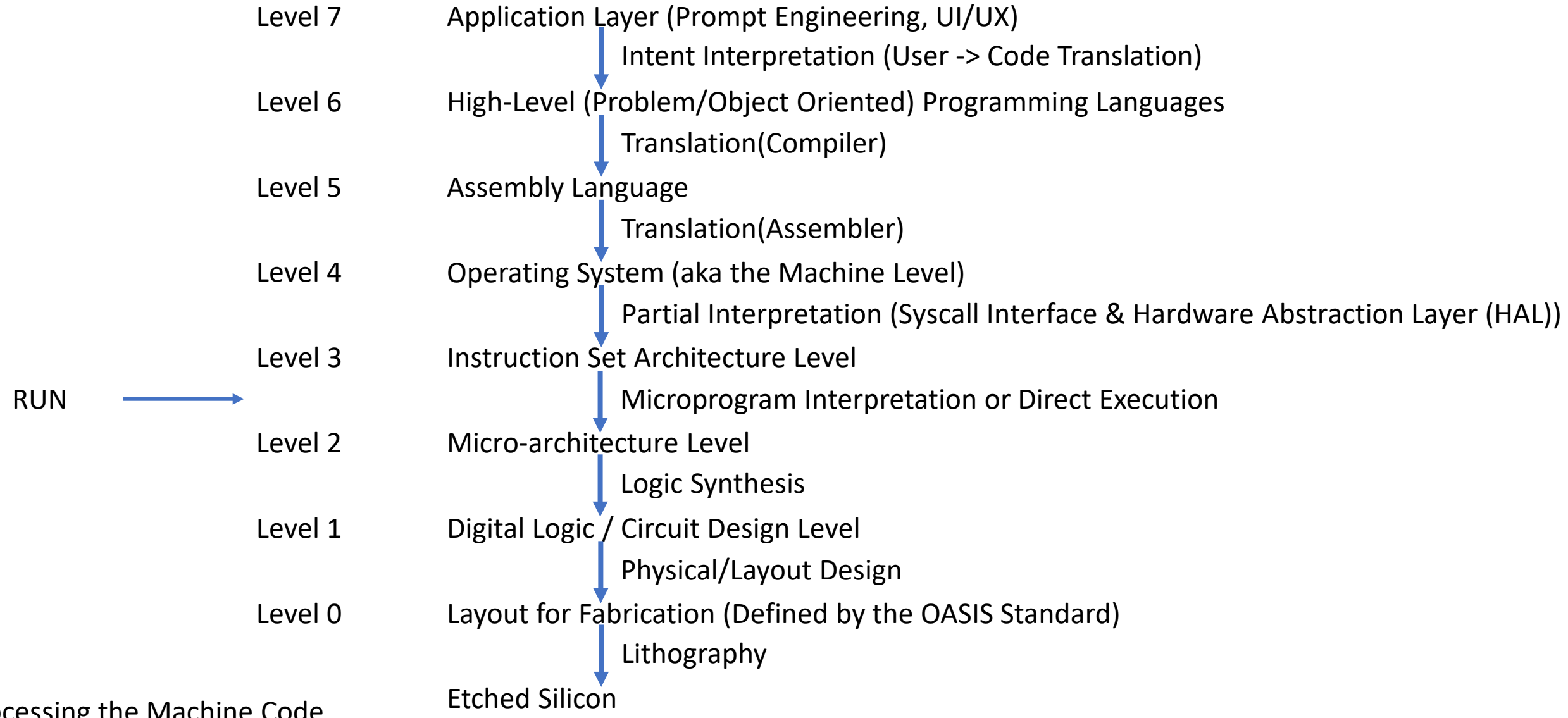
GNU Assembler (Inside GCC)

Programming Levels

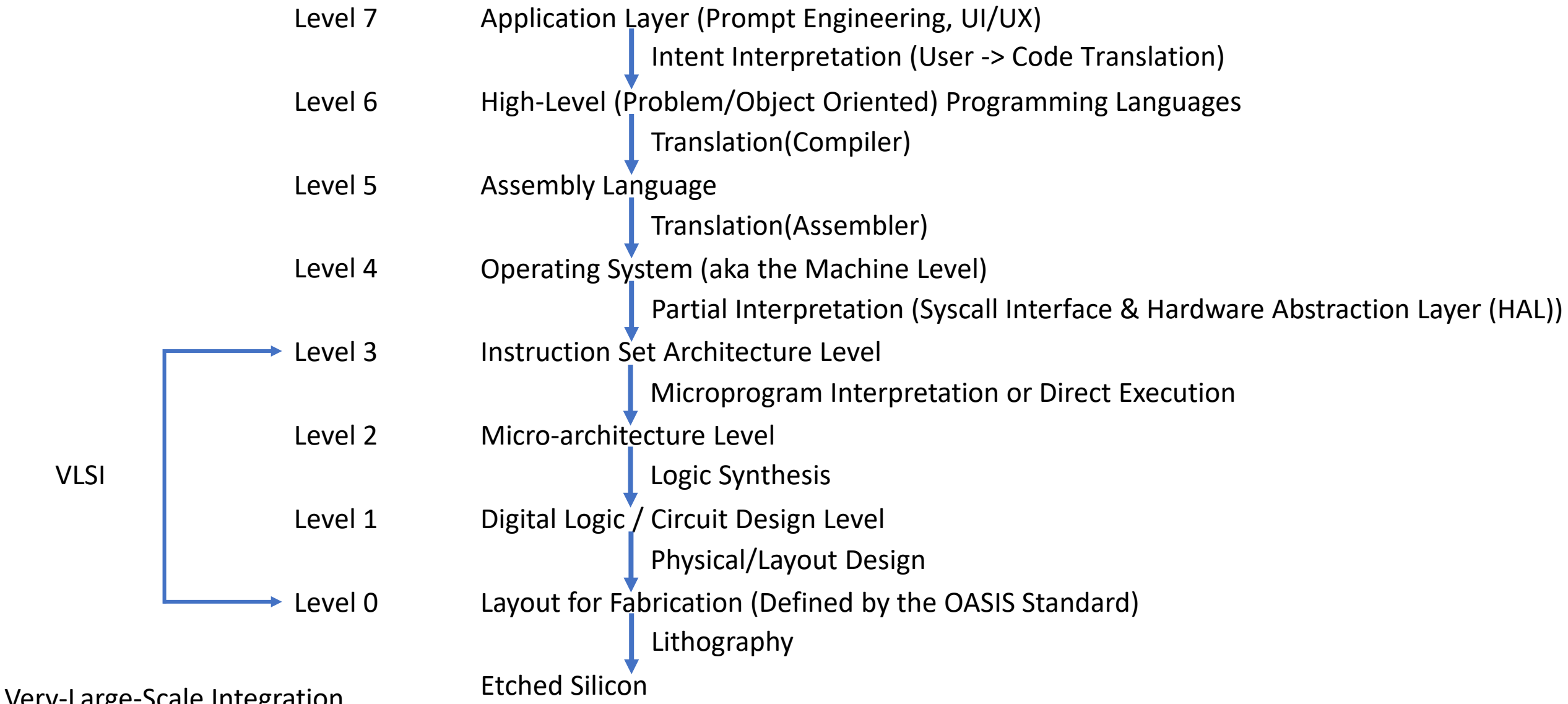


OS Manages Program -> Hardware

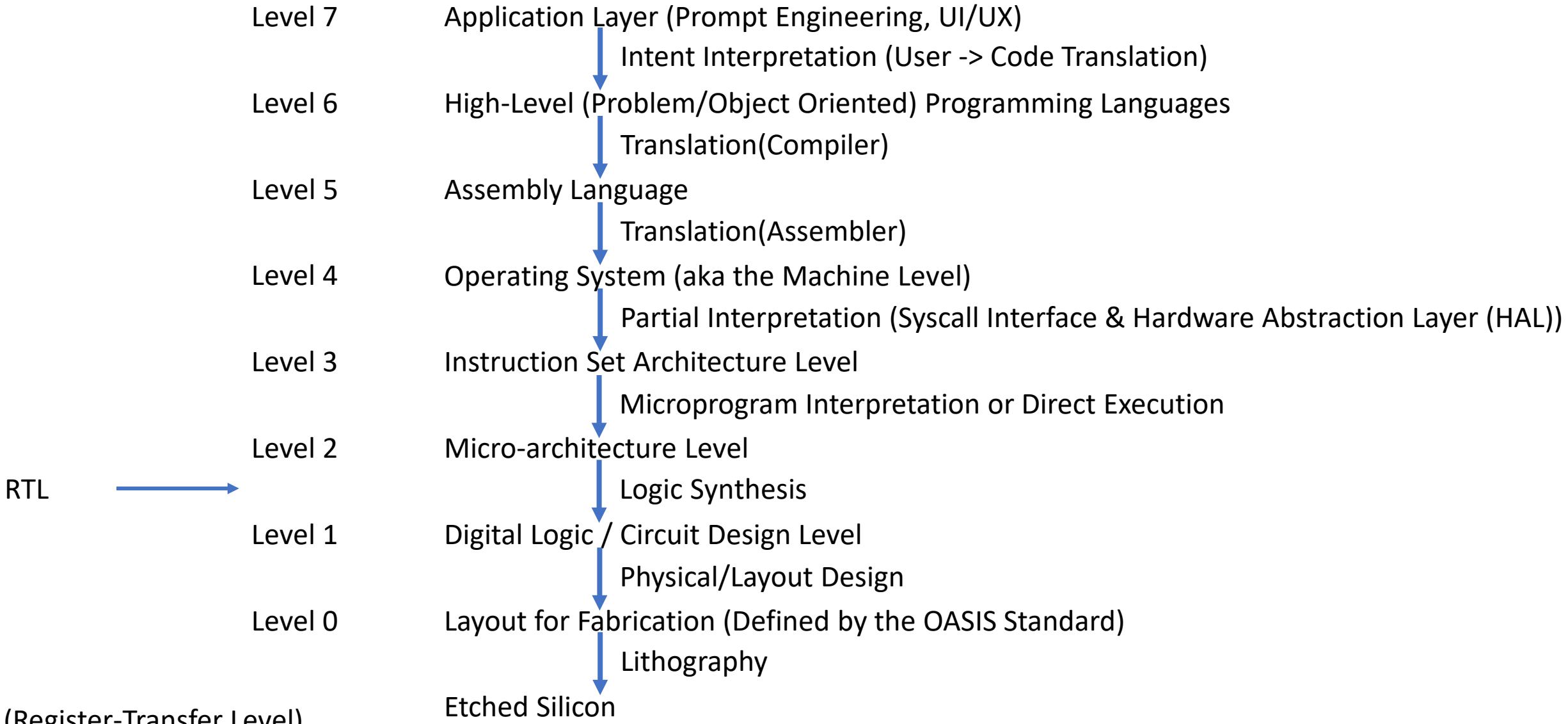
Programming Levels



Programming Levels

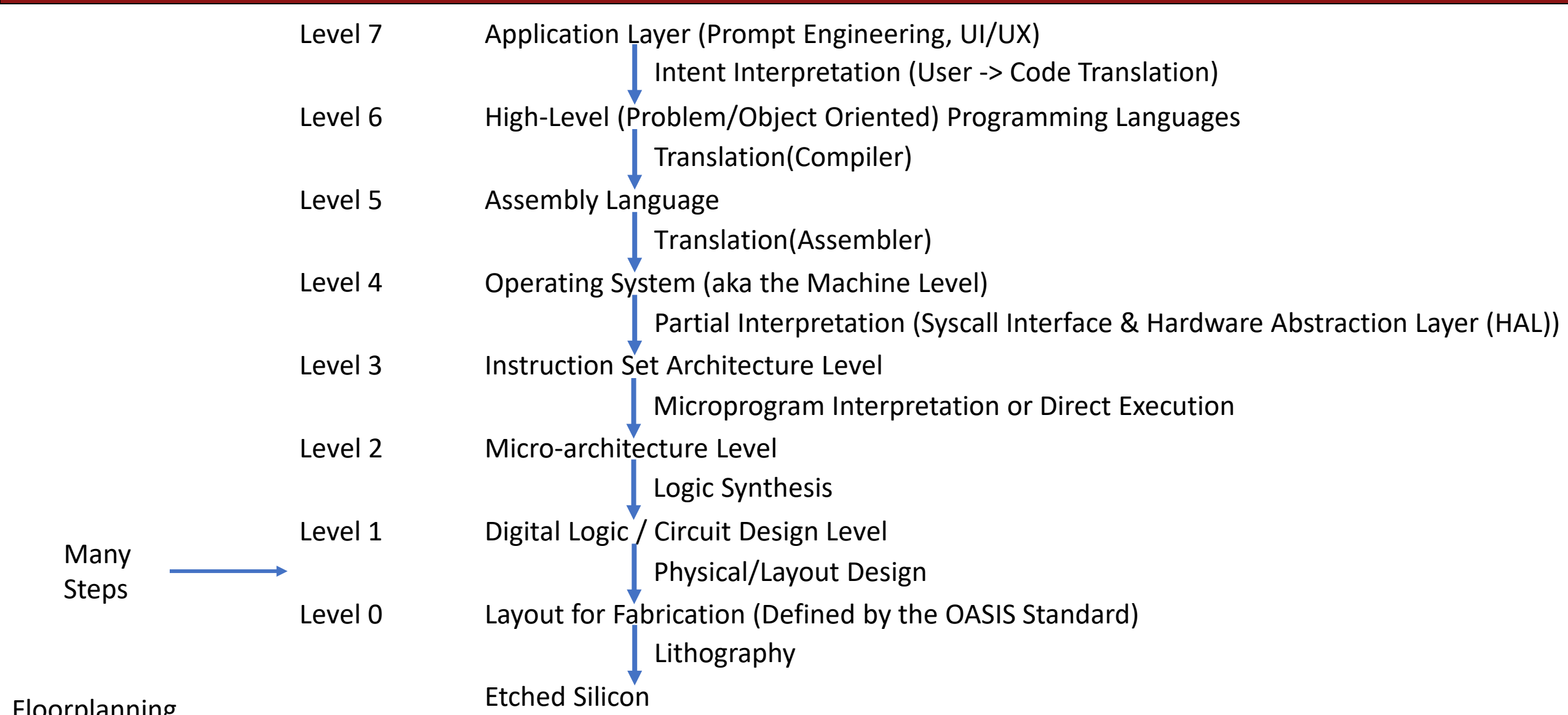


Programming Levels



RTL (Register-Transfer Level)

Programming Levels



Programming Levels

Level 7 — Application Layer (Prompt Engineering, UI/UX)

↓ Intent Interpretation (User -> Code Translation)

Level 6 — High-Level (Problem/Object Oriented) Programming Languages

↓ Translation(Compiler)

Level 5 — Assembly Language

↓ Translation(Assembler)

Level 4 — Operating System (aka the Machine Level)

↓ Partial Interpretation (Syscall Interface & Hardware Abstraction Layer (HAL))

Level 3 — Instruction Set Architecture Level

↓ Microprogram Interpretation or Direct Execution

Level 2 — Micro-architecture Level

↓ Logic Synthesis

Level 1 — Digital Logic / Circuit Design Level

↓ Physical/Layout Design

Level 0 — Layout for Fabrication (Defined by the OASIS Standard)

