File I/O for Game Developers:

Past, Present, and Future

GUY DAVIDSON

Cppcon
The C++ Conference

2023
October 01 - 06

FILE I/O: PAST, PRESENT AND FUTURE

CPPCON OCTOBER 3RD 2023

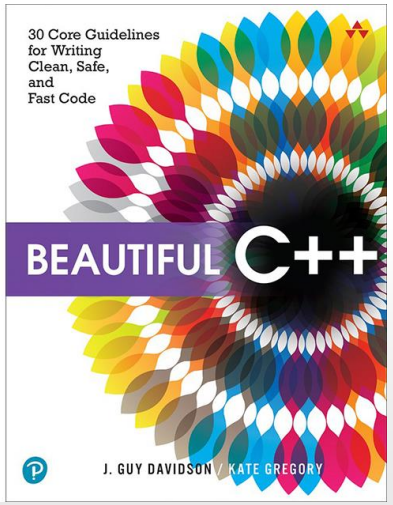
GUY DAVIDSON
@HATCAT01

INTRODUCTIONS

- Head of Engineering Practice at Creative Assembly
- 1980 – Acorn Atom
- ISO/IEC JTC1/SC22/WG21
- BSI IST/5
- Avid conference speaker and organiser

> *Beautiful C++* presents the C++ Core Guidelines from a developer's point of view with an emphasis on what benefits can be obtained from following the rules and what nightmares can result from ignoring them. For true geeks, it is an easy and entertaining read. For most software developers, it offers something new and useful.

Bjarne Stroustrup, inventor of C++ and co-editor of the C++ Core Guidelines



BEAUTIFUL C++

AGENDA

- Why do we have files?
- What is a filesystem?
- Why should we avoid buffered file IO?
- How do we optimise unbuffered file IO?
- How might the standard help us in future?

AGENDA

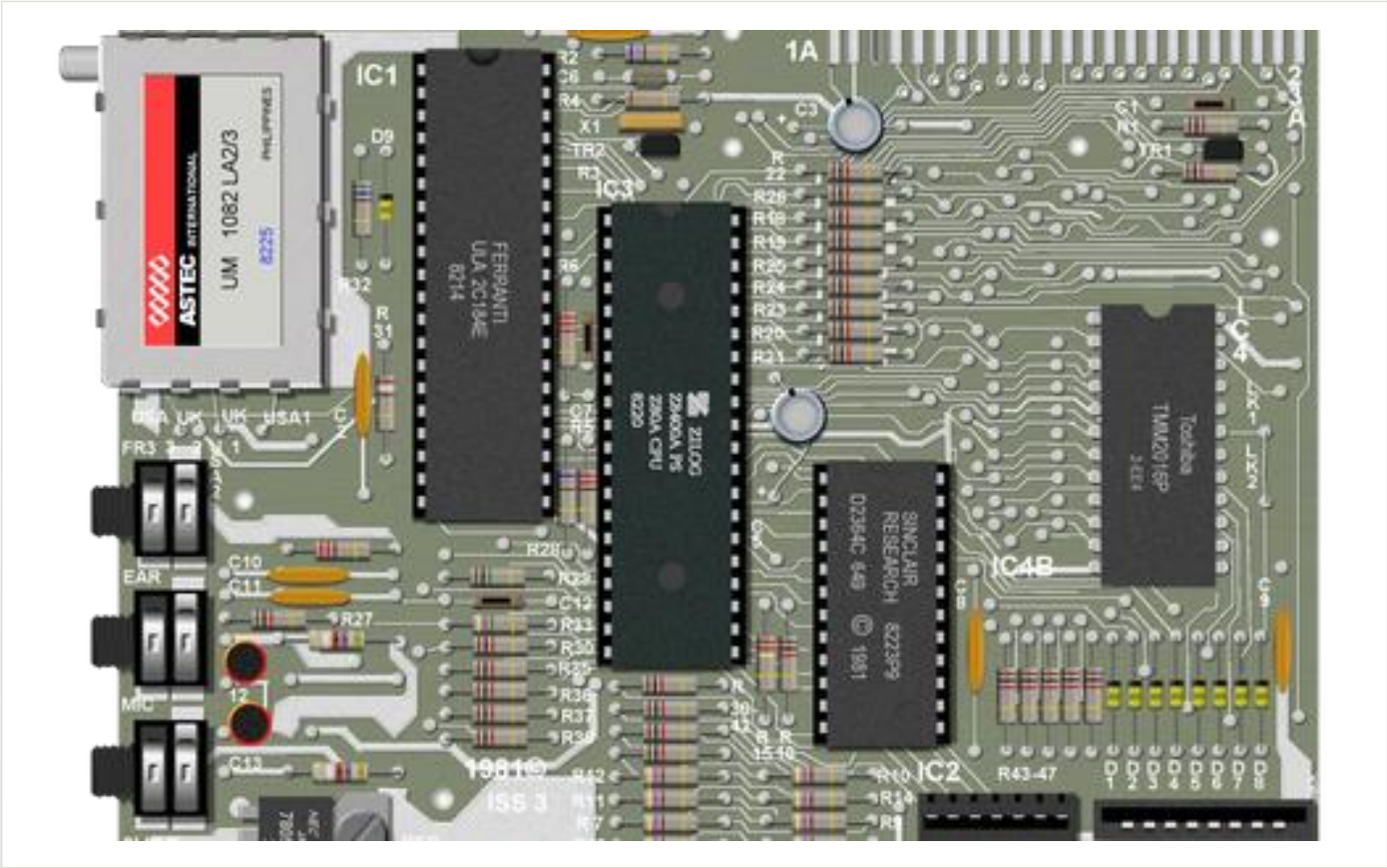
- Sub-megabyte days
- More RAM, more disk capacity
- Moving data into and out of RAM
- File IO in C++ from fstream to the OS SDK
- The 64-bit address space

SUB-MEGABYTE DAYS

- Why do we have files?

