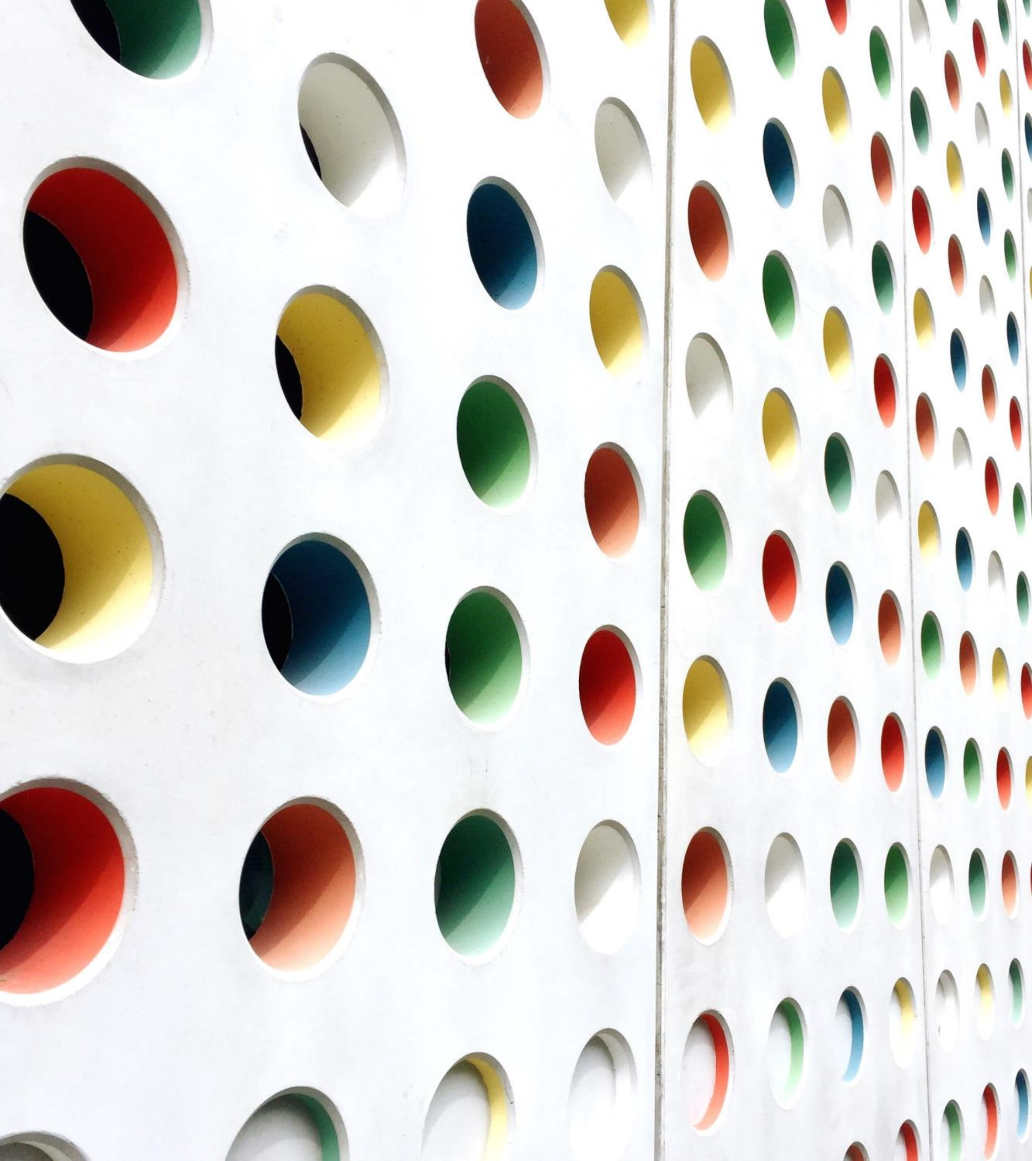


Memory Management and File IO

GUY DAVIDSON

Summary

The Computer

Memory management

Offline data

File IO

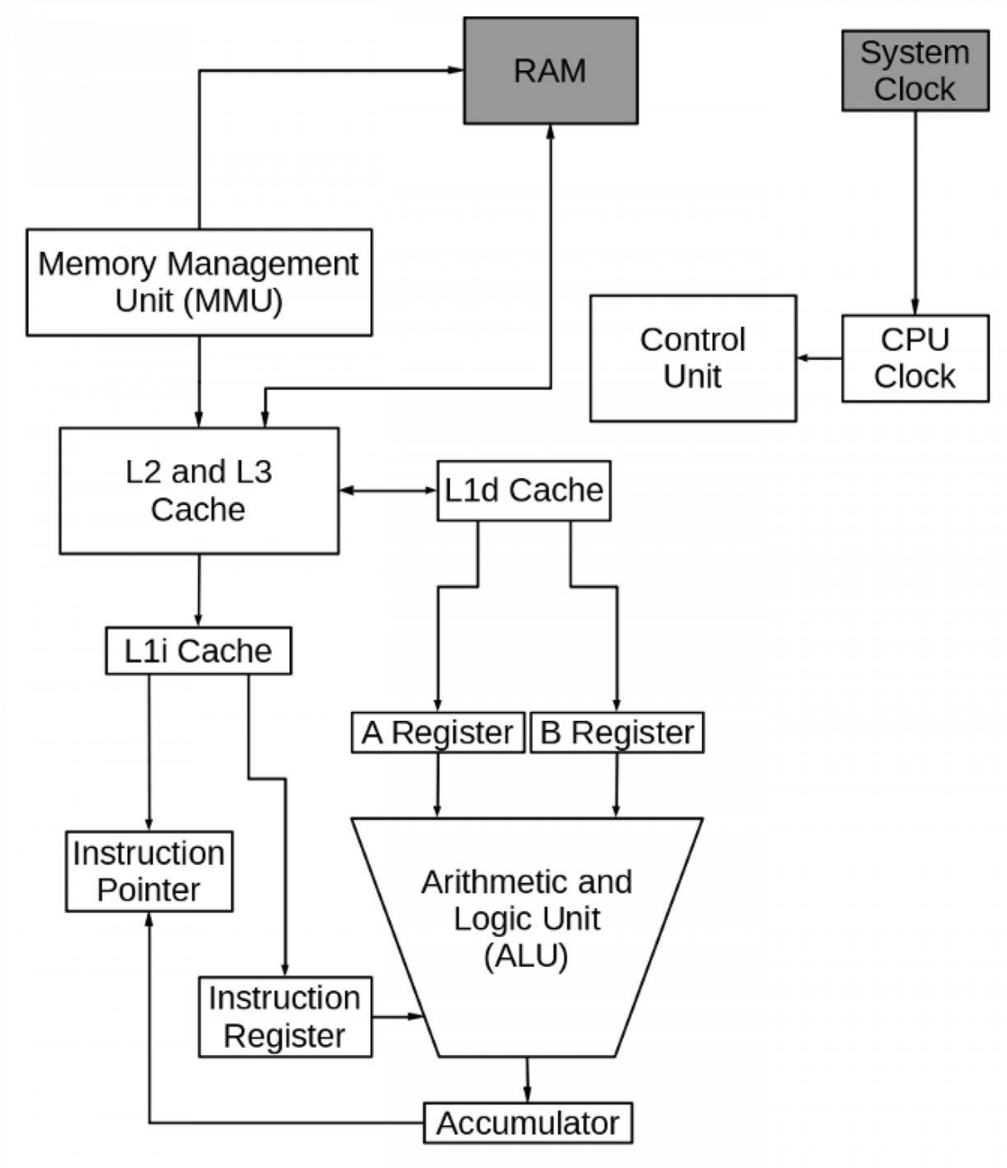
Computer Hardware: CPU

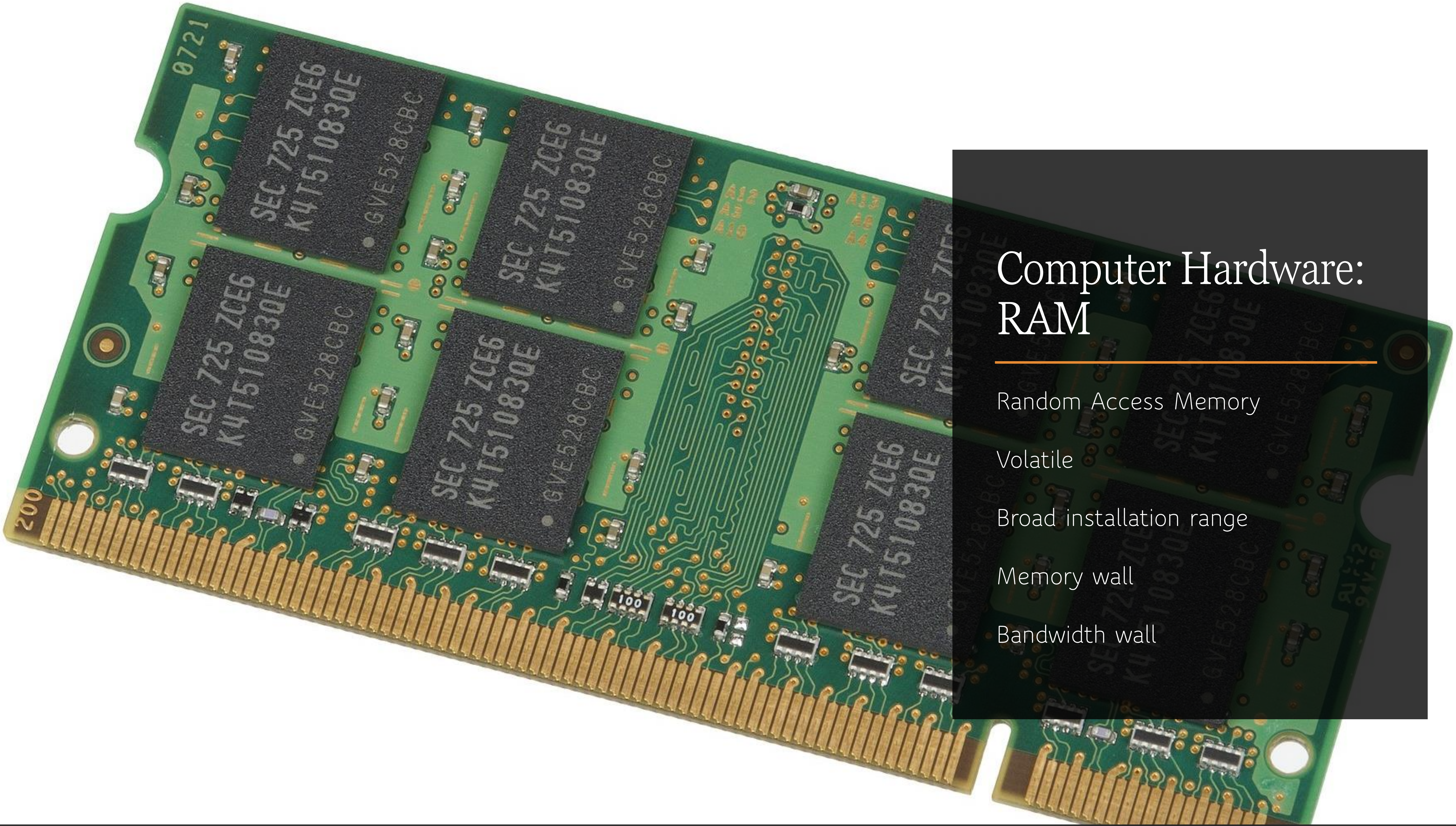
ALU

Instruction pointer

Cache

MMU





Computer Hardware: RAM

- Random Access Memory
- Volatile
- Broad installation range
- Memory wall
- Bandwidth wall

Computer Hardware: RAM

First of all, as chip geometries shrink and clock frequencies rise, the transistor leakage current increases, leading to excess power consumption and heat

Secondly, the advantages of higher clock speeds are in part negated by memory latency, since memory access times have not been able to keep pace with increasing clock frequencies

Third, for certain applications, traditional serial architectures are becoming less efficient as processors get faster (due to the so-called Von Neumann bottleneck), further undercutting any gains that frequency increases might otherwise buy

In addition, partly due to limitations in the means of producing inductance within solid state devices, resistance-capacitance (RC) delays in signal transmission are growing as feature sizes shrink, imposing an additional bottleneck that frequency increases don't address