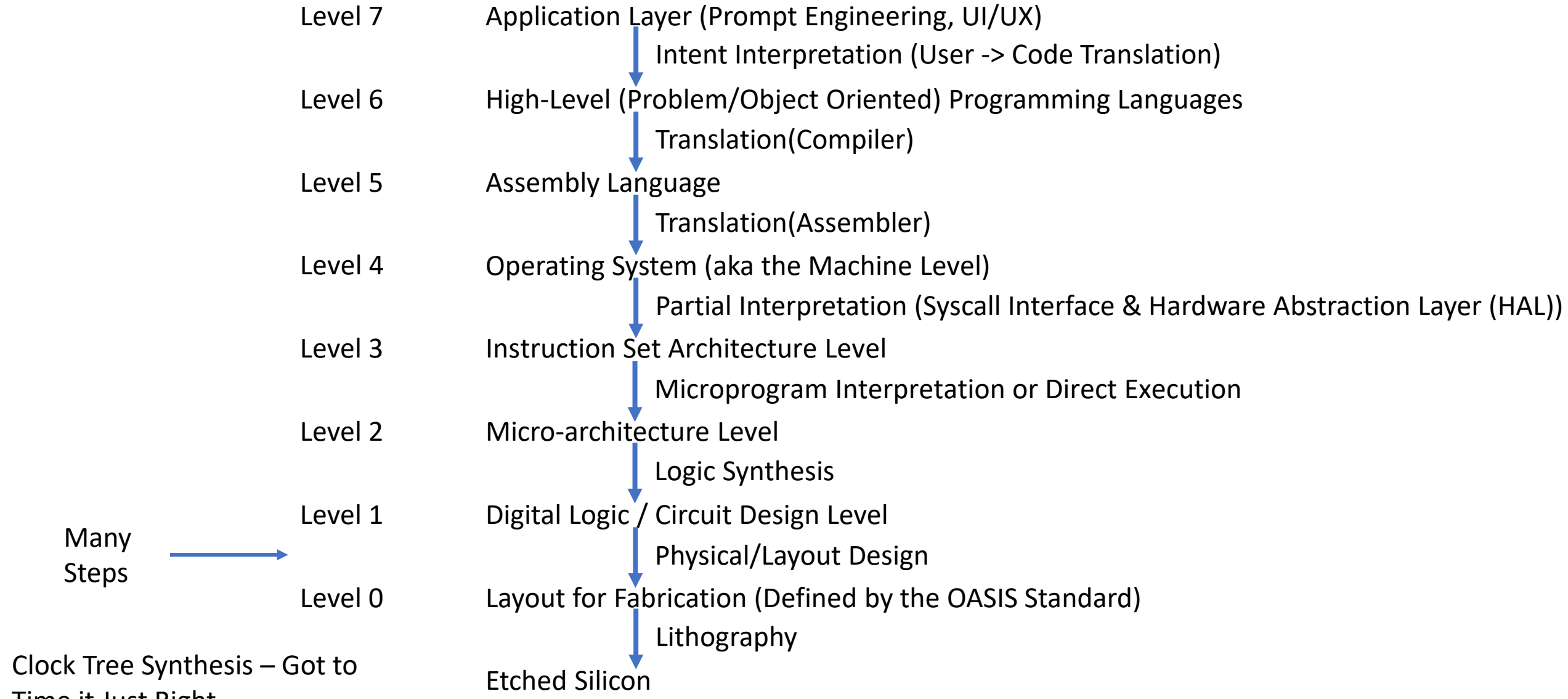
↓ Lithography

Etched Silicon

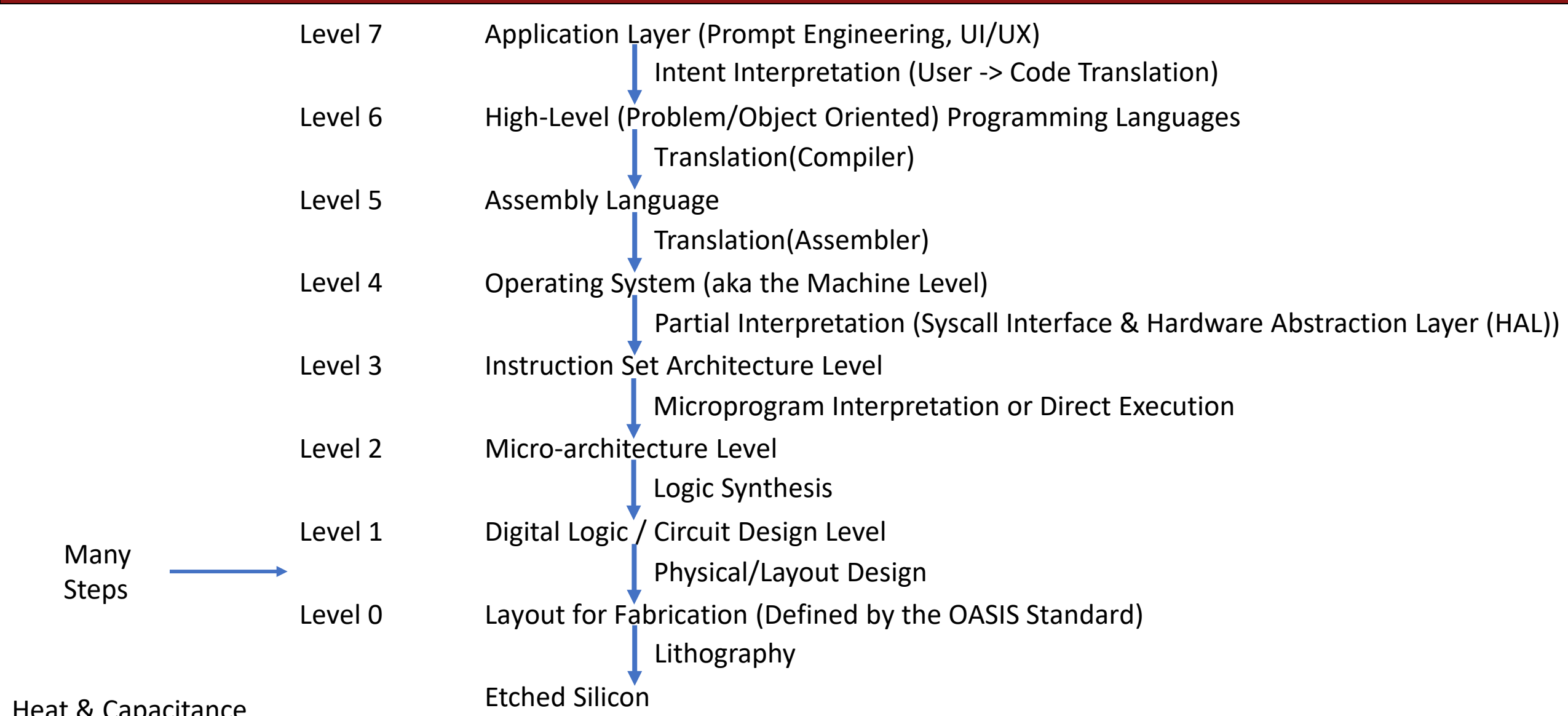
Many Steps →

Wire Routing
– Don't Cross the Wires

Programming Levels



Programming Levels



Programming Levels

Level 7	Application Layer (Prompt Engineering, UI/UX)
	↓ Intent Interpretation (User -> Code Translation)
Level 6	High-Level (Problem/Object Oriented) Programming Languages
	↓ Translation(Compiler)
Level 5	Assembly Language
	↓ Translation(Assembler)
Level 4	Operating System (aka the Machine Level)
	↓ Partial Interpretation (Syscall Interface & Hardware Abstraction Layer (HAL))
Level 3	Instruction Set Architecture Level
	↓ Microprogram Interpretation or Direct Execution
Level 2	Micro-architecture Level
	↓ Logic Synthesis
Level 1	Digital Logic / Circuit Design Level
	↓ Physical/Layout Design
Level 0	Layout for Fabrication (Defined by the OASIS Standard)
ASML →	↓ Lithography
	Etched Silicon

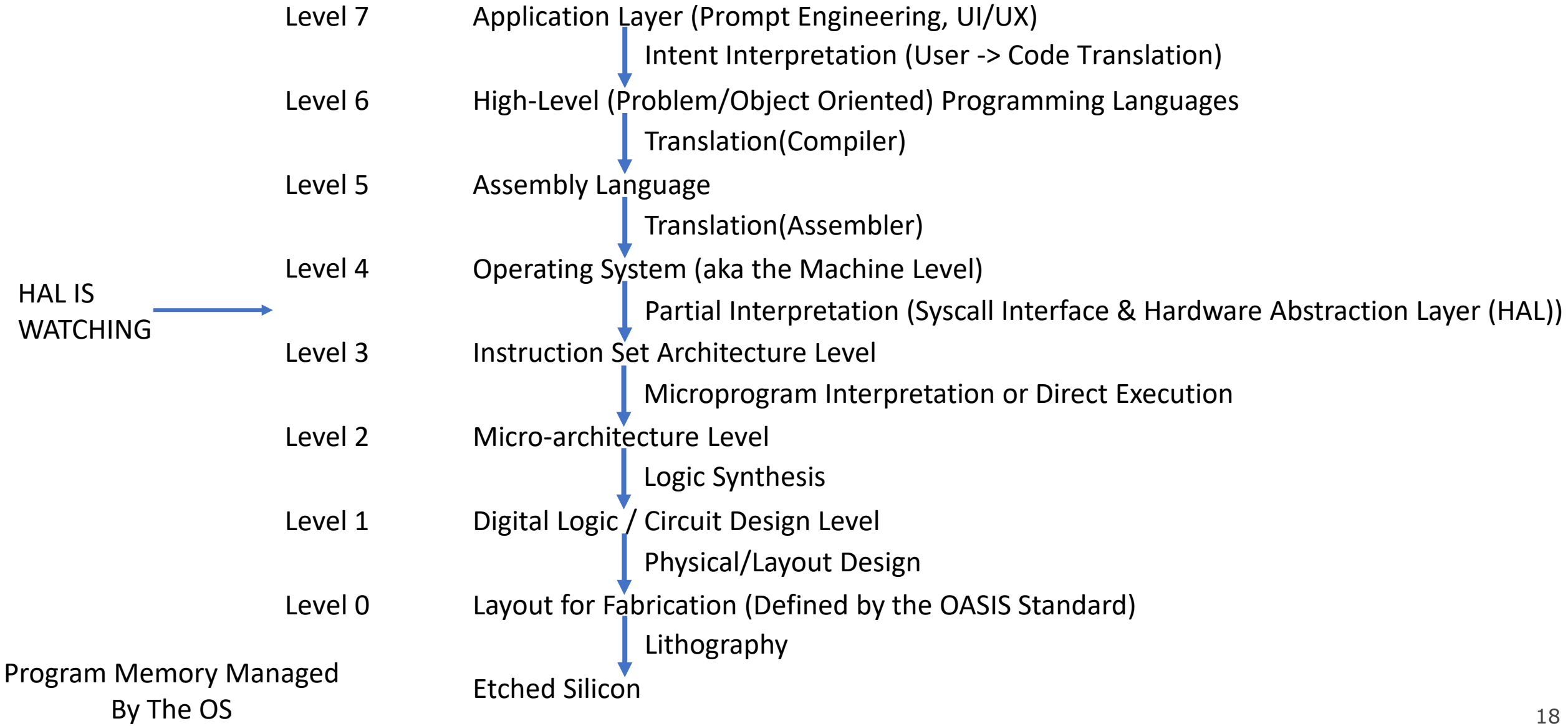
Checkout EUV Lithography

Programming Levels

Level 7	Application Layer (Prompt Engineering, UI/UX)
	↓ Intent Interpretation (User -> Code Translation)
Level 6	High-Level (Problem/Object Oriented) Programming Languages
	↓ Translation(Compiler)
Level 5	Assembly Language
	↓ Translation(Assembler)
Level 4	Operating System (aka the Machine Level)
	↓ Partial Interpretation (Syscall Interface & Hardware Abstraction Layer (HAL))
Level 3	Instruction Set Architecture Level
	↓ Microprogram Interpretation or Direct Execution
Level 2	Micro-architecture Level
	↓ Logic Synthesis
Level 1	Digital Logic / Circuit Design Level
	↓ Physical/Layout Design
Level 0	Layout for Fabrication (Defined by the OASIS Standard)
	↓ Lithography
	Etched Silicon

Which layer throws a segfault?

Programming Levels

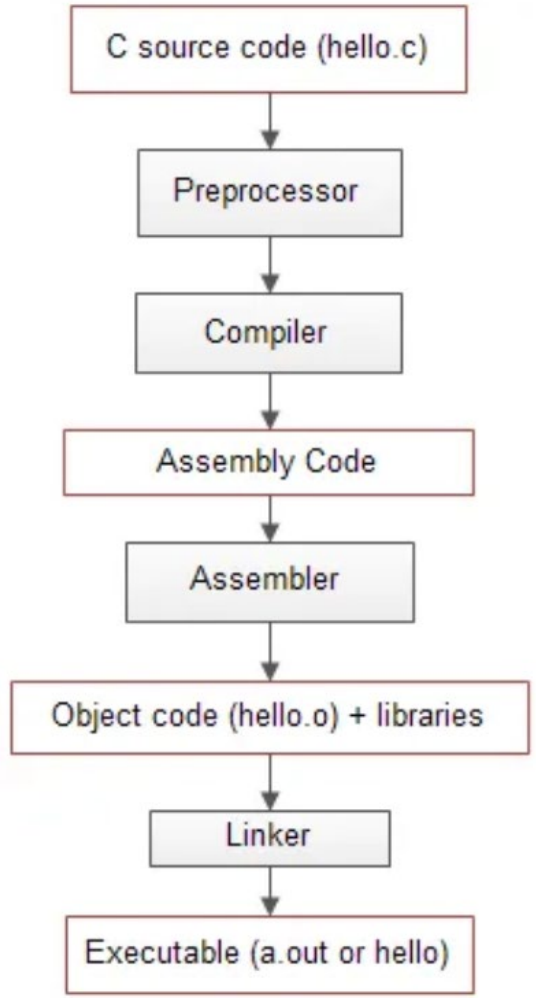


More on the Compiler

How Does GCC Work?