SUB-MEGABYTE DAYS

SUB-MEGABYTE DAYS

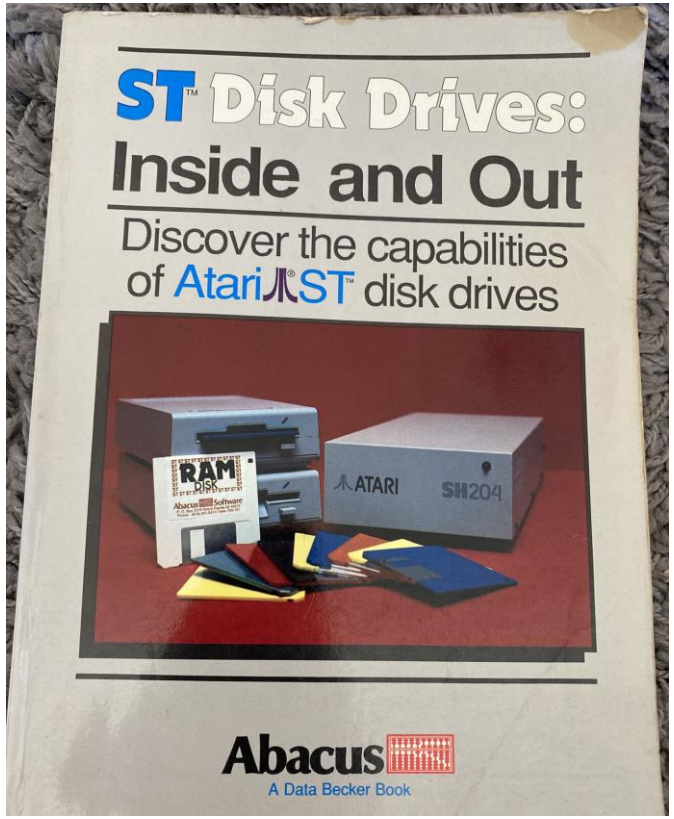


SUB-MEGABYTE DAYS

SUB-MEGABYTE DAYS

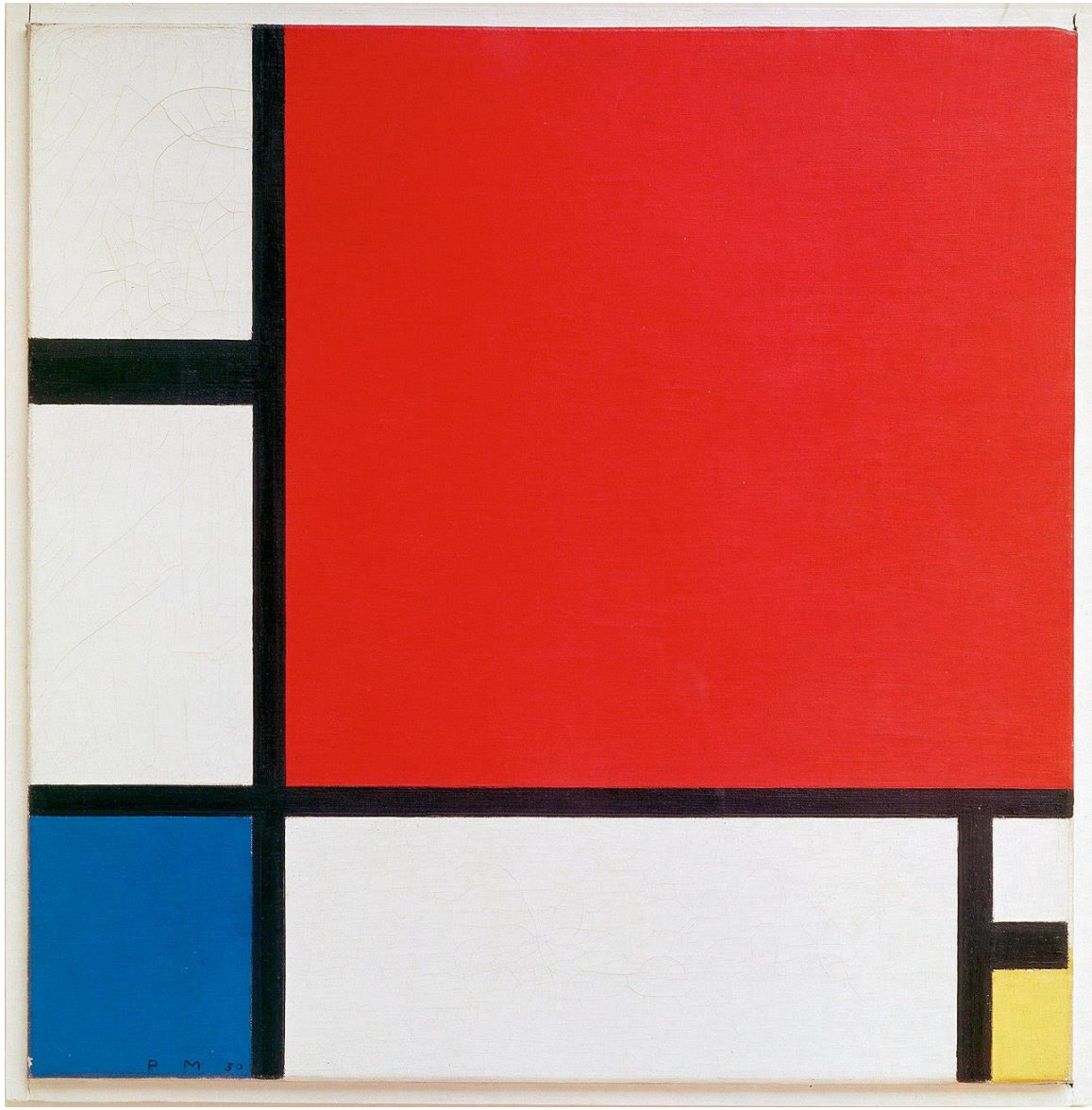
SUB-MEGABYTE DAYS

SUB-MEGABYTE DAYS



SUB-MEGABYTE DAYS

SUB-MEGABYTE DAYS


P M 30

MORE RAM, MORE DISK CAPACITY

- Picture of a 44MB hard disk

MORE RAM, MORE DISK CAPACITY

MORE RAM, MORE DISK CAPACITY

- FAT16

MORE RAM, MORE DISK CAPACITY

- FAT16
- File Allocation Table

MORE RAM, MORE DISK CAPACITY

- FAT16
- File Allocation Table
- Root directory

MORE RAM, MORE DISK CAPACITY

- FAT16
- File Allocation Table
- Root directory
- Entry

MORE RAM, MORE DISK CAPACITY

- FAT16
- File Allocation Table
- Root directory
- Entry
- Disk format

MORE RAM, MORE DISK CAPACITY

“The Portable Operating System Interface (POSIX; IPA: /ˈpɒz.ɪks/) is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines both the system and user-level application programming interfaces (APIs), along with command line shells and utility interfaces, for software compatibility (portability) with variants of Unix and other operating systems. POSIX is also a trademark of the IEEE. POSIX is intended to be used by both application and system developers.”