Computer Hardware: OS and processes

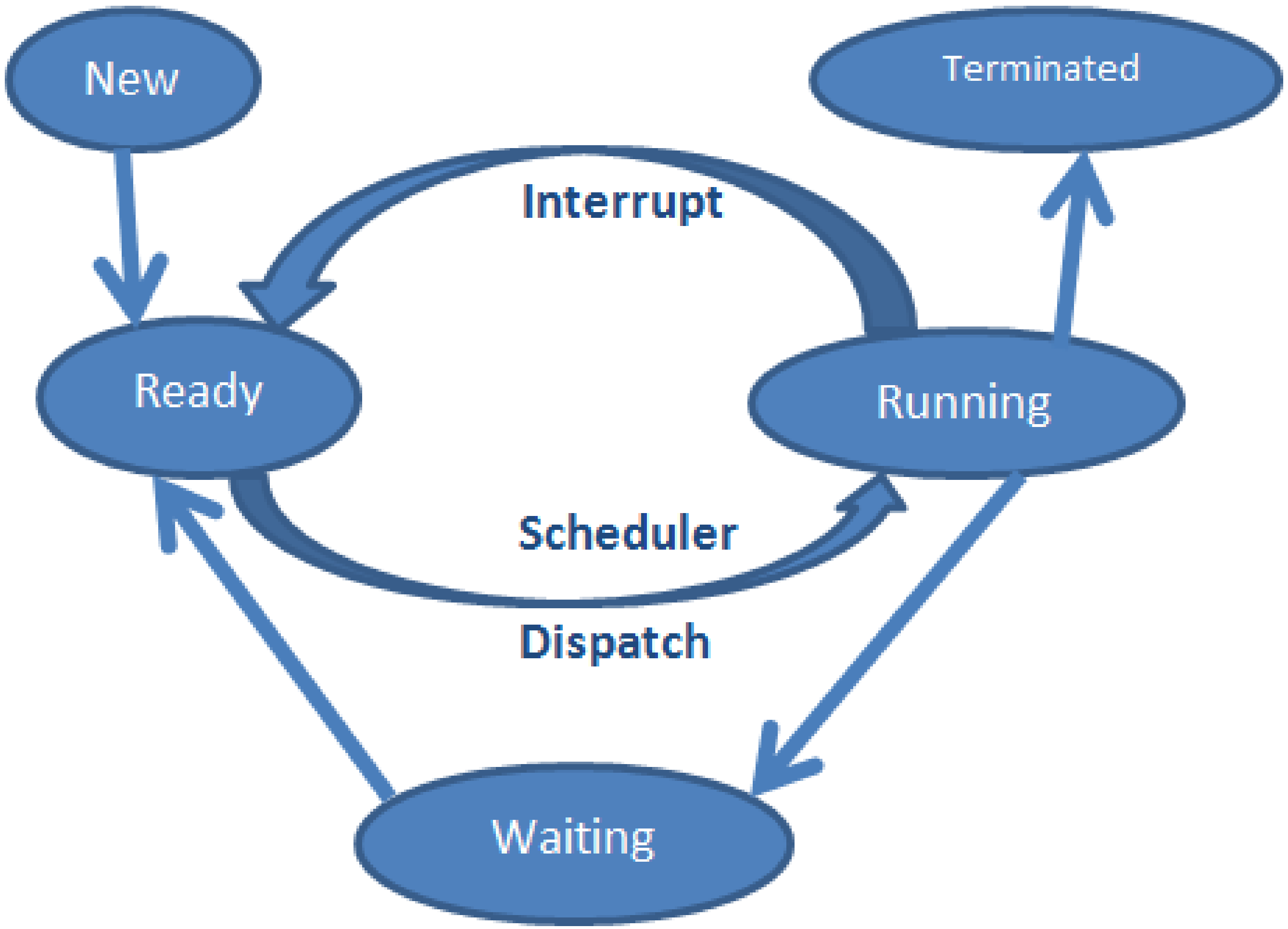
BIOS

Bootstrap

Thread

Process

Shell



Computer Hardware: Address space

Discrete addresses corresponding to logical or physical entities

RAM

Hierarchical