- One Unix Command – A lot of steps!

gcc hello.c -o hello

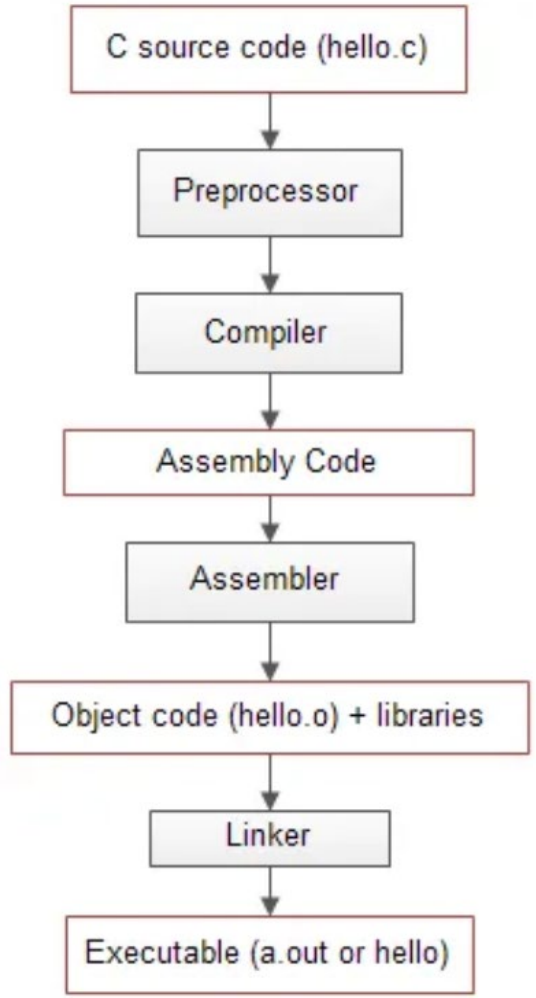


https://medium.com/@tuvo1106/the-gcc-compilation-process-8accb463e227

How Does GCC Work?

- Preprocessing – Handle Programmer Conveniences
  - #Macros convert to normal C code
  - Lines split by \ are joined
  - Comments are removed
    - NOTE: Some comments are added, but our comments are removed
  - Bring in functions and variables from the headers
    - This is how the #include is resolved

gcc -E hello.c > pre_processed_hello



https://medium.com/@tuvo1106/the-gcc-compilation-process-8accb463e227

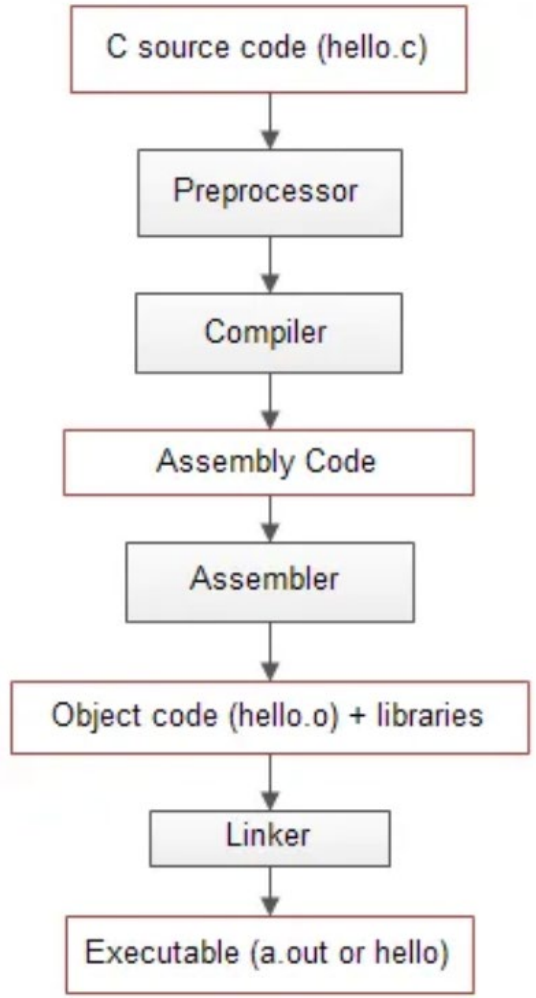
How Does GCC Work?

- Compilation – C to Assembly

gcc -S hello.c

- Will generate intermediate ‘human-readable’ assembly

- There are different styles/syntax for x86, we use AT&T
  - AT&T is also the gcc default



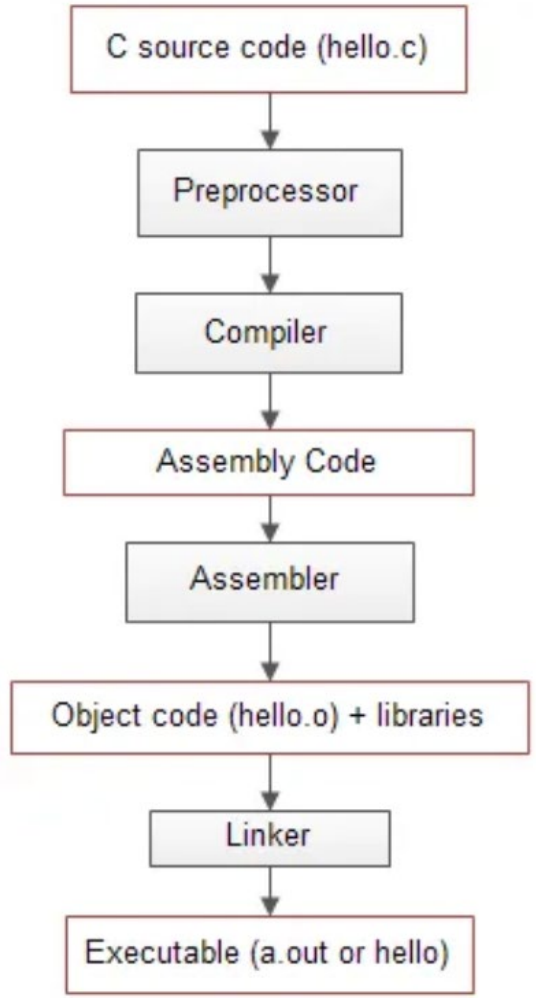
https://medium.com/@tuvo1106/the-gcc-compilation-process-8accb463e227

How Does GCC Work?

- Object Generation – C to Object File

gcc -c hello.c

- “Just compile; Don't link"
- This outputs a non-human readable Object File
  - It is defined as a type of incomplete machine code
  - With extra metadata to power linking
- Using objdump –d hello.o , we can see the assembly



https://medium.com/@tuvo1106/the-gcc-compilation-process-8accb463e227

How Does GCC Work?

- Linking – Bringing All the pieces together
  - Object Files & Libraries -> Fully Executable Machine Code

gcc hello.o -o hello

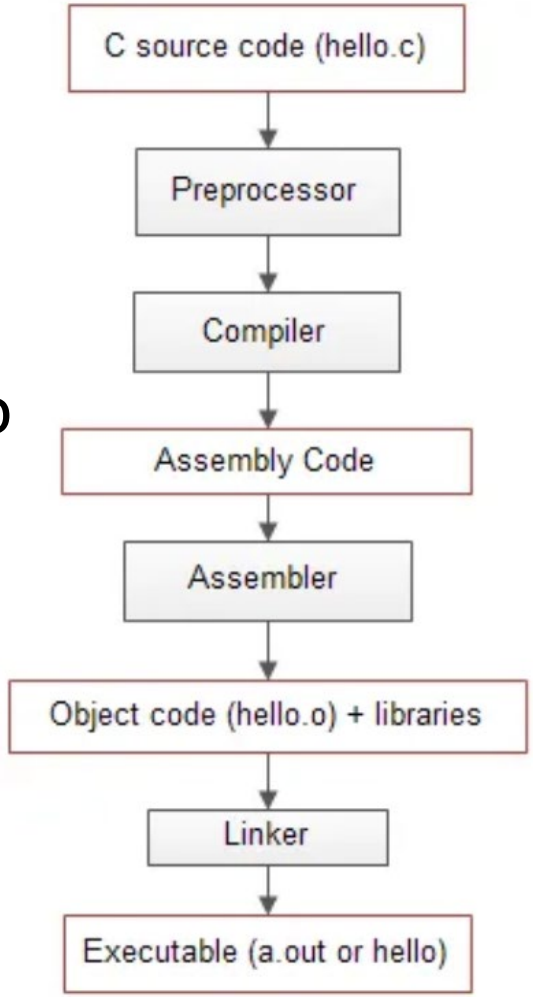
ld -o hello hello.o -lc -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crti.o /usr/lib/x86_64-linux-gnu/crtn.o

- NOTE: We can get our .o in more than one-way

gcc -c hello.c

OR

as hello.s



https://medium.com/@tuvo1106/the-gcc-compilation-process-8accb463e227

What does the Assembler Do?

A Two Step Process

- Pass 1: Setup Memory Addresses
  - The program reads in the assembly program identifying and tracking:
    - Labels
    - Literals
    - Data Variables

- Pass 2: Generate the Machine Code (Byte/Binary Code)
  - Identify Opcode from the mnemonic assembly
  - Resolve labels/literals/variables using the tables from Step 1
  - Convert Data to Binary
  - Identifies External (Out of Program) References and places markers for the Linker
  - Setup Metadata for linking if this program has loadable parts

Final Output is not runnable, but has all the parts need if linking can complete

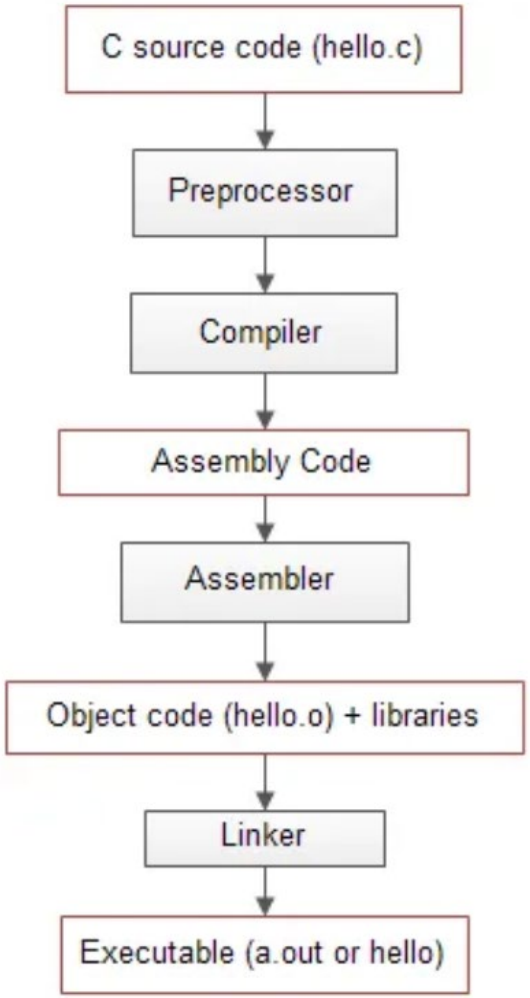
Why do we need a linker?

Many Links

- Every C file corresponds to a .o
- Libraries can also be made into linkable formats
- We don’t want to have to write all our code in 1 file and we want to use the STL
- Incremental Builds, change some files instead of all the files
- The linker makes this all possible

How Does GCC Work?

- Multi-Step Process -> Multiple Failure Points
- Compilation can fail for many reasons at different points
- Mainly two areas that fail ‘Compilation’ or Linking
- If compilation succeeds, Intermediate Assembly will be good!



https://medium.com/@tuvo1106/the-gcc-compilation-process-8accb463e227

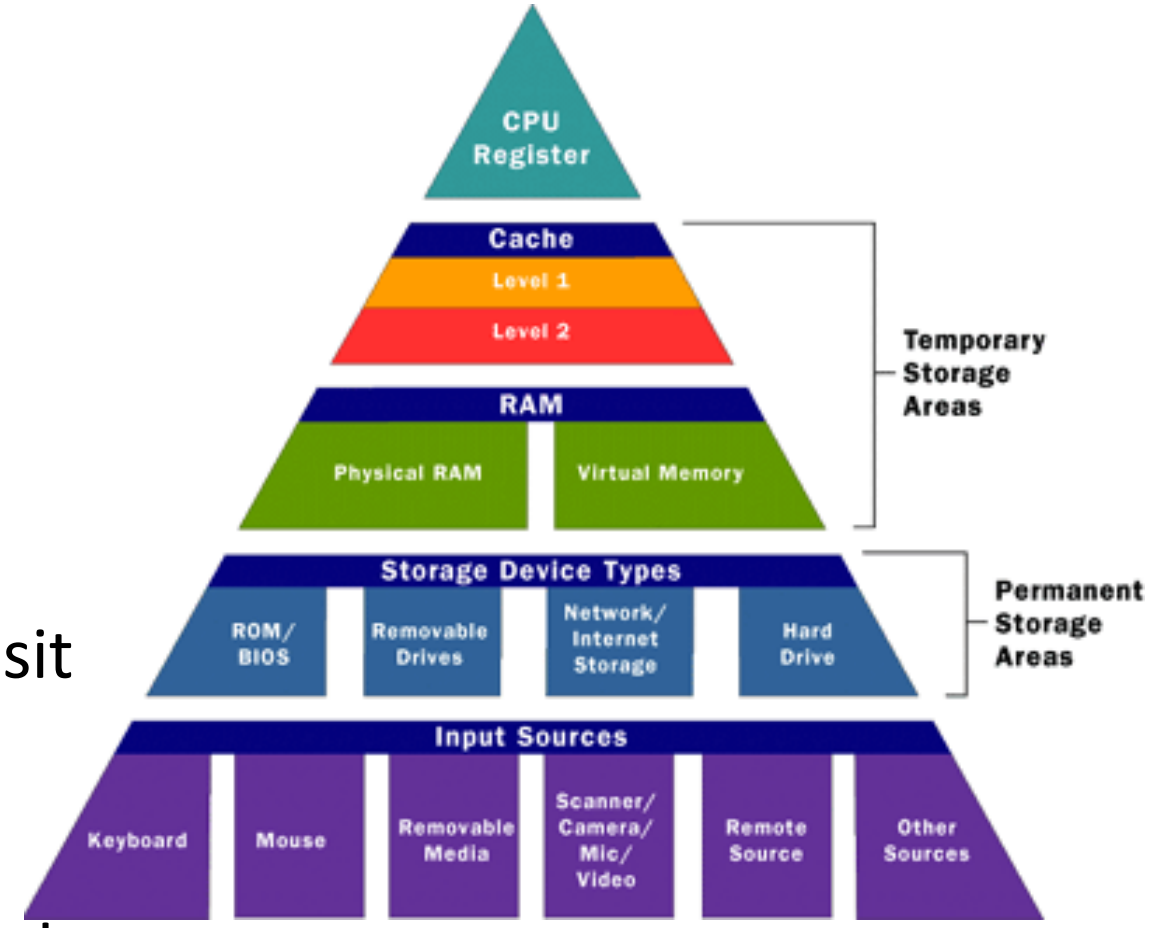
Lecture Plan

- Into the Architecture!
- **Peeking at Memory**
- Architecture & The ISA
- The heap so far
- What is a heap allocator?
- Heap allocator requirements and goals
- Method 0: Bump Allocator