MORE RAM, MORE DISK CAPACITY

- Implement strong consistency. For example, if a write happened before a read, the read must return the data written.
- Have atomic writes, where a read either returns all data written by a concurrent write or none of the data but is not an incomplete write.
- Implement certain operations, like random reads, writes, truncate, or fsync.
- Control access to files using permissions and implement calls like chmod, chown, and so on to modify them.

MORE RAM, MORE DISK CAPACITY

- File access –
  fopen
  fclose
  fflush
- Direct i/o –
  fread
  fwrite
- Unformatted i/o –
  fgetc/fgets
  fputc/fputs
  getchar
  putchar
- Formatted i/o –
  scanf
  printf
- File positioning –
  ftell
  fgetpos
  fseek
  fsetpos
  rewind
- Operations on files –
  remove
  rename

MORE RAM, MORE DISK CAPACITY

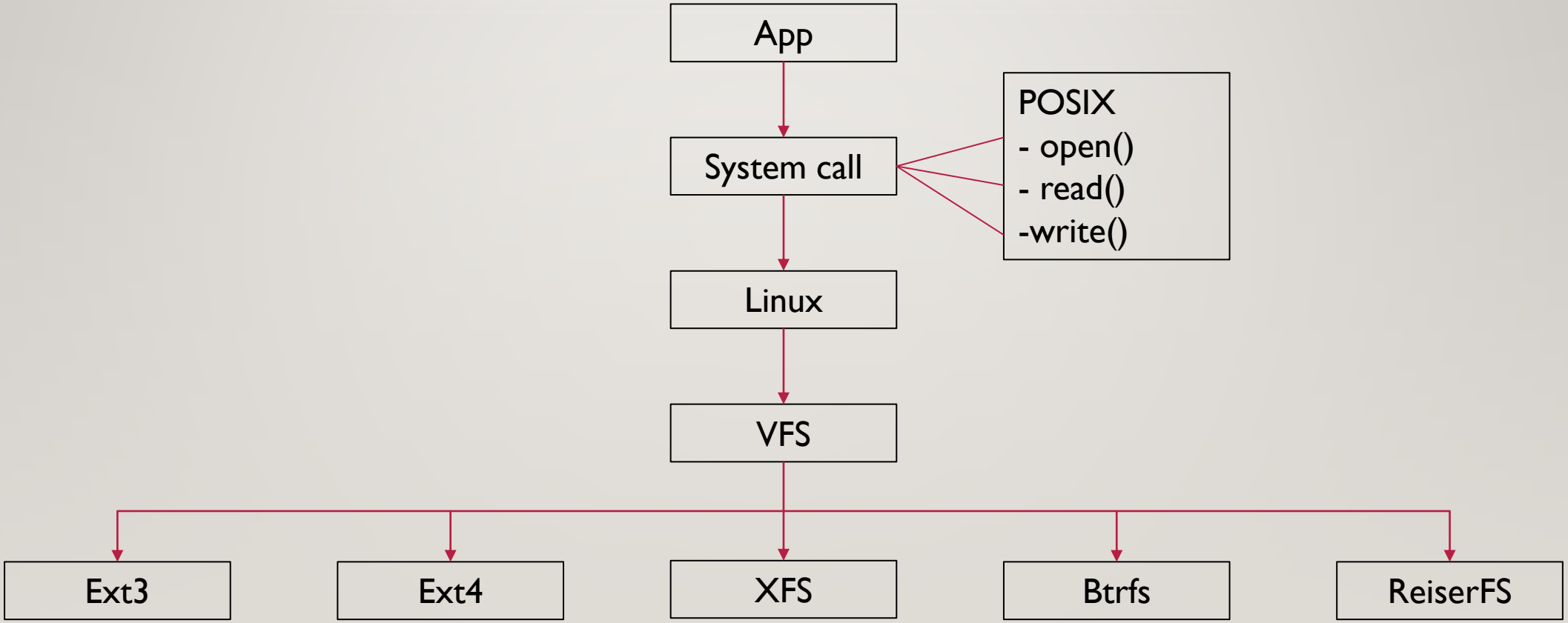
MORE RAM, MORE DISK CAPACITY

```
int fscanf(FILE* stream, char const* format, …);
int fprintf(FILE* stream, char const* format, …);
int fgetc(FILE* stream);
int fputc(int ch, FILE* stream);
size_t fread(void* buffer, size_t size, size_t count, FILE* stream);
size_t fwrite(void* buffer, size_t size, size_t count, FILE* stream);
```