IP

Mapping and translation

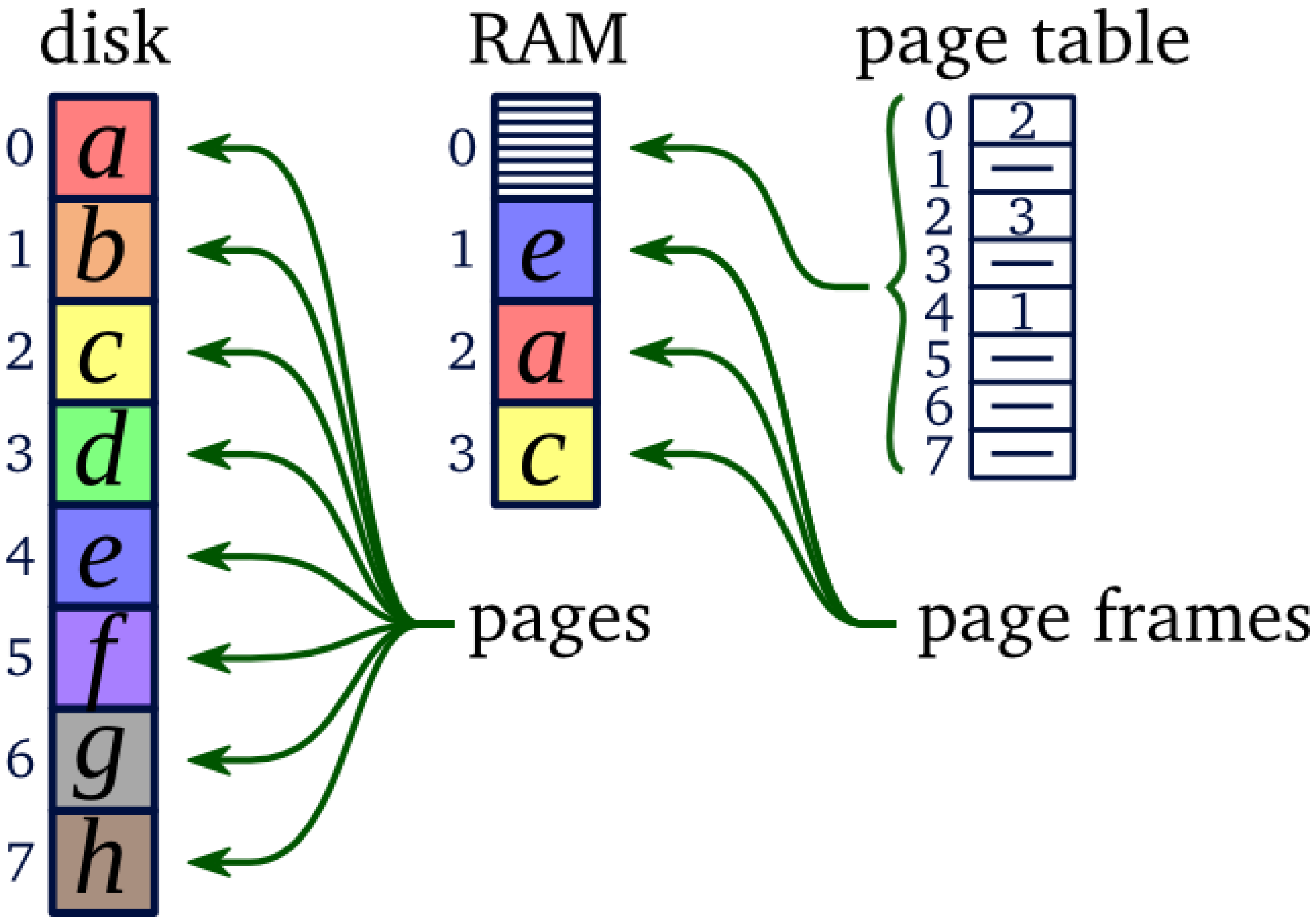
File system

DNS

Virtual memory

Computer Hardware: Pages

- Fixed length and contiguous
- Processor architecture
- Translation Lookaside Buffer
- Fragmentation