Peeking at Memory

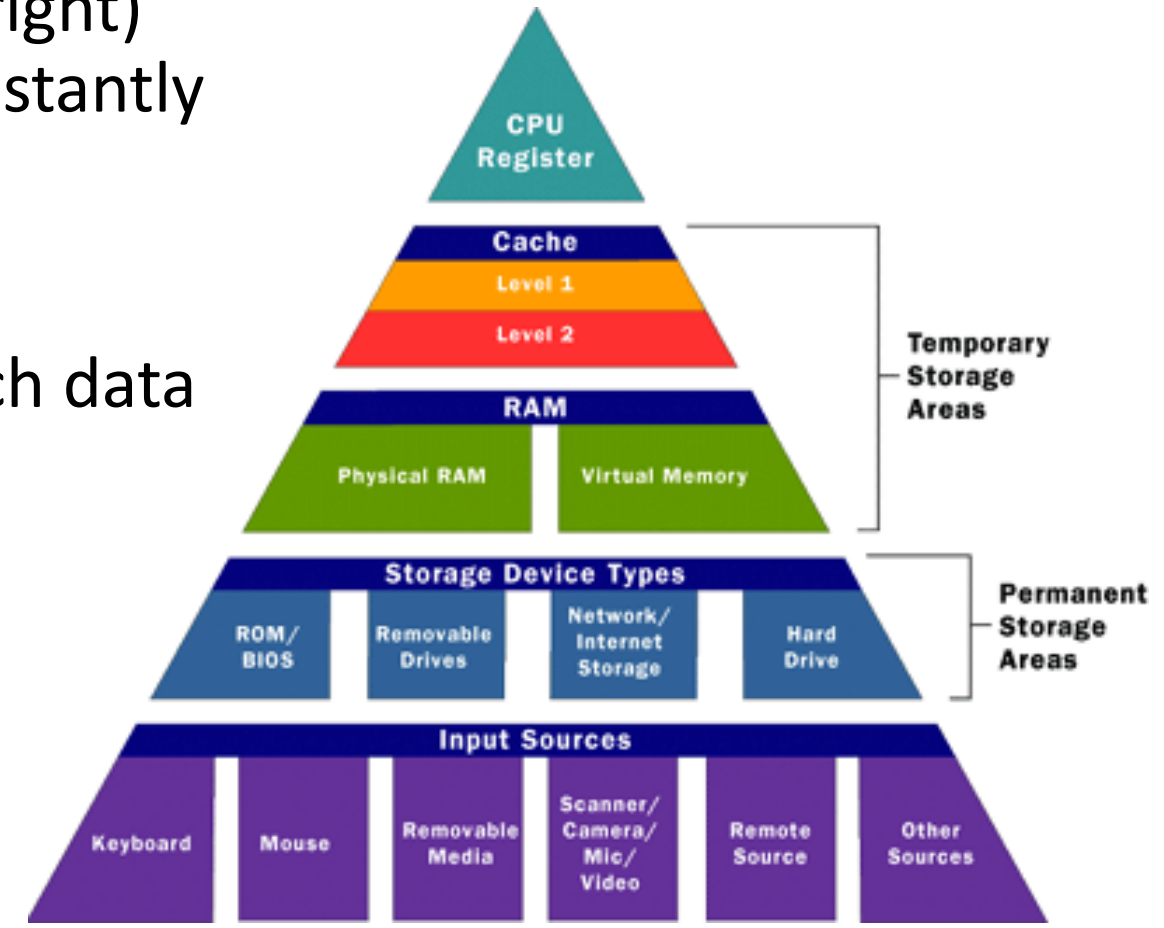
Speed vs Space

- CPU is the most important place
  - Closer to CPU, less travel time
  - But limited space, so bottleneck getting there
- Think of the CPU like downtown, generally expensive and highly desirable real estate
- The BUS (actual technical name) is our transit system around the computer
- Places close to the CPU are more limited and more valuable, since they can get to the CPU faster



Speed vs Space

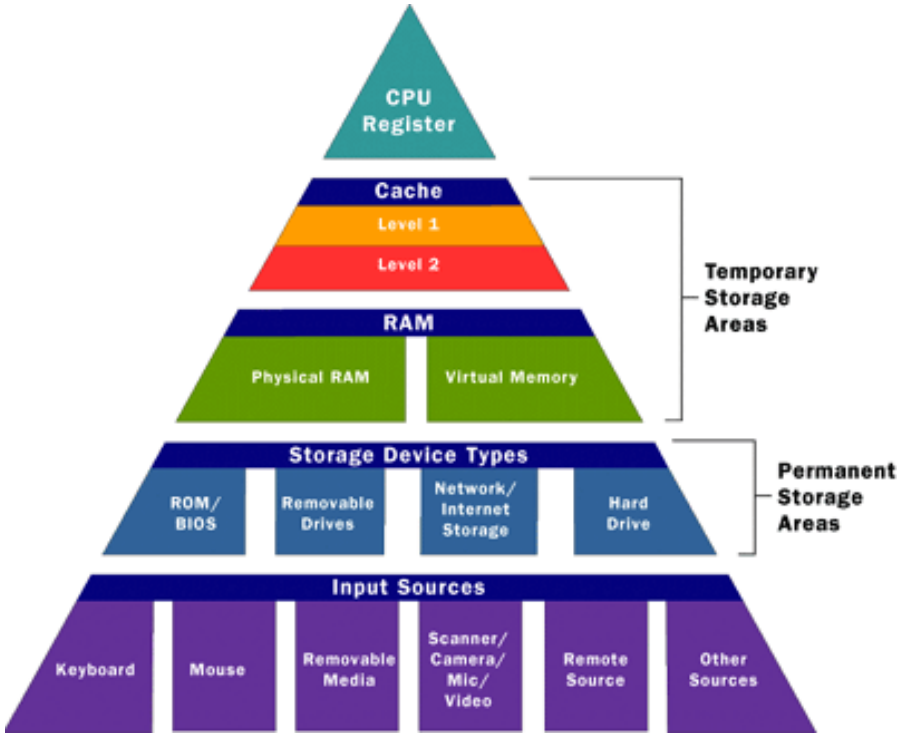
- All of Memory (Temporary Storage on the right) and the registers is rent only, so data is constantly moving around
- Many algorithms developed to decide which data gets to live where and for how long
- Proper access makes a huge difference on performance



Speed vs Space

- Approximate Access Times

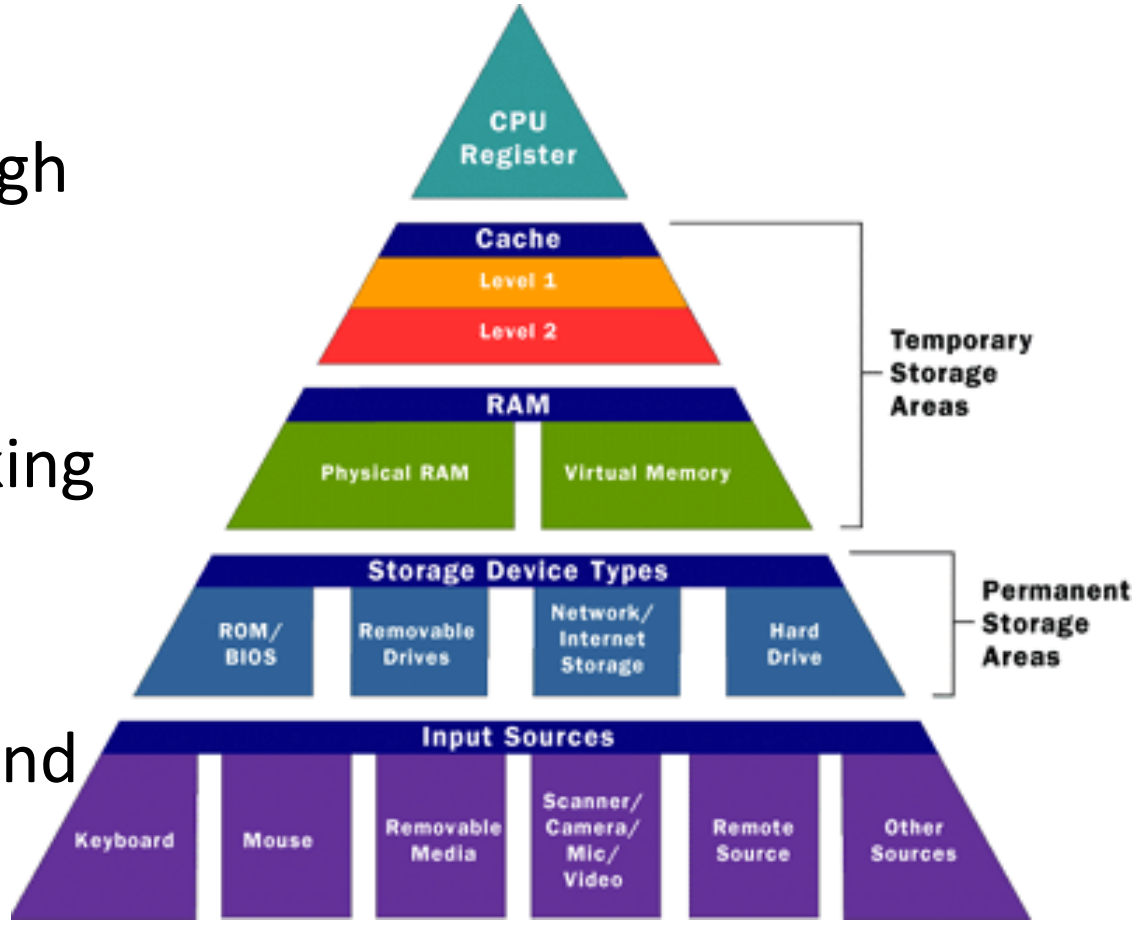
Resource	Latency Time
Register	0 Cycles (already here)
Level 1 Cache	~0.5 ns
Level 2 Cache	~7 ns (14x L1)
RAM	~100 ns (20x L2, 200x L1)
SSD	~100-150 us (~14Kx L2, 200Kx L1)
Hard (Spinning) Disk	~10 ms (~2.8Mx L2, 40Mx L1)
Network Packet CA -> Netherlands -> CA	~150 ms (~21Mx L2, 300Mx L1)
Average Human Response Time to Visual Stimulus	~200 ms (~28Mx L2, 400Mx L1)



For more on speed checkout:
https://www.cs.princeton.edu/courses/archive/spring20/cos217/lectures/20_Mem_Storage_Hierarchy.pdf
https://gist.github.com/jboner/2841832

Speed vs Space

- Pre-emptive requests and moving of data is critical
- Orders of Magnitude Improvements from high locality
- Every part of the pyramid is working on making this faster
- Better BUS, faster storage(both temporary and permanent), bigger RAM, better algorithms

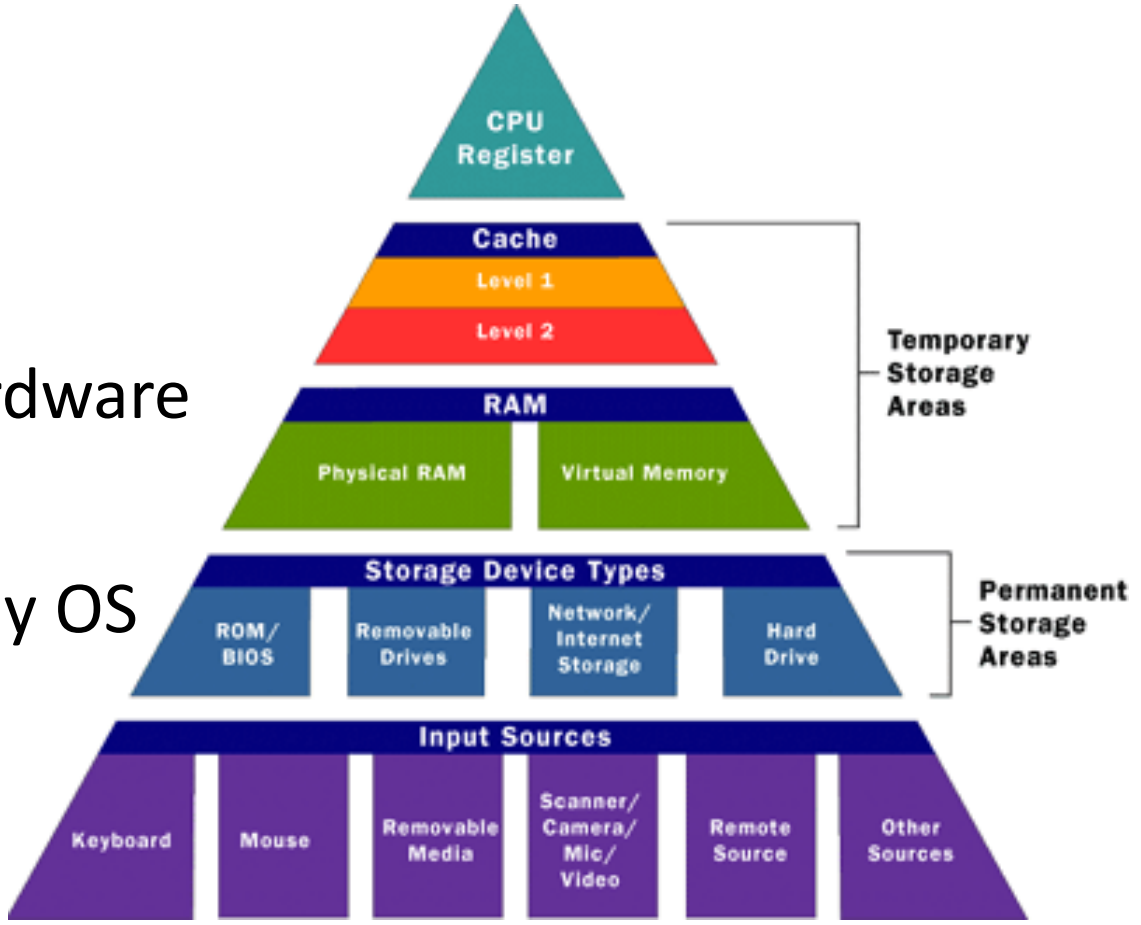


What is Locality?

- Temporal Locality
  - Has the data been used recently? Then we expect to be used again soon
- Spatial Locality
  - The data appears close together in the program/memory, so it will likely be needed at the same time.
- Hardware and OS designers consider algorithms to predict and leverage locality to optimize management of memory resources
- Cache in particular is a limited resource and must be used effectively to leverage benefits

Who Gets to Manage the Memory?