MORE RAM, MORE DISK CAPACITY

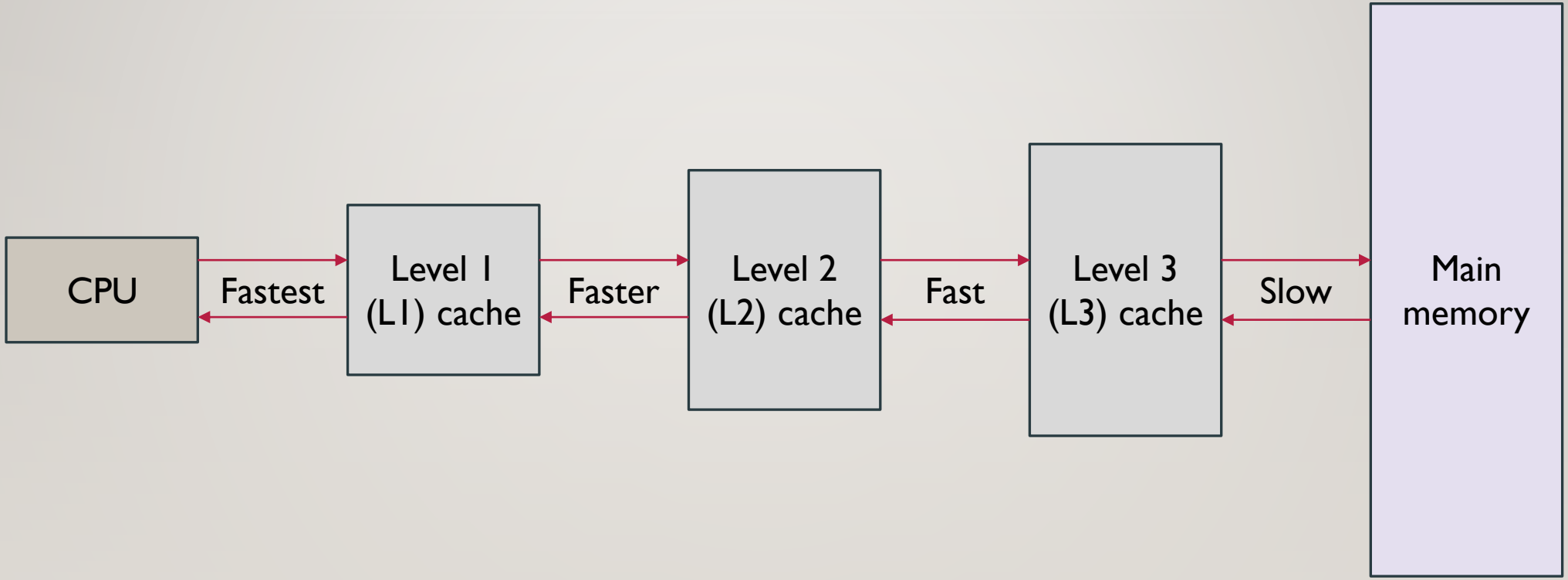
- `std::filesystem`
- Directories, links, block files, character files, sockets…
- Special names
- Paths
- Directory iteration, remaining space, permissions
- Copy files, create directories, file sizes, resize, remove

MORE RAM, MORE DISK CAPACITY



MORE RAM, MORE DISK CAPACITY

MOVING DATA INTO AND OUT OF RAM



MOVING DATA INTO AND OUT OF RAM

MOVING DATA INTO AND OUT OF RAM

- `FILE* fopen(char const* filename, char const* mode);`
- `BUFSIZ`
- `int setvbuf(FILE* stream, char* buffer, int mode, size_t size);`
- `size_t fread(void* buffer, size_t size, size_t count, std::FILE* stream);`

MOVING DATA INTO AND OUT OF RAM

- `fstream`
- `operator>>(char&), operator<<(char const&)`
- `fgetc(FILE*), fputc(FILE*)`

MOVING DATA INTO AND OUT OF RAM

```
struct staff_member {
  char surname[32];
  char forename[32];
  time_t join_date;
  int staff_id_hi;
  int staff_id_lo;
  int salary;
};
```

MOVING DATA INTO AND OUT OF RAM

```
fread(buf, sizeof(staff_member), 10, file);
fread(buf, 10, sizeof(staff_member), file);
fwrite(buf, sizeof(staff_member), 10, file);
```

MOVING DATA INTO AND OUT OF RAM

```
basic_istream& operator>>(staff_member&);

basic_ostream& operator<<(staff_member const&);
```