Computer Hardware: Swapping

2^{64} = 16 exabytes

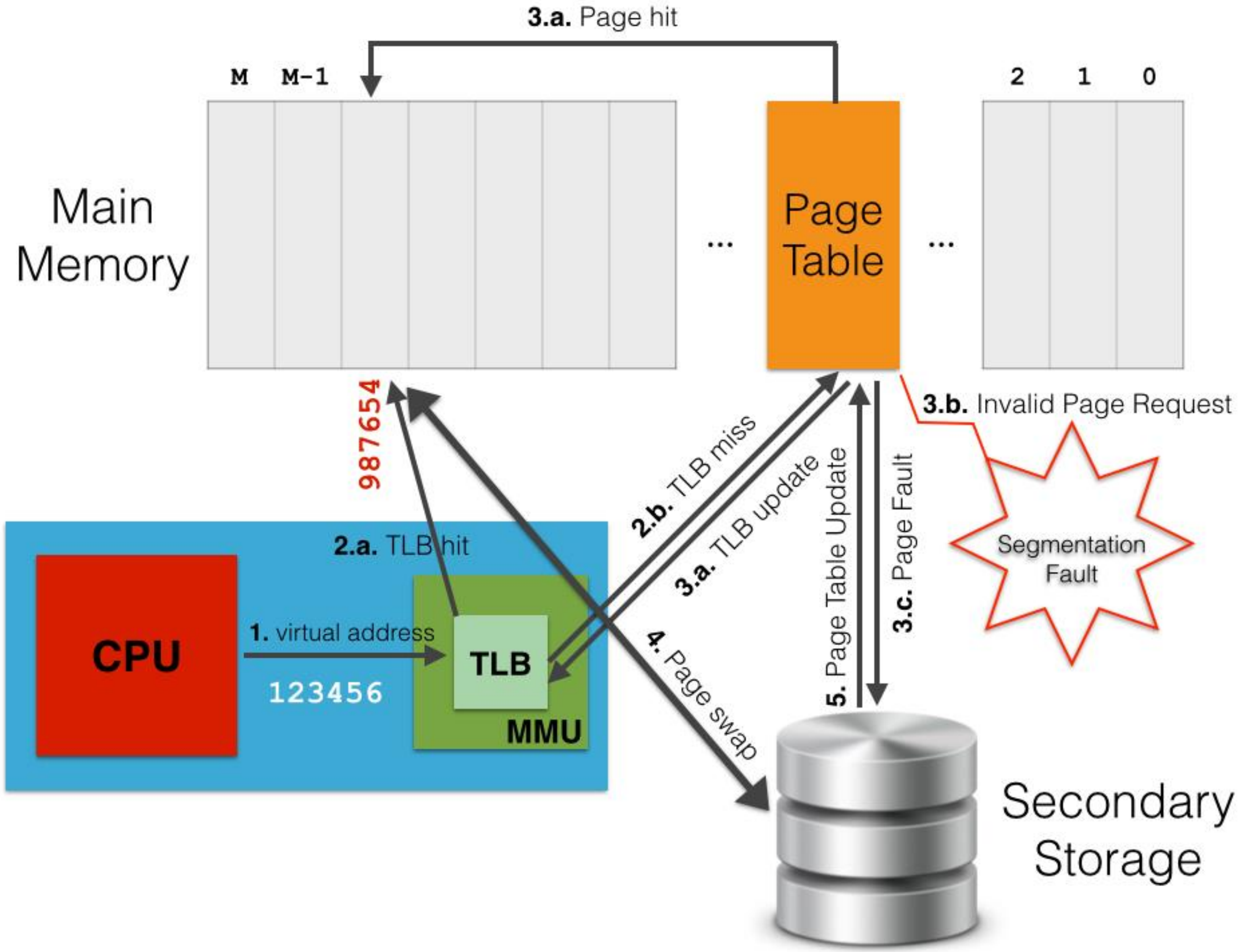
Address request

TLB hit or miss

Update

Page hit or fault

Swap to or from storage



Computer Hardware: Hibernating

Offline storage = non-volatile memory

850W

600W

Long startup times

Move RAM to non-volatile memory

BIOS cooperation

Multiple power-down states

hiberfil.sys

Computer Hardware: Summary

CPU

RAM

OS and processes

Address space

Pages

Swapping

Hibernating

Memory Management: The new operator

```
void* malloc(size_t);

void init(float, int, int);

new MyType(a, b, c);
```

Memory Management: operator new

```
<new>

void* operator new(std::size_t);

void* malloc(size_t);
```

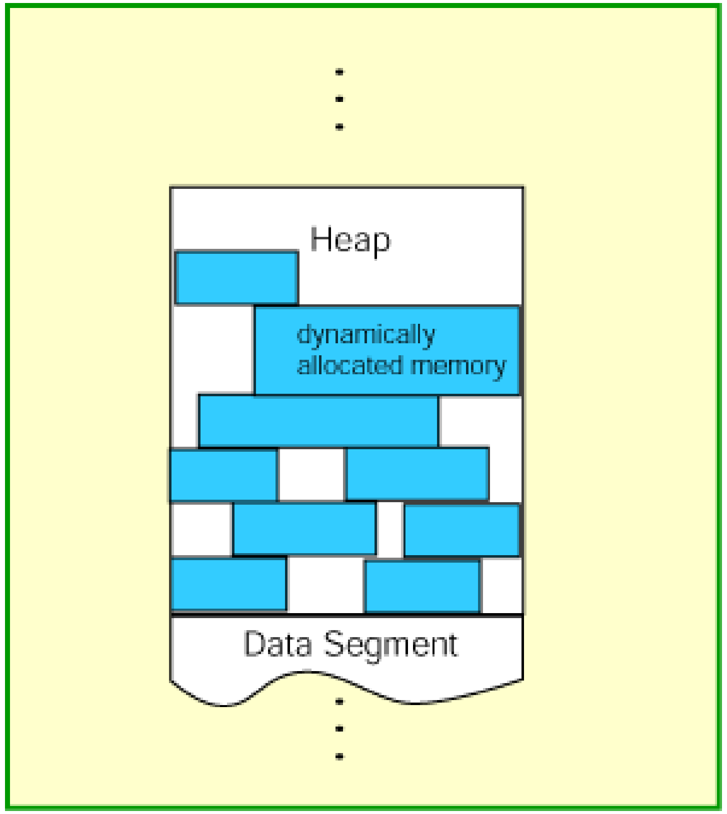
Memory Management: Fragmentation

Static - simple

Automatic - simple

Dynamic – not so simple

The Tetris problem



Memory Management: Strategies

Stack allocator

Destruction in reverse order of construction

UI system

Front end => graphics settings => resolution picker

Like a regular stack

Memory Management: Strategies

Frame allocator

Game loop

Mayfly fragmentation

Memory Management: Strategies

Pools

Object size frequency

Easy deallocation

Minimise page faults

Memory Management: Strategies

Stack allocators

Small String Optimisation (SSO)

Small Buffer Optimisation (SBO)

Automatic storage duration, NOT dynamic storage duration

Memory Management: Strategies

Double-ended allocators

Object pairs

Terrible fragmentation

Different sizes at either end

Memory Management: The delete operator

```
void deinit(void*);

void free(void*);

delete p;
```

Memory Management: operator delete

```
void operator delete(void*);

void free(void*);
```