- Registers – Managed by the Compiler/Assembler
- Cache – Managed by Hardware Designers
- Memory – Mainly the OS, influenced by hardware
- Disk – Managed by the user and occasionally OS

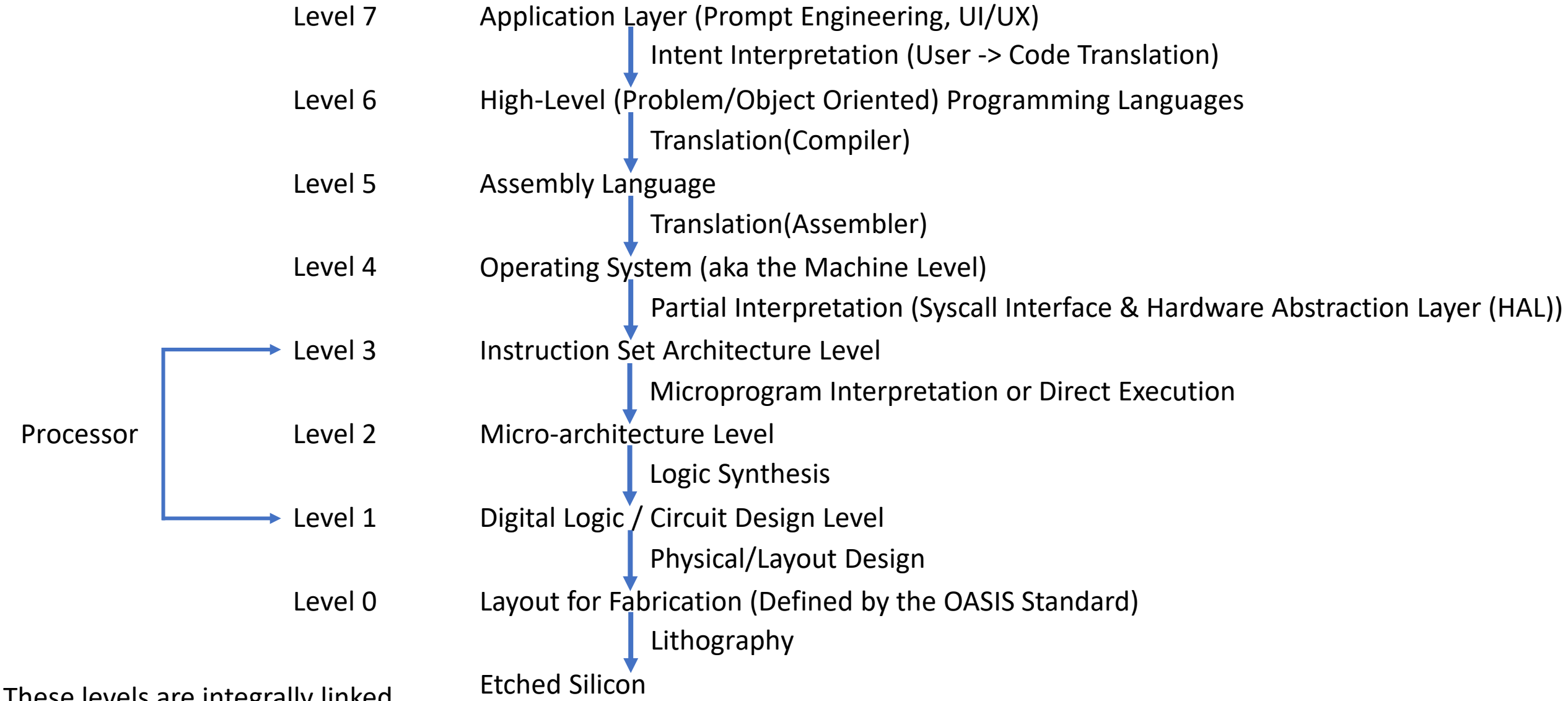


Lecture Plan

- Into the Architecture!
- Peeking at Memory
- **Architecture & The ISA**
- The heap so far
- What is a heap allocator?
- Heap allocator requirements and goals
- Method 0: Bump Allocator

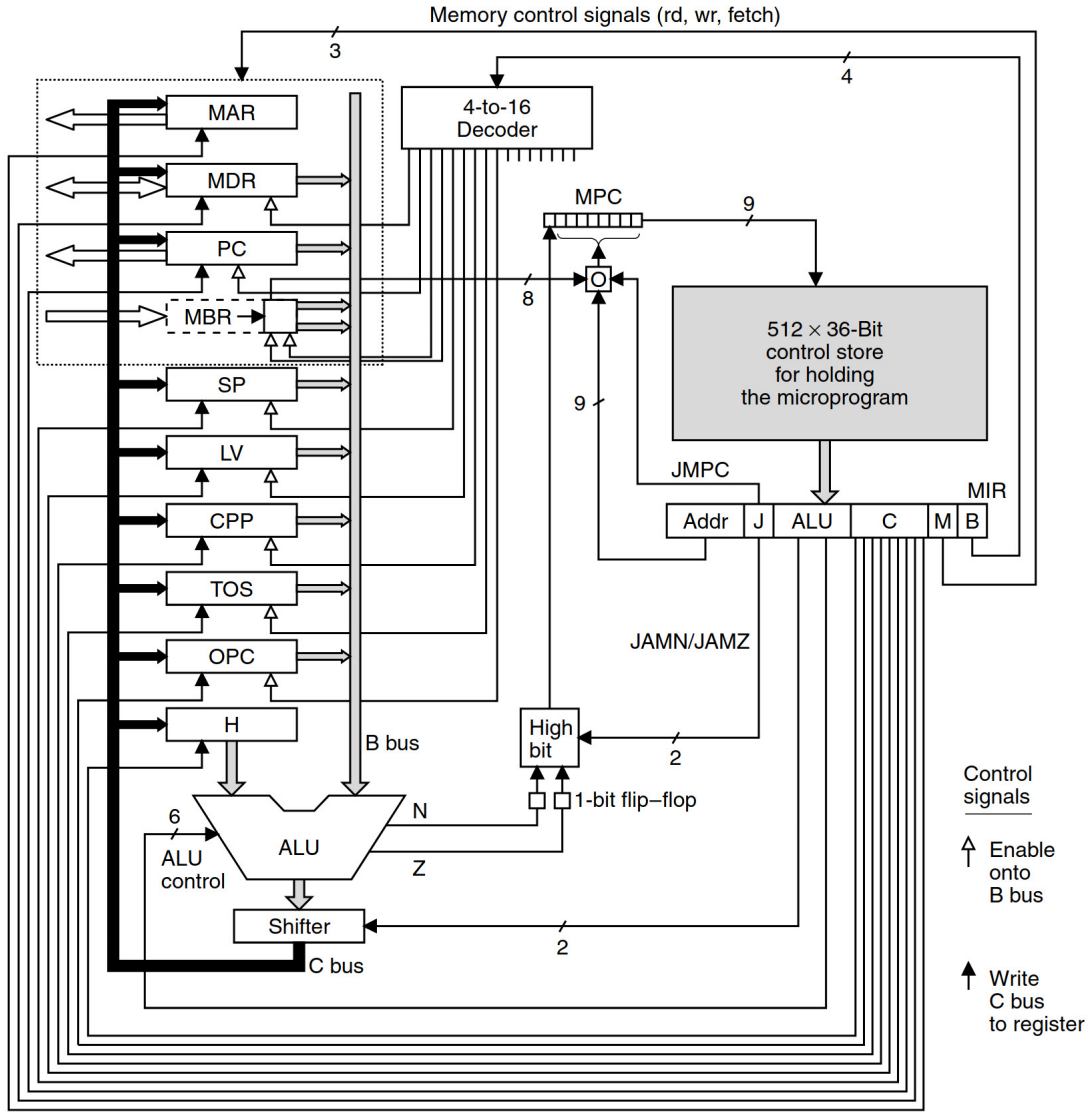
Architecture & The ISA

Programming Levels



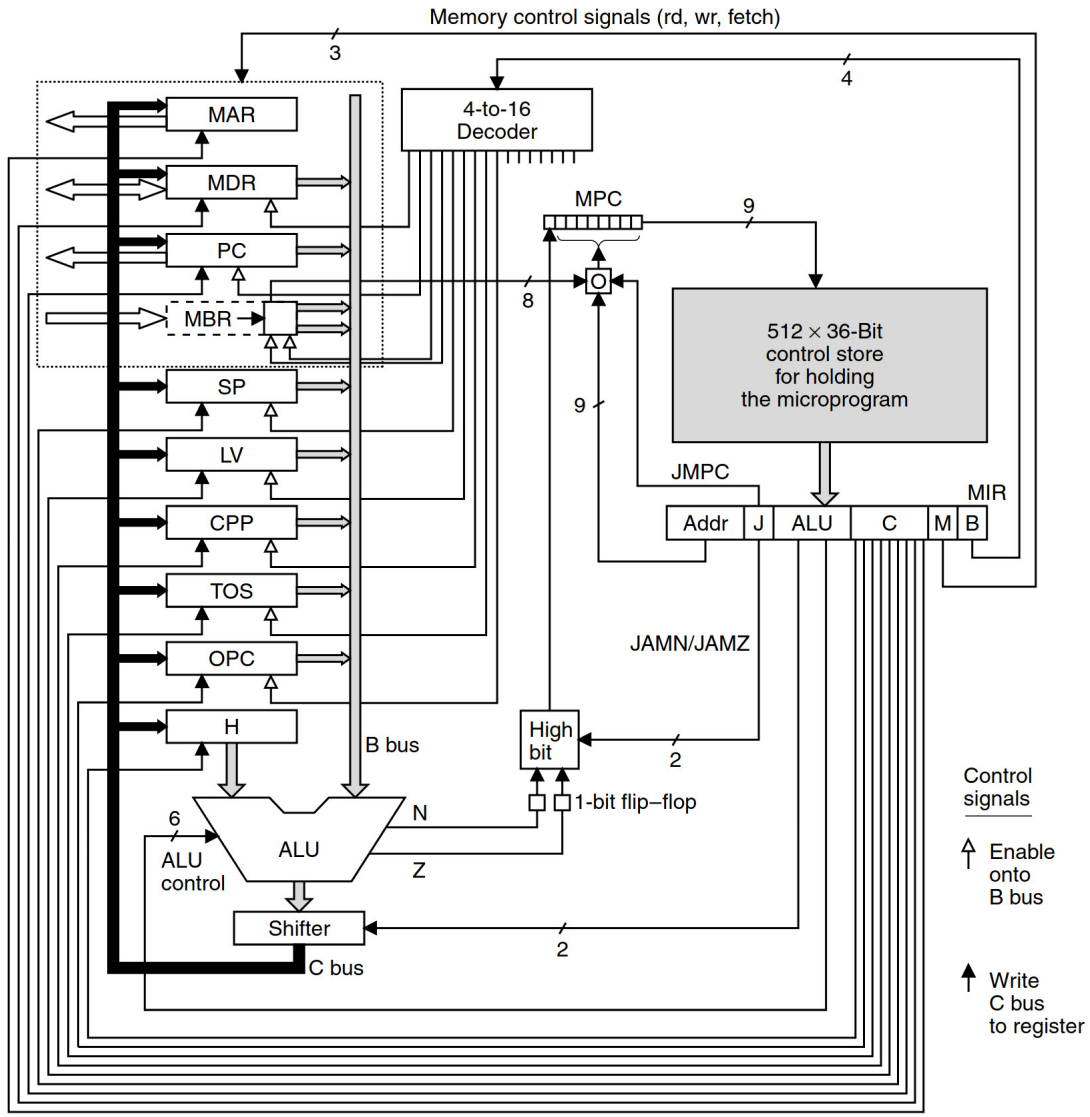
A ‘Simple’ Example

- MIC-1 Architecture (Tanenbaum - Structured Computer Organization 6th Edition)
- IJVM ISA – Subset of the Java Virtual Machine
- A ‘Vanilla’ processor design



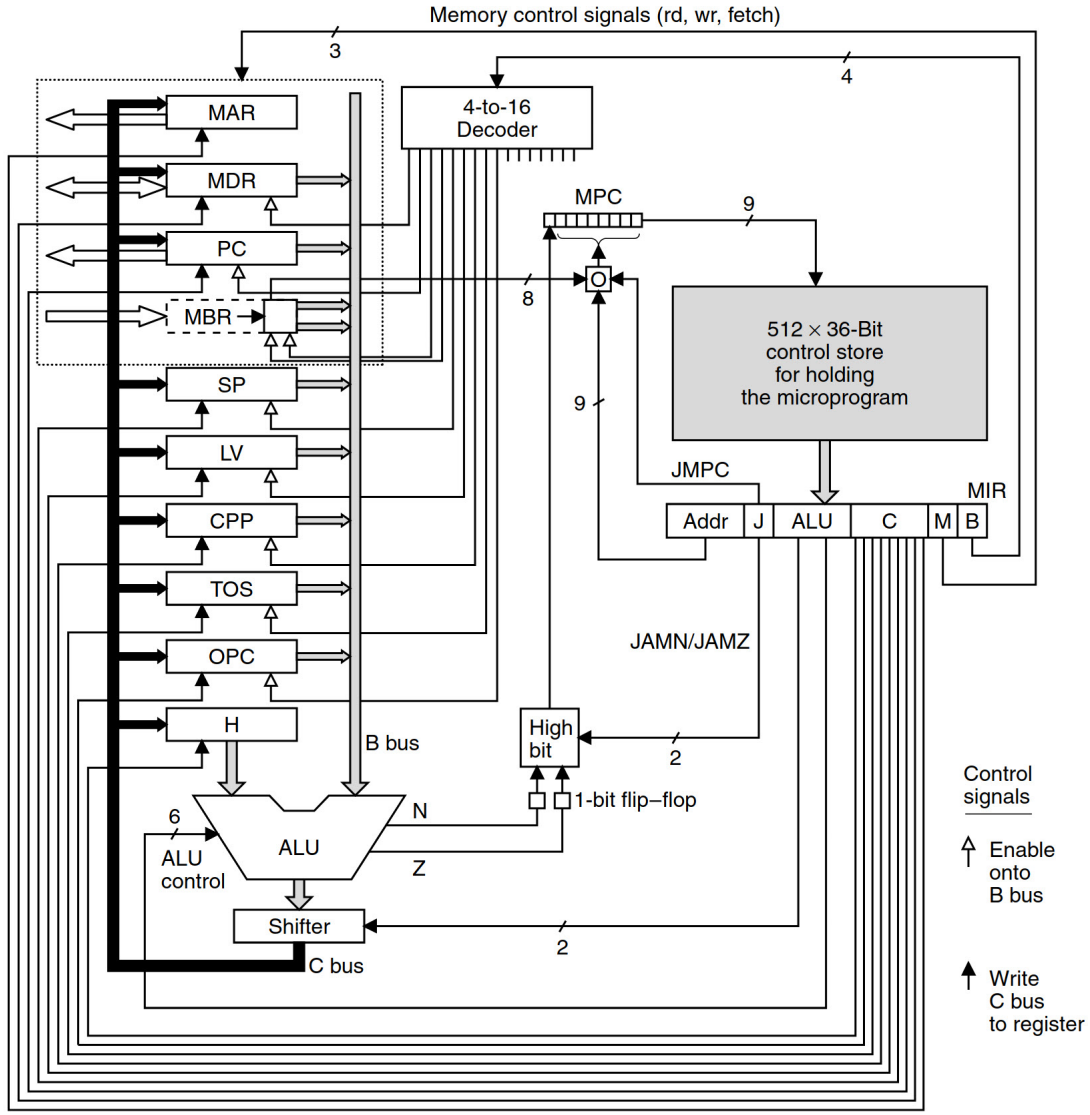
A 'Simple' Example

- Control Store is the most important part!
- Our ISA is defined by that unit
- 9 wires in -> 2**9 possible combinations, 2**9 (512) possible commands
- Each command drives 36 wires to control the chip
- Assembly/Machine Language is defined by the hardware