MOVING DATA INTO AND OUT OF RAM

```
operator();
```

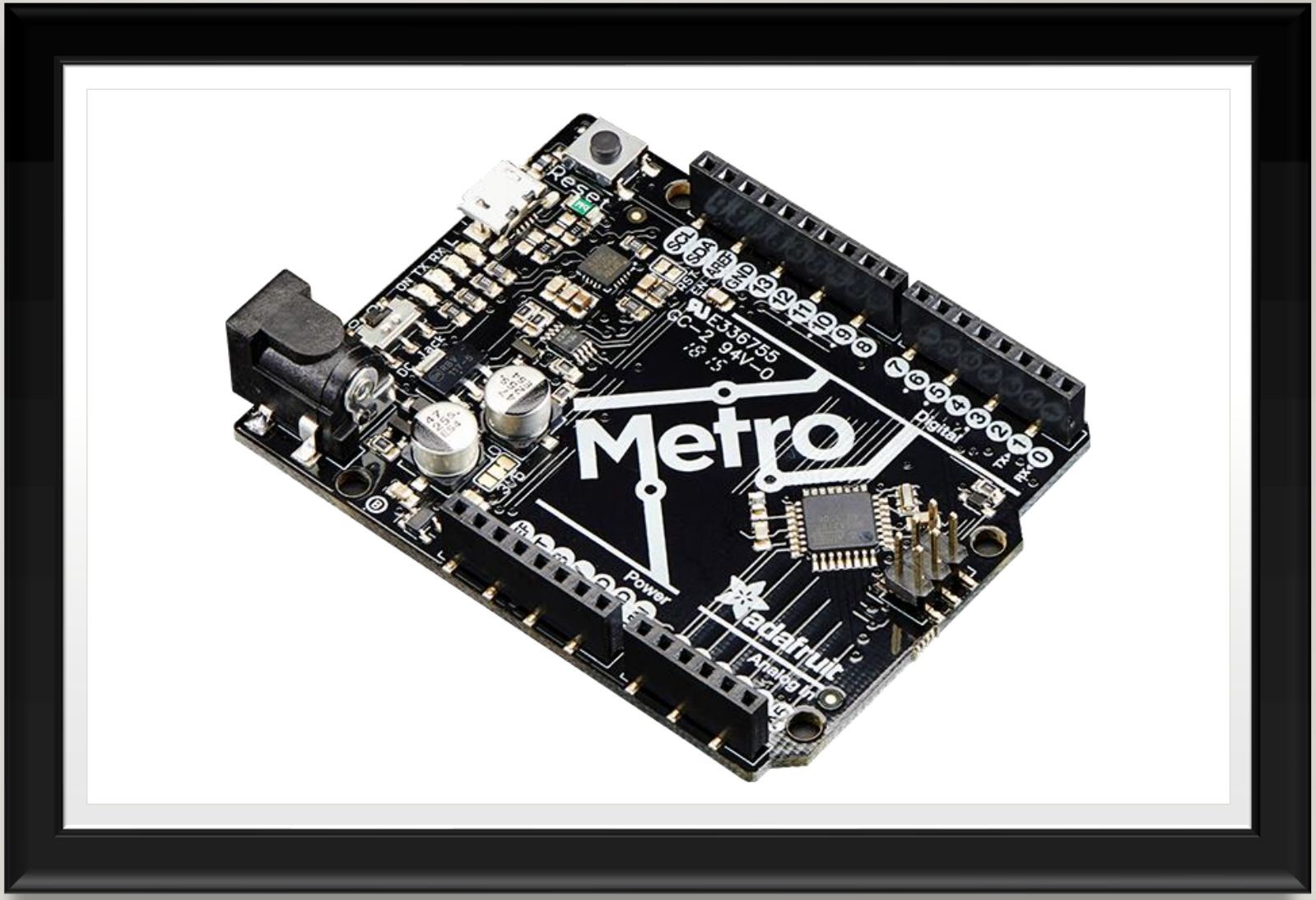
MOVING DATA INTO AND OUT OF RAM

```
operator();

operator[];
```

MOVING DATA INTO AND OUT OF RAM

```
basic_istream& operator>>(staff_member&);

basic_ostream& operator<<(staff_member const&);
```

MOVING DATA INTO AND OUT OF RAM



FILE IO IN C++ FROM FSTREAM TO THE OS SDK

```
constexpr size_t buf_size = 4096u;

char buf[buf_size];

ifstream file(“huge_log.txt”);

file.read(buf, buf_size);
```

FILE IO IN C++ FROM FSTREAM TO THE OS SDK

```
constexpr size_t buf_size = 4096u;

char buf[buf_size];

FILE* f = fopen("huge_log.txt", "r");

fread(buf, 1, buf_size, f);
```