Memory Management: Tracking

How much and when

Database problem

Stack trace

Timestamp

Page count

Memory Management: Raising the Abstraction Level

```
template <class T> struct allocator;
```

```
<memory_resource>
```

Memory Management: Raising the Abstraction Level

template <class T> struct allocator;

<memory_resource>

std::unique_ptr

Memory Management: Raising the Abstraction Level

template <class T> struct allocator;

<memory_resource>

std::unique_ptr

std::shared_ptr

Memory Management: Summary

The new operator and operator new

Fragmentation

Allocation strategies

The delete operator and operator delete

Tracking

std::allocator and std::unique_ptr

std::shared_ptr

Offline Data: Process Size

Remote from the executable

Available while powered down

1KB

512KB + 720KB

1MB + 44MB

64GB + 6TB

Offline Data: DLLs

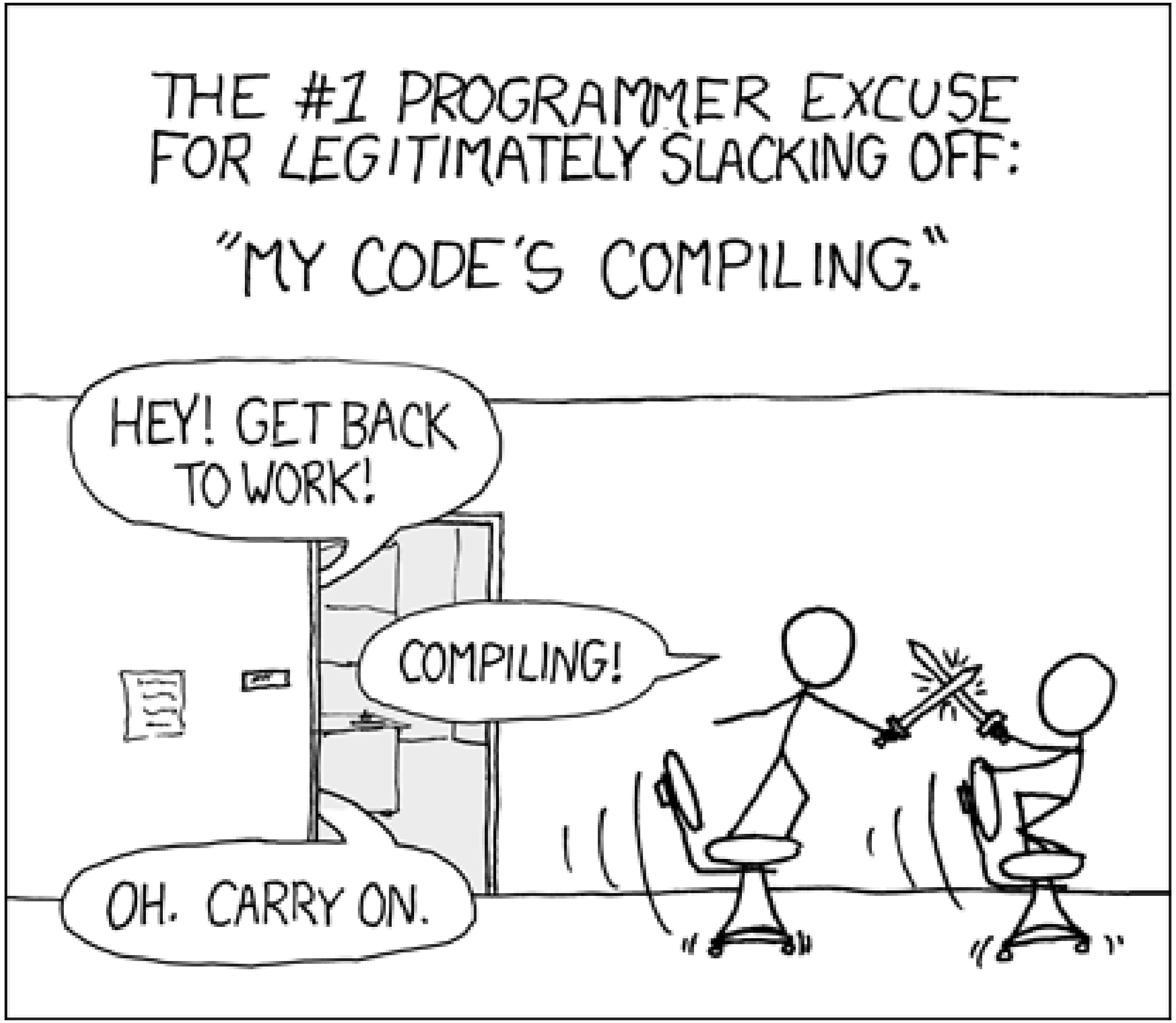
Executable offline data

No recompilation required on update

Driver files

Export table of function pointers

No concerns about static data



Offline Data: Import Library vs LoadLibrary

.dll + .lib

Hotload

Avoid long iteration

Offline Data: Process Data

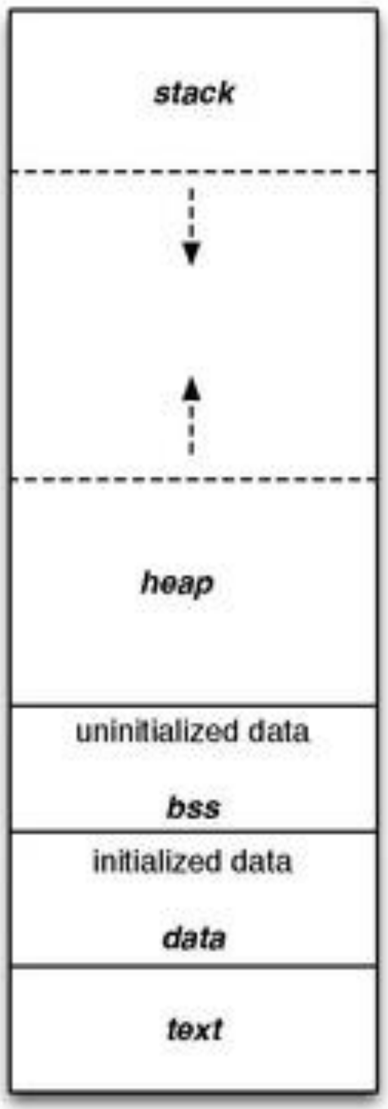
.text

char[] const

.data

.bss

https://wg21.link/P1040 std::embed



Offline Data: Off-RAM data

Why have offline data?

Modularity

Offline Data: Remote data

Separate from power and motherboard

Security

Offline Data: L0 Cache => L7 Cache

L0: Registers

386: cache on motherboard

486: L1 8Kb on die, L2 on motherboard

Pentium: L1 and L2 on die

AMD Socket 7: L1 and L2 on die, L3 on motherboard

Multicore: L1, L2 and L3 on die

Haswell: L1, L2, L3 and L4 on die

Offline Data: Summary

Process size

DLLs

Import library versus LoadLibrary

Process data

Off-RAM data

Remote data

L0 cache => L7 cache

File IO: fstream

```
auto file = std::fstream(“my_file.bin”, ios_base::in);

std::basic_istream::get(char_type&);