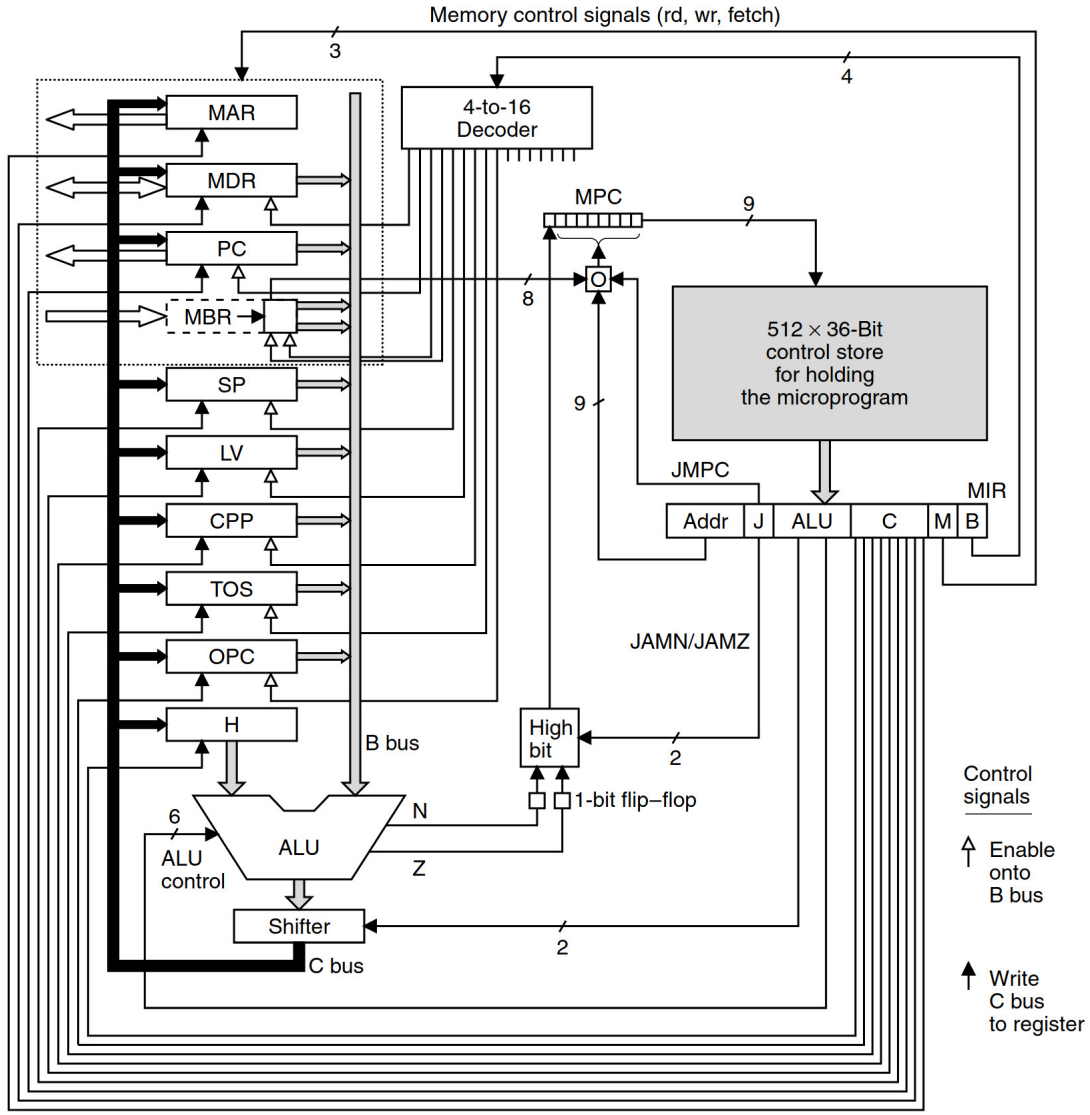
A 'Simple' Example

- ALU – Arithmetic & Logic Unit
  - Performs Math & Logic Operations
- MAR – H are the registers
- B + Decoder – Enables Register to load onto B Bus
- Z and N act similar to our condition codes, but in a much more limited/simple way
- C controls the C Bus, informing the destination register to receive its value



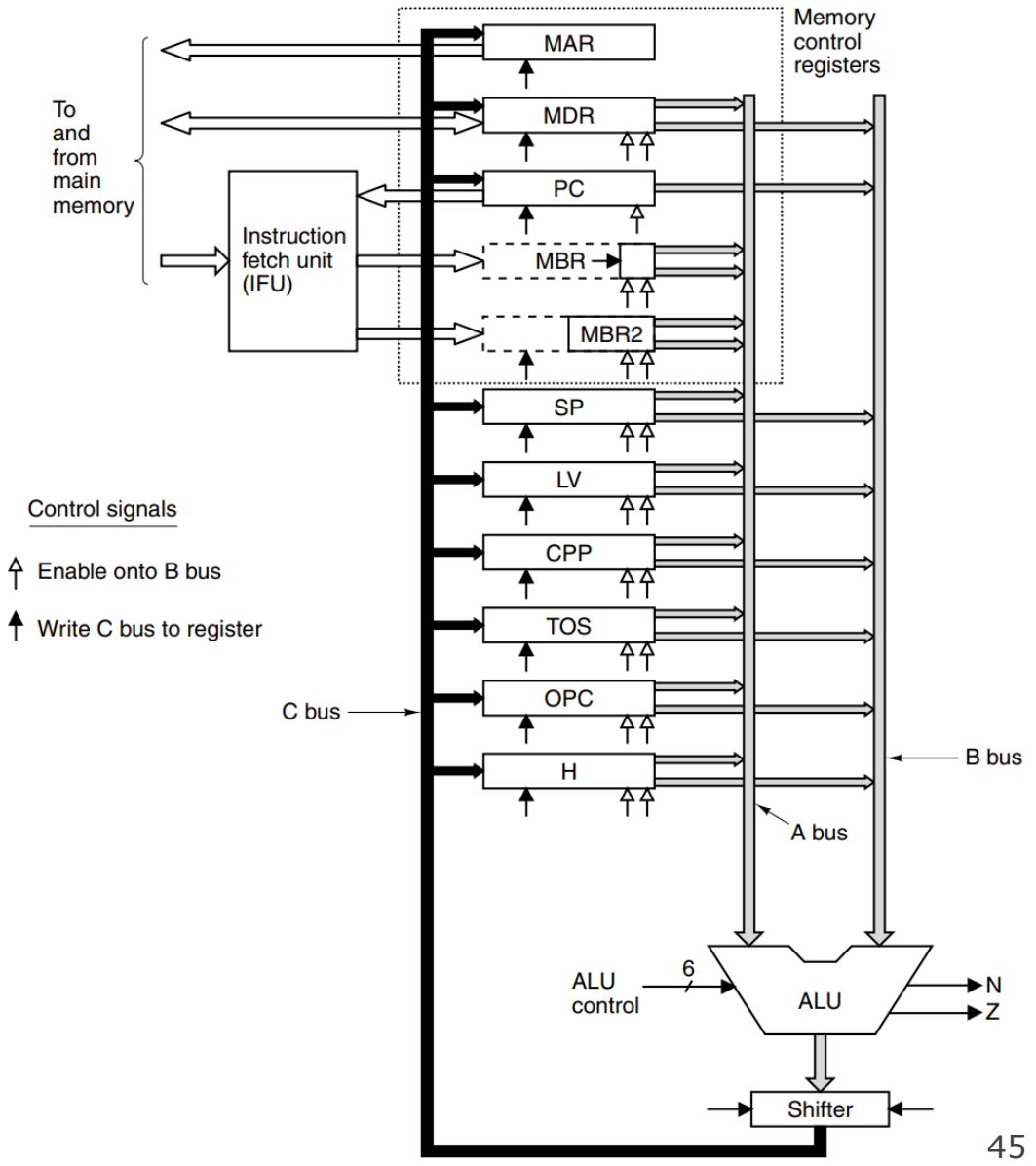
A ‘Simple’ Example

- Notice how the ALU is only able to take in the left operand from the H register
- All two operand ALU operations, would need to first load the left operand to H
- This would be an example of a hardware based constraint



Better Design Better Performance

- The MIC-2 Fixes this issue by adding another BUS improving the Datapath
- Design directly impacts the ISA that we can make available



Lecture Plan

- Into the Architecture!
- Peeking at Memory
- Architecture & The ISA
- **The heap so far**
- What is a heap allocator?
- Heap allocator requirements and goals
- Method 0: Bump Allocator

CS107 Topic 6: How do the core `malloc/realloc/free` memory-allocation operations work?

How do malloc/realloc/free work?

Pulling together all our CS107 topics this quarter:

- Testing
- Efficiency
- Bit-level manipulation
- Memory management
- Pointers
- Generics
- Assembly
- And more…

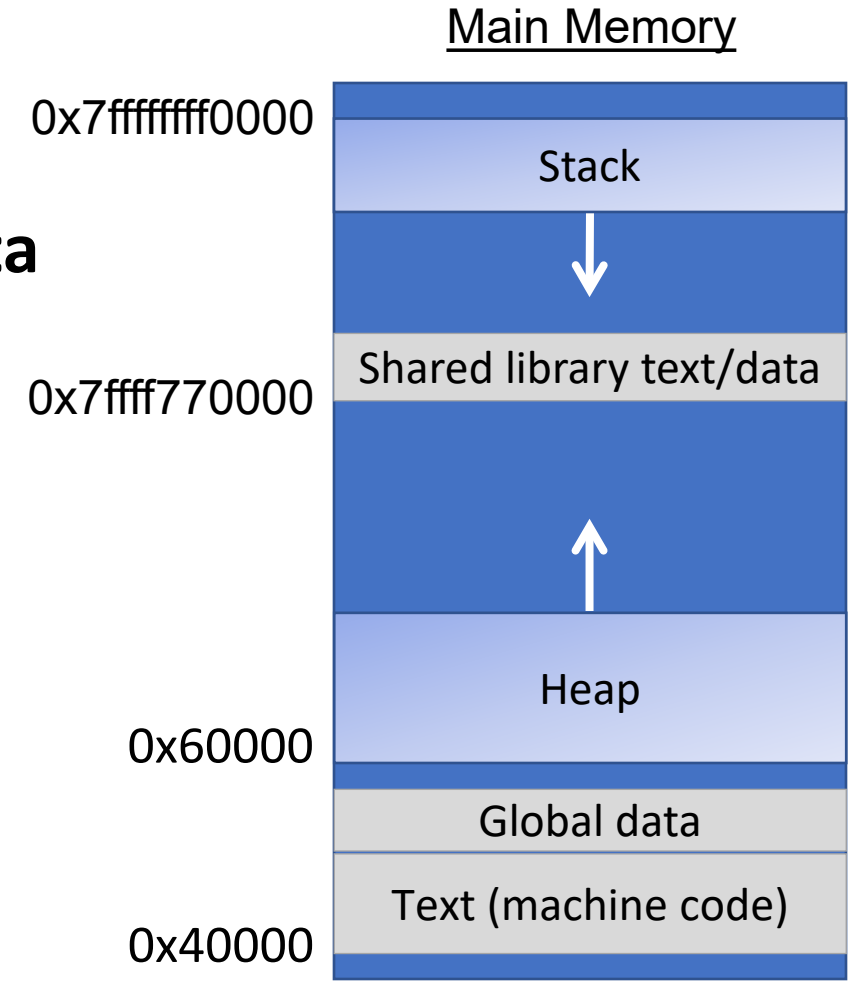
Learning Goals

- Learn the restrictions, goals and assumptions of a heap allocator
- Understand the conflicting goals of utilization and throughput
- Learn about different ways to implement a heap allocator

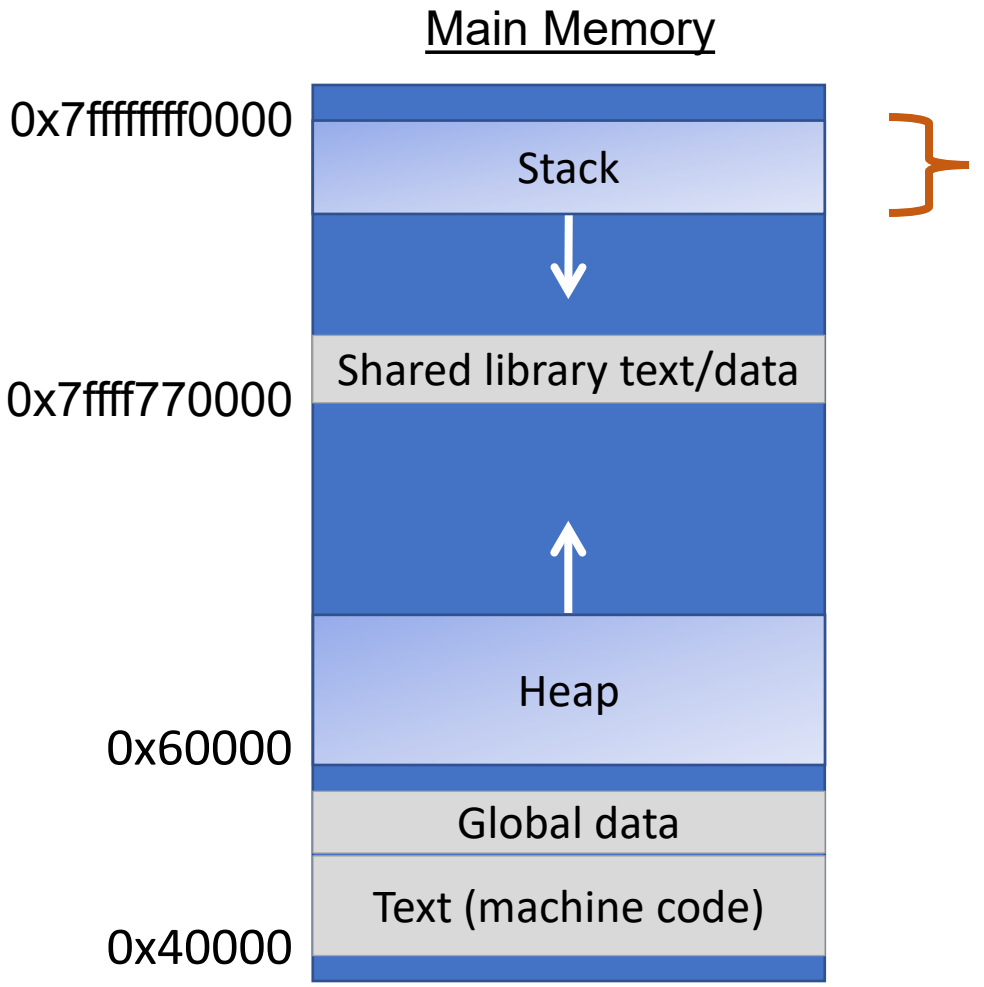
Running a program

- **Creates new process**
- **Sets up address space/segments**
- **Read executable file, load instructions, global data**
  Mapped from file into gray segments
- **Libraries loaded on demand**

- **Set up stack**
  Reserve stack segment, init %rsp, call main
- **malloc written in C, will init self on use**
  Asks OS for large memory region,
  parcels out to service requests