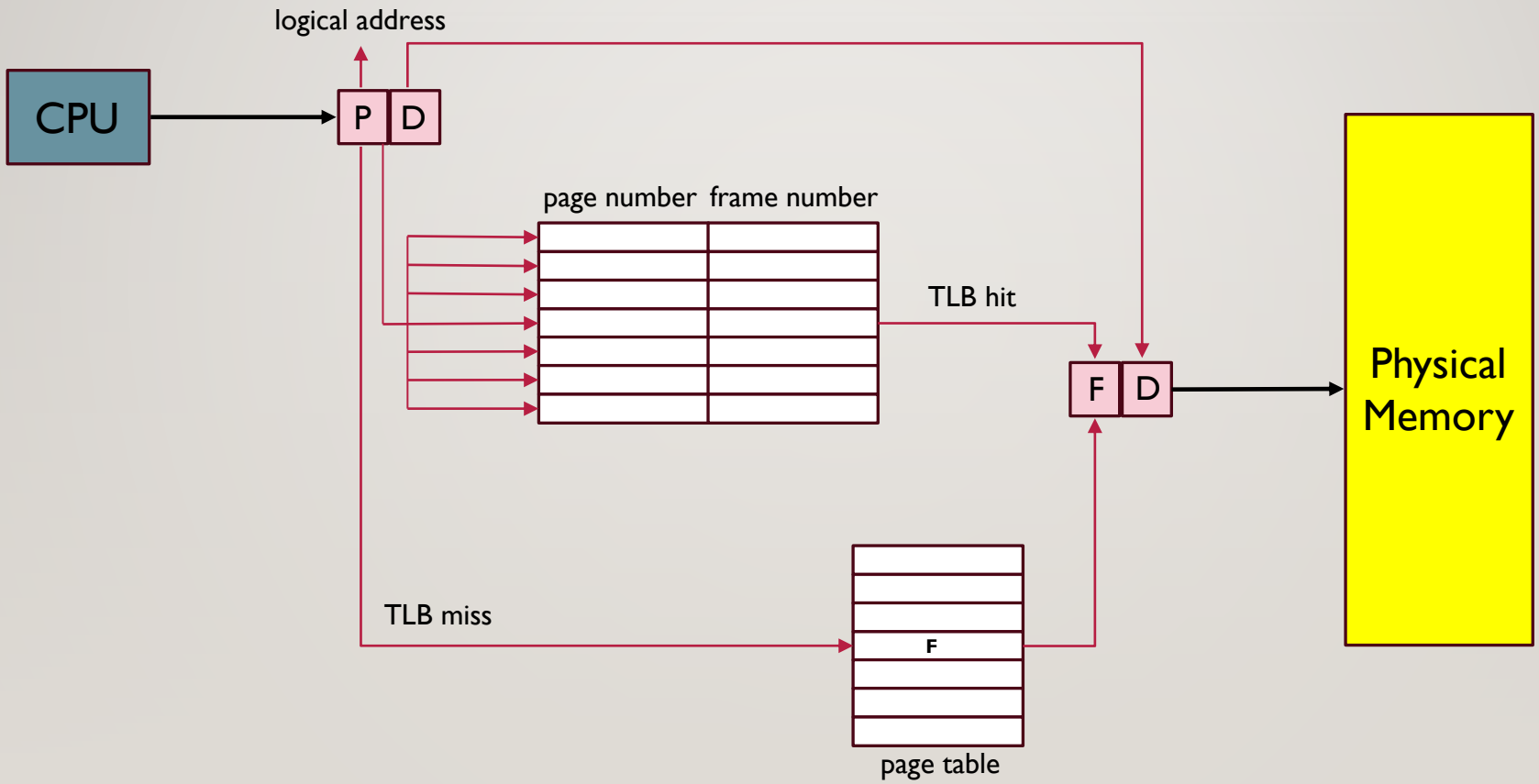
FILE IO IN C++ FROM FSTREAM TO THE OS SDK

```
constexpr size_t buf_size = 4096u;

char buf[buf_size];

HANDLE file  = CreateFile(“huge_log.txt”, GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

ReadFile(file, buf, 4096, NULL, NULL);
```

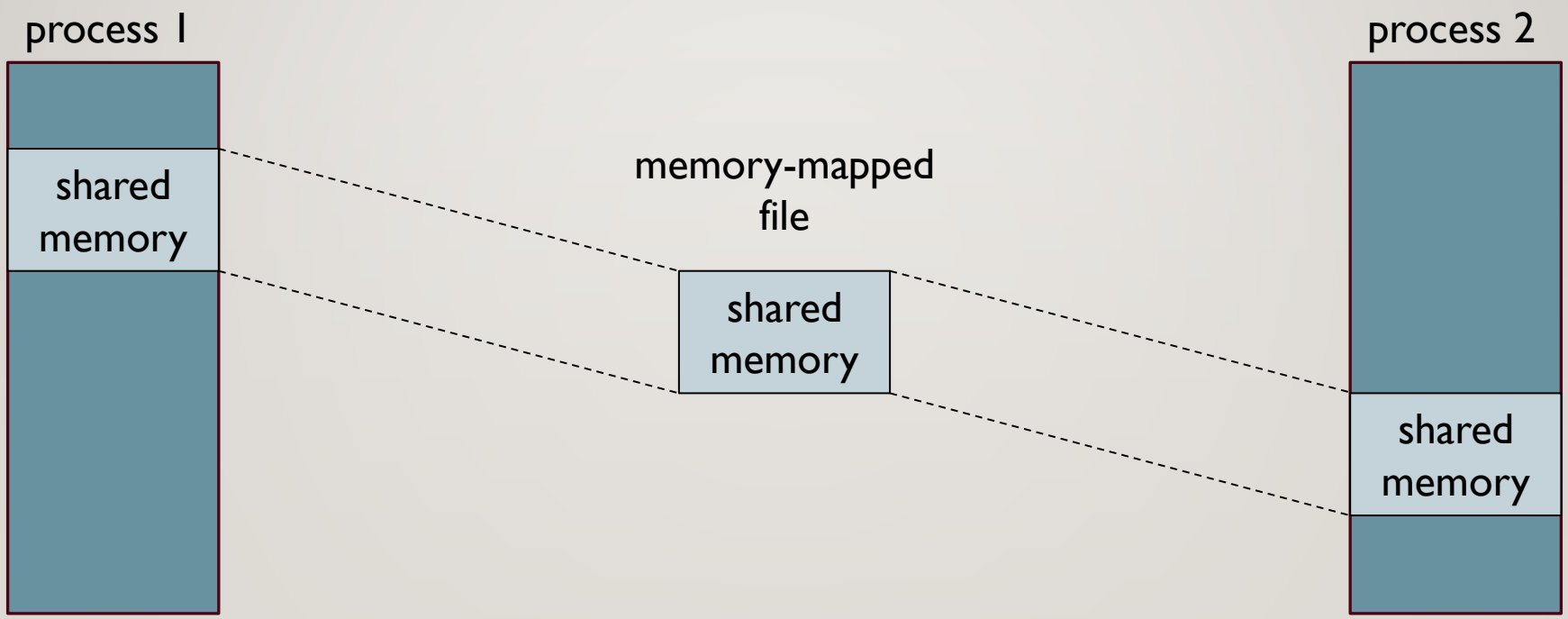
FILE IO IN C++ FROM FSTREAM TO THE OS SDK