std::basic_istream::getline(char_type*, std::streamsize);

std::basic_istream::operator >>(int&);

std::basic_istream::operator >>(
          basic_istream& (*func)(basic_istream&));

file >> my_type >> hex >> ptr >> dec >> id;
```

File IO: fopen, fread, fclose

```
std::FILE* file = fopen(“my_file.bin”, “r”);

std::ios_base::in -> “r”

std::size_t fread(void* buffer, std::size_t size,
             std::size_t count, std::FILE* stream);

int fclose(std::FILE*);
```

File IO: CreateFile, ReadFile, CloseHandle

```
HANDLE CreateFile(
  LPCSTR                lpFileName,
  DWORD                 dwDesiredAccess,
  DWORD                 dwShareMode,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  DWORD                 dwCreationDisposition,
  DWORD                 dwFlagsAndAttributes,
  HANDLE                hTemplateFile
);
```

File IO: CreateFile, ReadFile, CloseHandle

```
BOOL ReadFile(
  HANDLE       hFile,
  LPVOID       lpBuffer,
  DWORD        nNumberOfBytesToRead,
  LPDWORD      lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
```

File IO: CreateFile, ReadFile, CloseHandle

```
BOOL CloseHandle(
  HANDLE hObject
);
```

File IO: Memory Mapping

Address space ⇔ offline storage

File = memory

CreateFileMapping, MapViewOfFile

Multiple of the page size

Page boundaries

Hard to instrument

File IO: Asynchronous IO

ReadFile

Event handle

IO completion port

Function pointer (ReadFileEx)

File IO: Buses

SATA

PCIe

USB

File IO: Direct Memory Access (DMA)

Standard

Bus mastering

Cache coherency

File IO: Summary

fstream

fopen, fread, fclose

CreateFile, ReadFile, CloseHandle

Memory mapping

Asynchronous I/O

Buses

DMA

Summary

The Computer

Memory management

Offline data

File IO

Title Lorem Ipsum

LOREM IPSUM DOLOR SIT AMET, CONSECTETUER ADIPISCING ELIT.

NUNC VIVERRA IMPERDIET ENIM. FUSCE EST. VIVAMUS A TELLUS.

PELLENTESQUE HABITANT MORBI TRISTIQUE SENECTUS ET NETUS.