The Stack

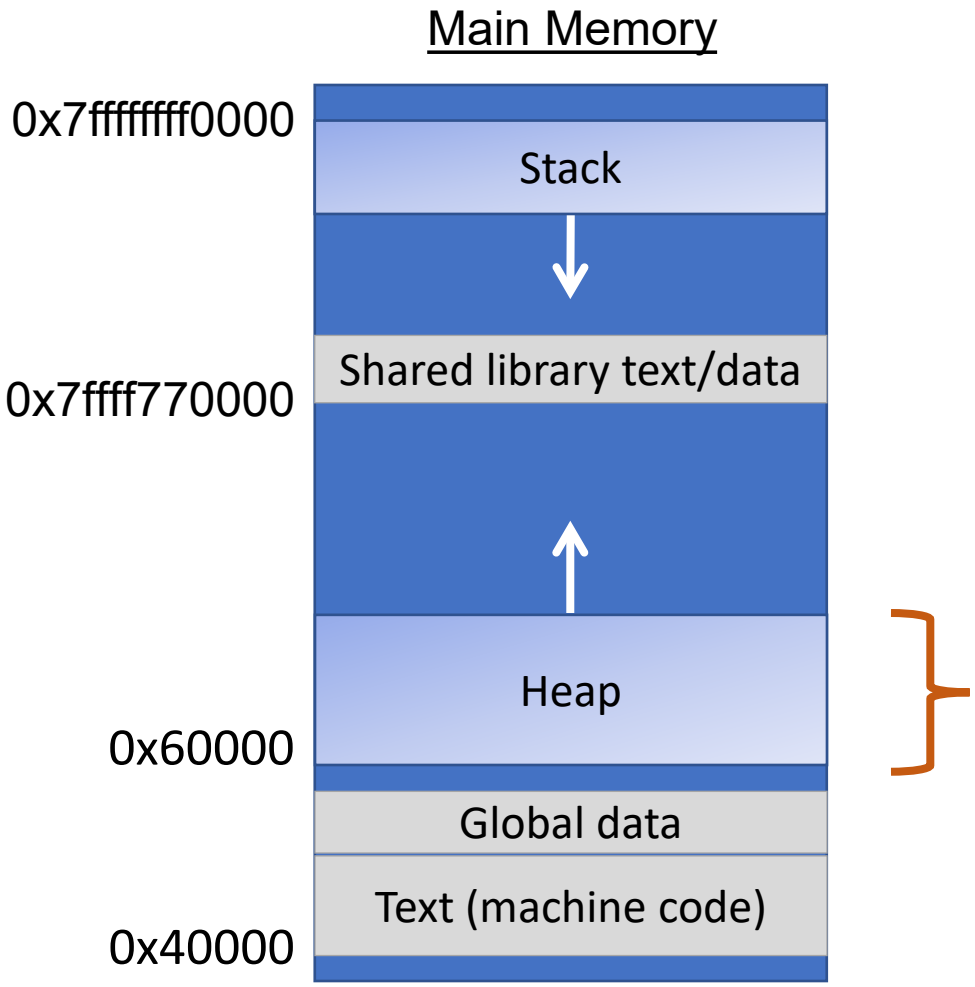


Stack memory "goes away" after function call ends.

Automatically managed at compile-time by gcc

From Assembly:
Stack management == moving %rsp around (pushq, popq, mov)

Today: The Heap



Heap memory persists until caller indicates it no longer needs it.

Managed by C standard library functions (malloc, realloc, free)

This lecture:
How does heap management work?

Lecture Plan

- Into the Architecture!
- Peeking at Memory
- Architecture & The ISA
- The heap so far
- **What is a heap allocator?**
- Heap allocator requirements and goals
- Method 0: Bump Allocator

Your role so far: Client

```
void *malloc(size_t size);
```

Returns a pointer to a block of heap memory of at least size bytes, or NULL if an error occurred.

```
void free(void *ptr);
```

Frees the heap-allocated block starting at the specified address.

```
void *realloc(void *ptr, size_t size);
```

Changes the size of the heap-allocated block starting at the specified address to be the new specified size. Returns the address of the new, larger allocated memory region.

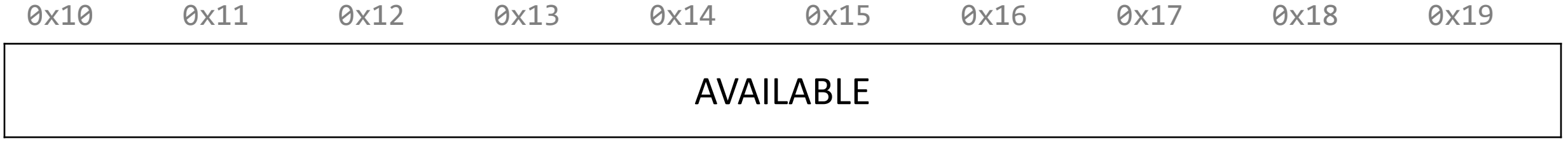
Your role now: Heap Hotel Concierge

(aka **Heap Allocator**)

http://screencrave.com/wp-content/uploads/2014/03/the-grand-budapest-hotel-anderson-image-2.jpg

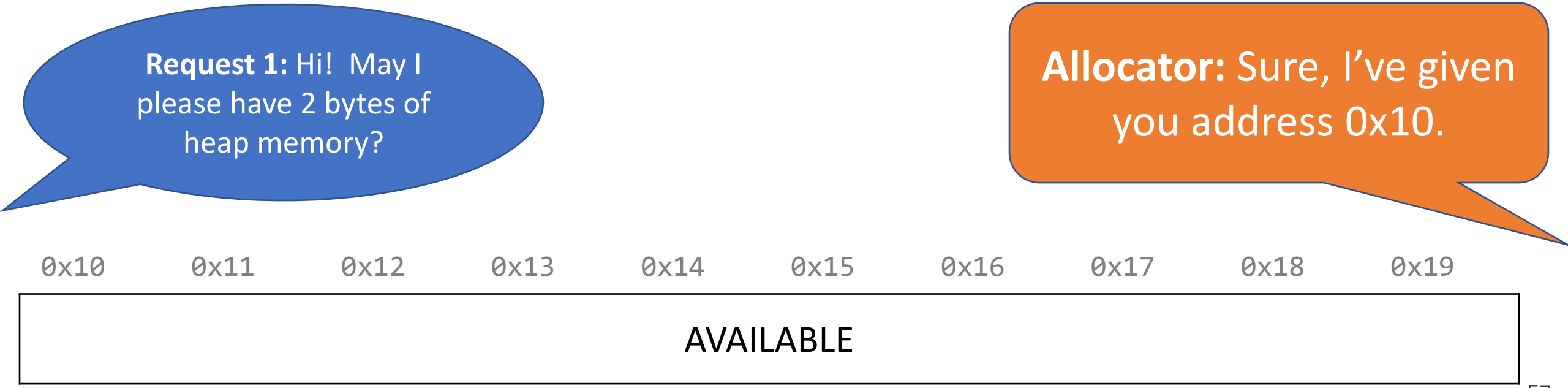
What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).



What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

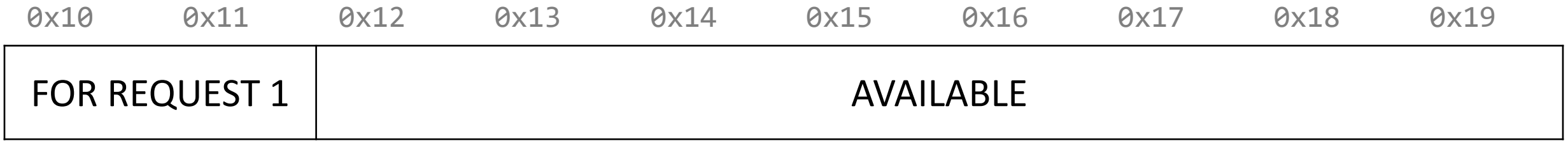


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 1: Hi! May I please have 2 bytes of heap memory?

Allocator: Sure, I've given you address 0x10.

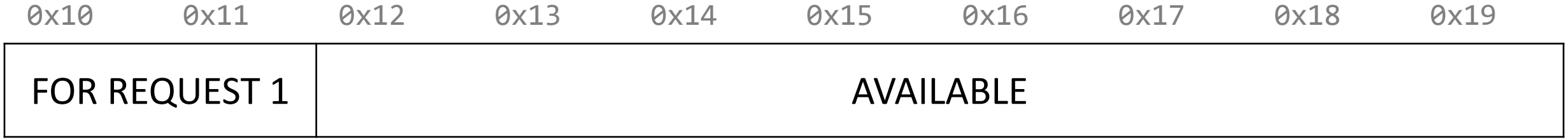


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 2: Howdy! May I please have 3 bytes of heap memory?

Allocator: Sure, I've given you address 0x12.

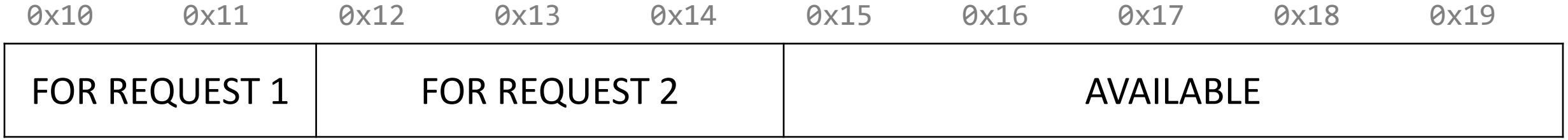


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 2: Howdy! May I please have 3 bytes of heap memory?

Allocator: Sure, I've given you address 0x12.

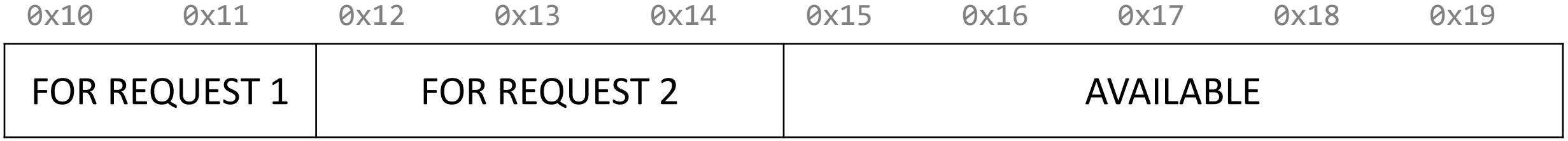


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 1: I'm done with the memory I requested. Thank you!

Allocator: Thanks. Have a good day!

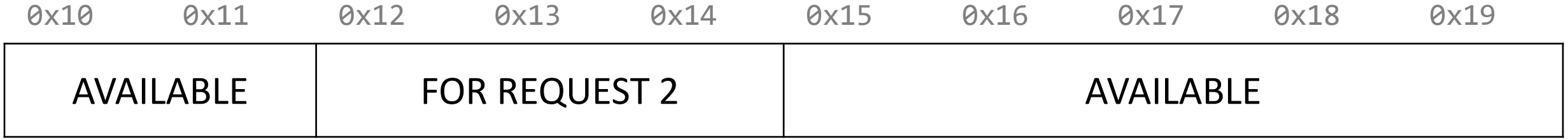


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 1: I'm done with the memory I requested. Thank you!

Allocator: Thanks. Have a good day!

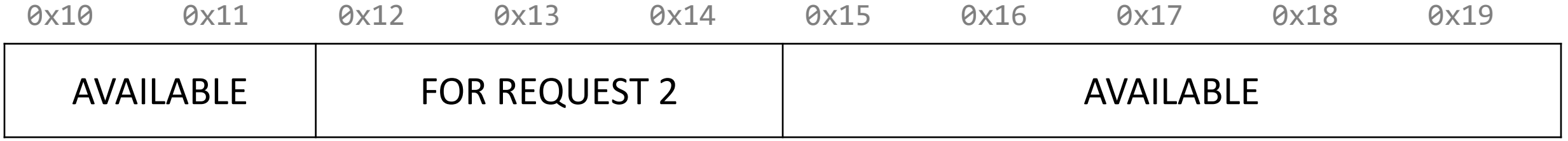


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 3: Hello there! I'd like to request 2 bytes of heap memory, please.

Allocator: Sure thing. I've given you address 0x10.

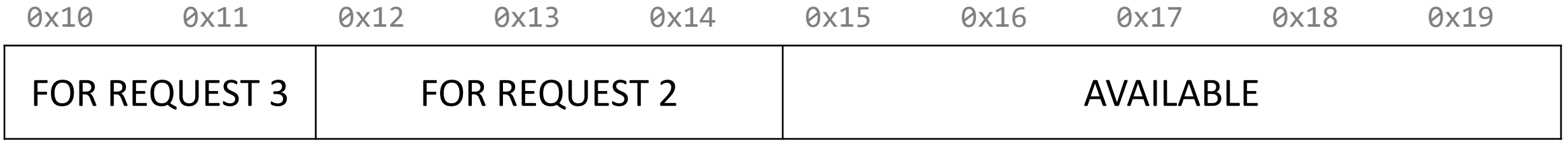


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 3: Hello there! I'd like to request 2 bytes of heap memory, please.

Allocator: Sure thing. I've given you address 0x10.

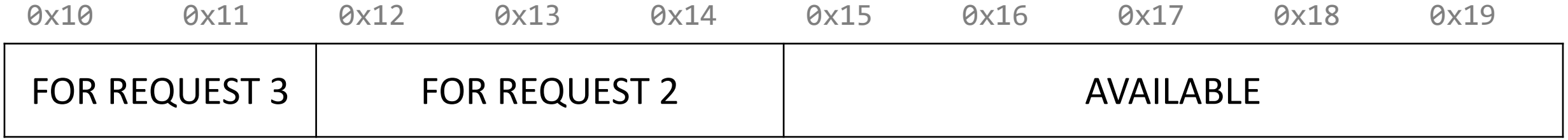


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 3: Hi again! I'd like to request the region of memory at 0x10 be reallocated to 4 bytes.

Allocator: Sure thing. I've given you address 0x15.

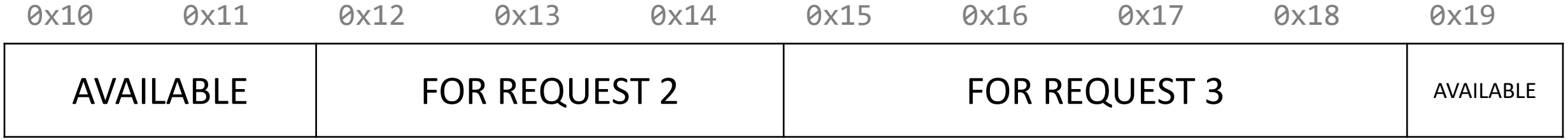


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 3: Hi again! I'd like to request the region of memory at 0x10 be reallocated to 4 bytes.

Allocator: Sure thing. I've given you address 0x15.



Lecture Plan

- Into the Architecture!
- Peeking at Memory
- Architecture & The ISA
- The heap so far
- What is a heap allocator?
- **Heap allocator requirements and goals**
- Method 0: Bump Allocator

Heap Allocator Functions

```
void *malloc(size_t size);

void free(void *ptr);

void *realloc(void *ptr, size_t size);
```

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. Keep track of which memory is allocated and which is available
3. Decide which memory to provide to fulfill an allocation request
4. Immediately respond to requests without delay
5. Return addresses that are 8-byte-aligned (must be multiples of 8).

Heap Allocator Requirements

A heap allocator must…

1. **Handle arbitrary request sequences of allocations and frees**
2. Keep track of which memory is allocated and which is available
3. Decide which memory to provide to fulfill an allocation request
4. Immediately respond to requests without delay
5. Return addresses that are 8-byte-aligned (must be multiples of 8).

A heap allocator cannot assume anything about the order of allocation and free requests, or even that every allocation request is accompanied by a matching free request.

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. **Keep track of which memory is allocated and which is available**
3. Decide which memory to provide to fulfill an allocation request
4. Immediately respond to requests without delay
5. Return addresses that are 8-byte-aligned (must be multiples of 8).