FILE IO IN C++ FROM FSTREAM TO THE OS SDK

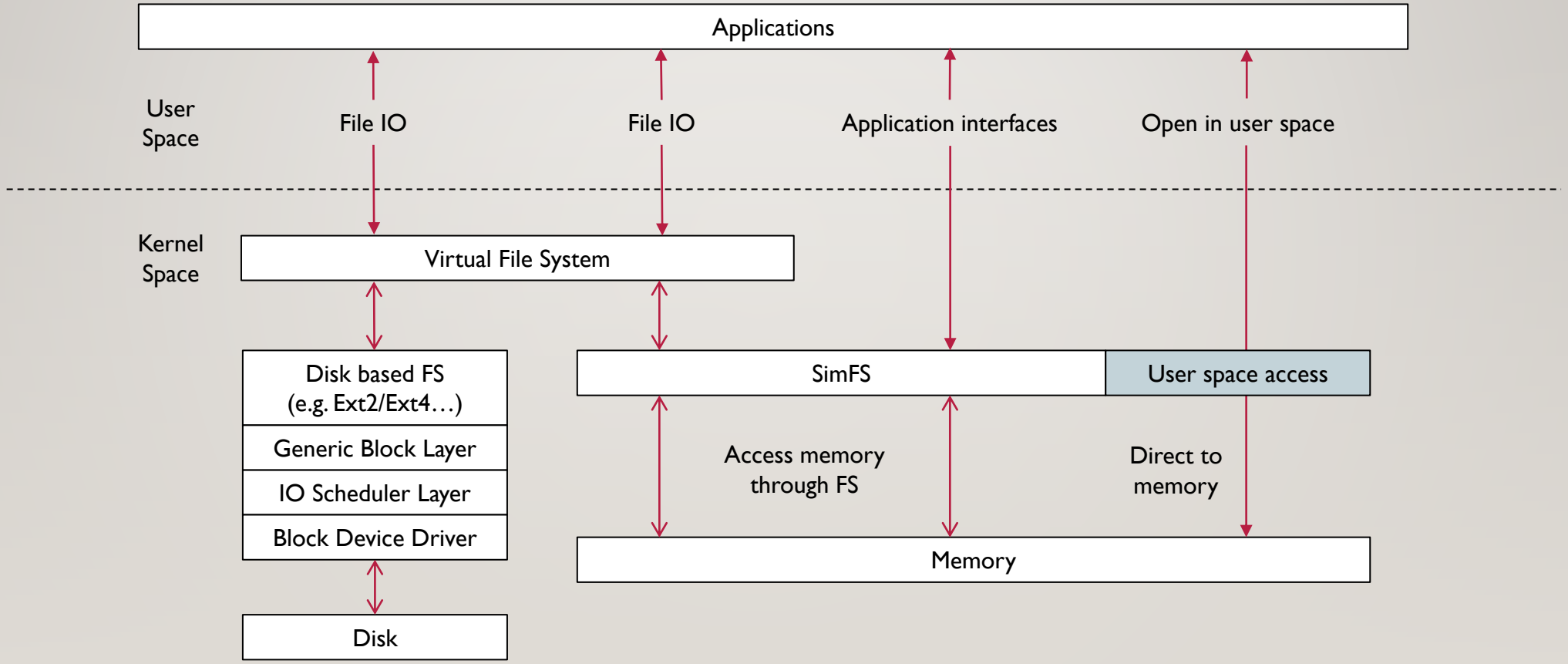
FILE IO IN C++ FROM FSTREAM TO THE OS SDK