A heap allocator marks memory regions as **allocated** or **available**. It must remember which is which to properly provide memory to clients.

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. Keep track of which memory is allocated and which is available
3. **Decide which memory to provide to fulfill an allocation request**
4. Immediately respond to requests without delay
5. Return addresses that are 8-byte-aligned (must be multiples of 8).

A heap allocator may have options for which memory to use to fulfill an allocation request. It must decide this based on a variety of factors.

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. Keep track of which memory is allocated and which is available
3. Decide which memory to provide to fulfill an allocation request
4. **Immediately respond to requests without delay**
5. Return addresses that are 8-byte-aligned (must be multiples of 8).

A heap allocator must respond immediately to allocation requests and should not e.g. prioritize or reorder certain requests to improve performance.

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. Keep track of which memory is allocated and which is available
3. Decide which memory to provide to fulfill an allocation request
4. Immediately respond to requests without delay
5. **Return addresses that are 8-byte-aligned (must be multiples of 8).**

Heap Allocator Goals

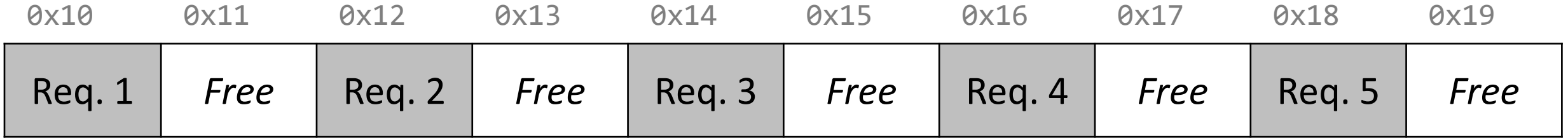
- Goal 1: Maximize **throughput**, or the number of requests completed per unit time. This means minimizing the average time to satisfy a request.
- Goal 2: Maximize memory **utilization**, or how efficiently we make use of the limited heap memory to satisfy requests.

Utilization

- The primary cause of poor utilization is **fragmentation**. **Fragmentation** occurs when otherwise unused memory is not available to satisfy allocation requests.
  - **External Fragmentation (this example)**: no single space is large enough to satisfy a request, even though enough aggregate free memory is available
  - **Internal Fragmentation**: space allocated for a block is larger than needed (more later).
- In general: we want the largest address used to be as low as possible.

Request 6: Hi! May I please have 4 bytes of heap memory?

Allocator: I'm sorry, I don't have a 4 byte block available…



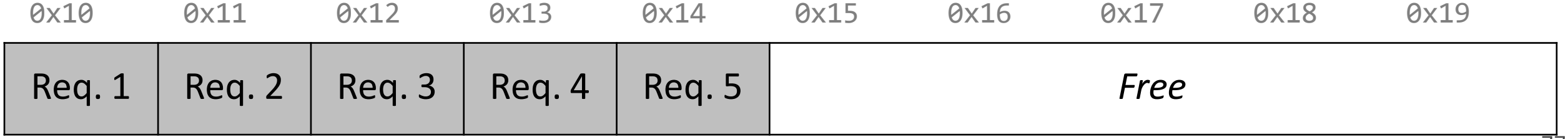
Utilization

Question: what if we shifted these blocks down to make more space? Can we do this?

A. YES, great idea!

B. YES, it can be done, but not a good idea for some reason (e.g. not efficient use of time)

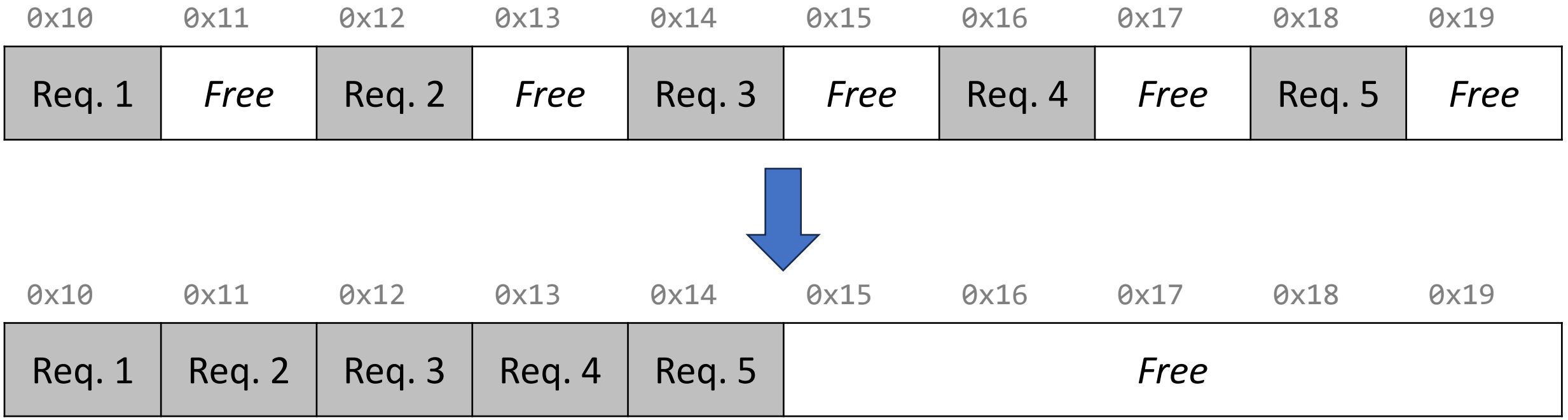
C. NO, it can't be done!



Utilization

Question: Can we / should we shift these blocks down to make more space?

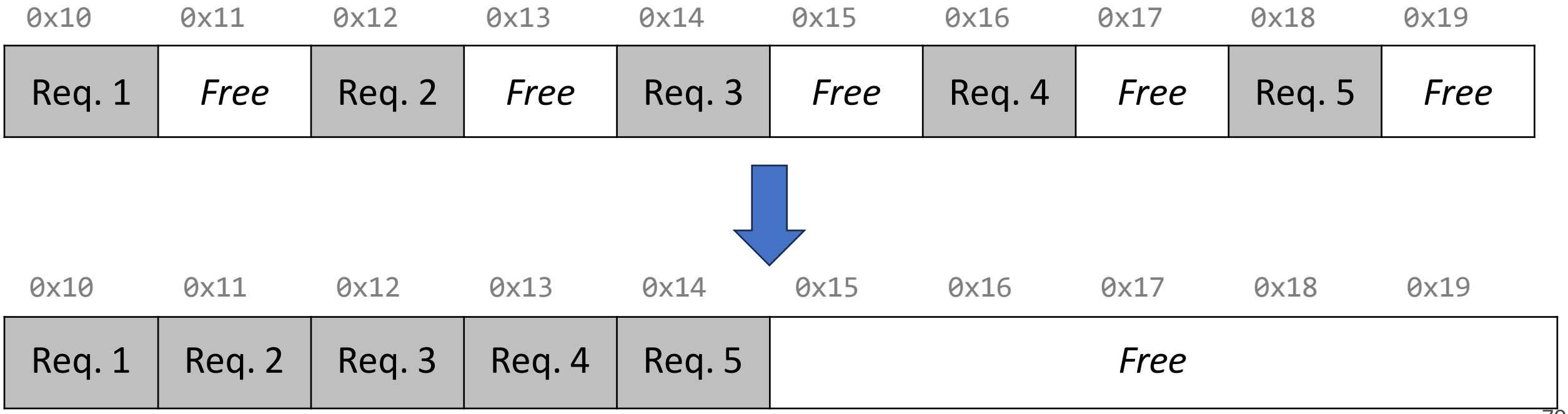
- YES, good idea!
- YES, but not a good idea for some reason
- NO, it can't be done!



Utilization

Question: what if we shifted these blocks down to make more space? Can we do this?

- **No -** we have already guaranteed these addresses to the client. We cannot move allocated memory around, since this will mean the client will now have incorrect pointers to their memory!



Fragmentation

- **Internal Fragmentation**: an allocated block is larger than what is needed (e.g. due to minimum block size)
- **External Fragmentation**: no single block is large enough to satisfy an allocation request, even though enough aggregate free memory is available

Heap Allocator Goals

- Goal 1: Maximize **throughput**, or the number of requests completed per unit time. This means minimizing the average time to satisfy a request.
- Goal 2: Maximize memory **utilization**, or how efficiently we make use of the limited heap memory to satisfy requests.

These are seemingly conflicting goals – for instance, it may take longer to better plan out heap memory use for each request. **Heap allocators must find an appropriate balance between these two goals!**

Heap Allocator Goals

- Goal 1: Maximize **throughput**, or the number of requests completed per unit time. This means minimizing the average time to satisfy a request.
- Goal 2: Maximize memory **utilization**, or how efficiently we make use of the limited heap memory to satisfy requests.

Other desirable goals:

Locality ("similar" blocks allocated close in space)

Robust (handle client errors)

Ease of implementation/maintenance

Lecture Plan

- Into the Architecture!
- Peeking at Memory
- Architecture & The ISA
- The heap so far
- What is a heap allocator?
- Heap allocator requirements and goals
- **Method 0: Bump Allocator**

Bump Allocator

Let's say we want to entirely prioritize throughput, and do not care about utilization at all. This means we do not care about reusing memory. How could we do this?

A **bump allocator** is a heap allocator design that simply allocates the next available memory address upon an allocate request and does nothing on a free request.

Bump Allocator Performance

1. Utilization

Never reuses memory

2. Throughput

Ultra fast, short routines

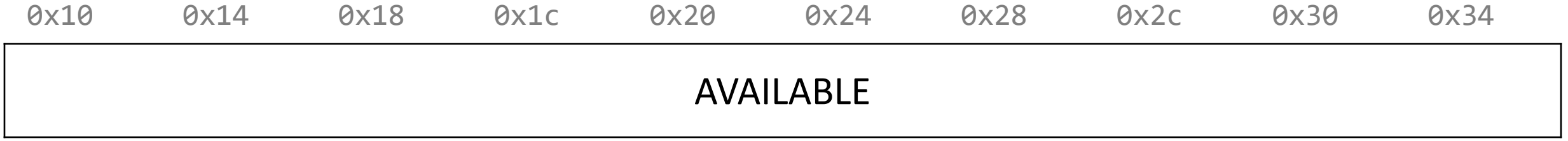
Bump Allocator

A **bump allocator** is a heap allocator design that simply allocates the next available memory address upon an allocate request and does nothing on a free request.

- Throughput: each **malloc** and **free** execute only a handful of instructions:
  - It is easy to find the next location to use
  - Free does nothing!
- Utilization: we use each memory block at most once. No freeing at all, so no memory is ever reused. ☹
- We provide a bump allocator implementation as part of assign6 as a code reading exercise.

Bump Allocator

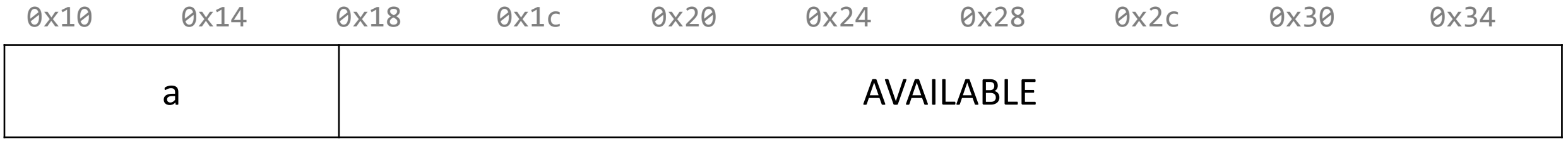
```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

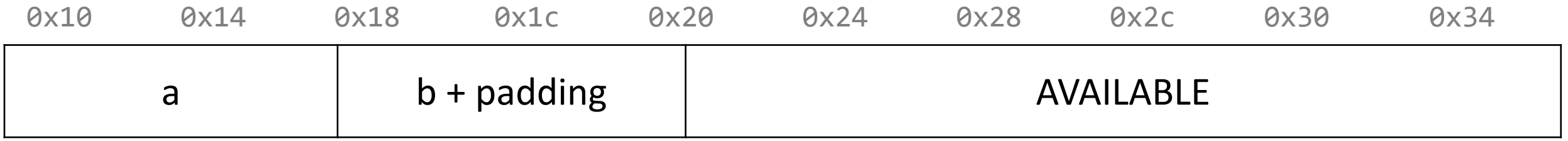
Variable	Value
a	0x10



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

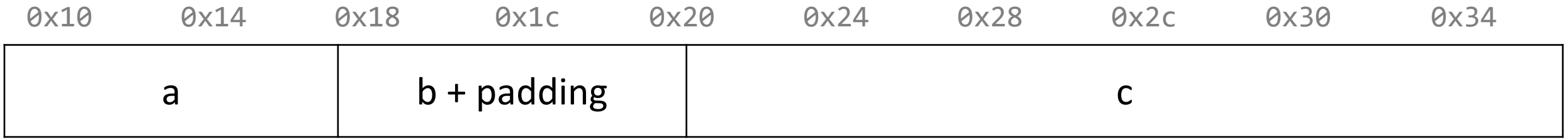
Variable	Value
a	0x10
b	0x18



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

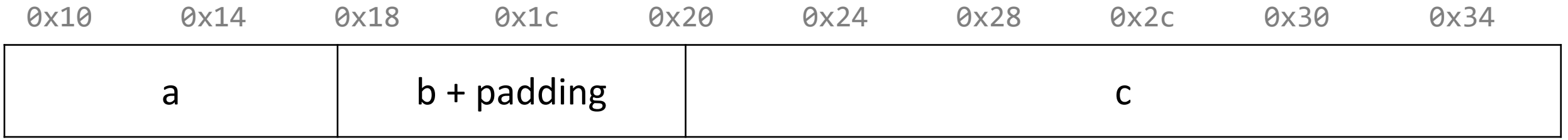
Variable	Value
a	0x10
b	0x18
c	0x20



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

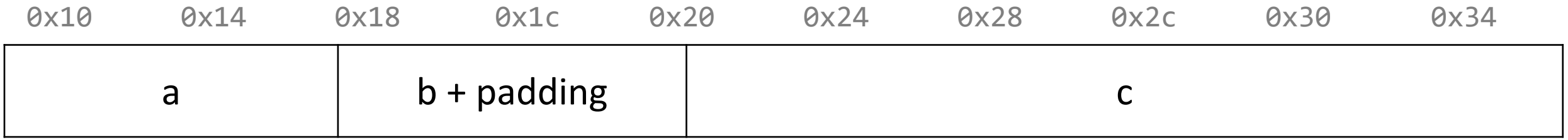
Variable	Value
a	0x10
b	0x18
c	0x20



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

Variable	Value
a	0x10
b	0x18
c	0x20
d	NULL



Summary: Bump Allocator

- A bump allocator is an extreme heap allocator – it optimizes only for **throughput**, not **utilization**.
- Better allocators strike a more reasonable balance. How can we do this?

Questions to consider:

1. How do we keep track of free blocks?
2. How do we choose an appropriate free block in which to place a newly allocated block?
3. After we place a newly allocated block in some free block, what do we do with the remainder of the free block?
4. What do we do with a block that has just been freed?