FILE IO IN C++ FROM FSTREAM TO THE OS SDK



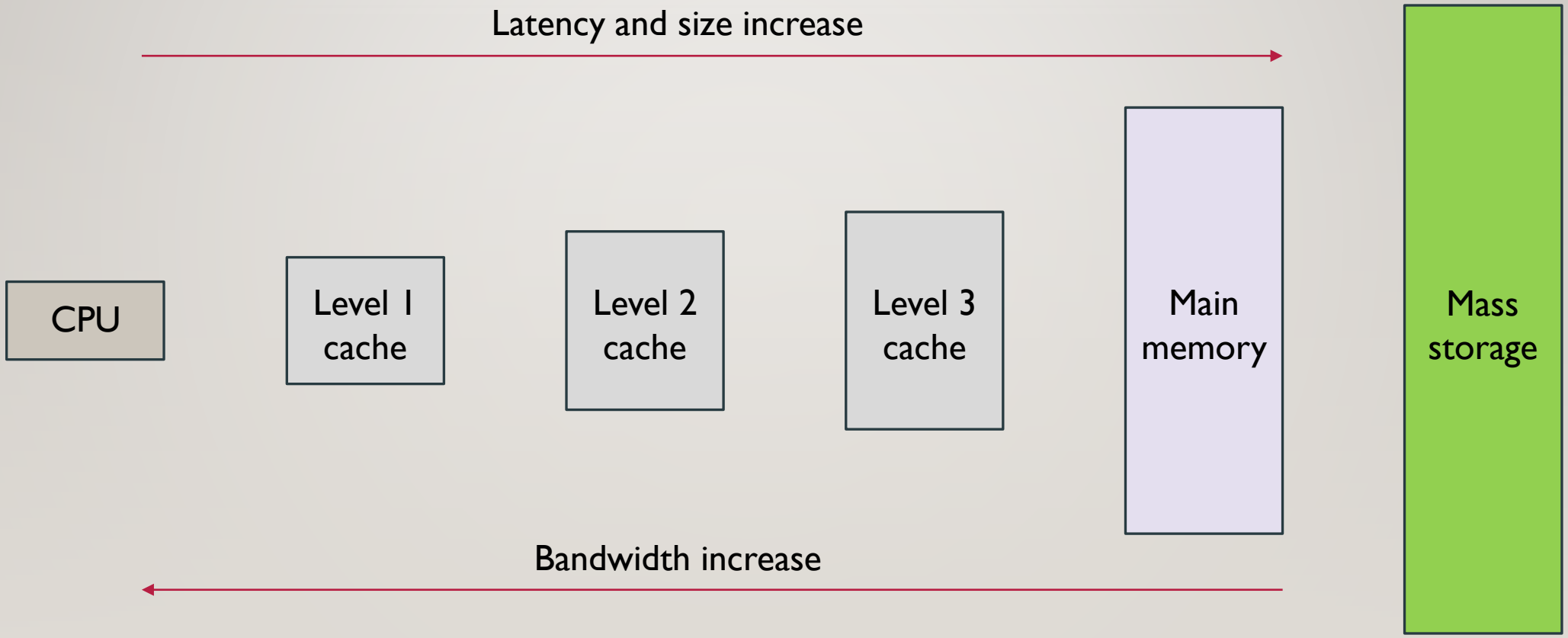
FILE IO IN C++ FROM FSTREAM TO THE OS SDK



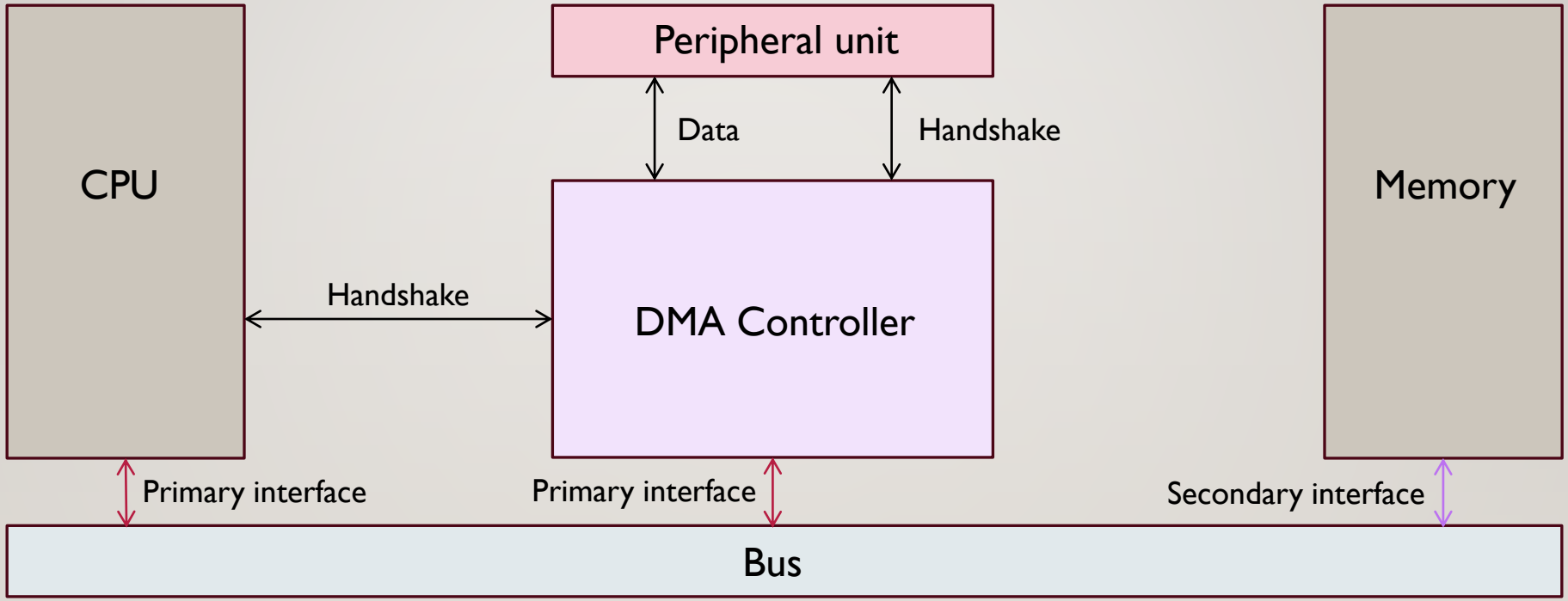
FILE IO IN C++ FROM FSTREAM TO THE OS SDK

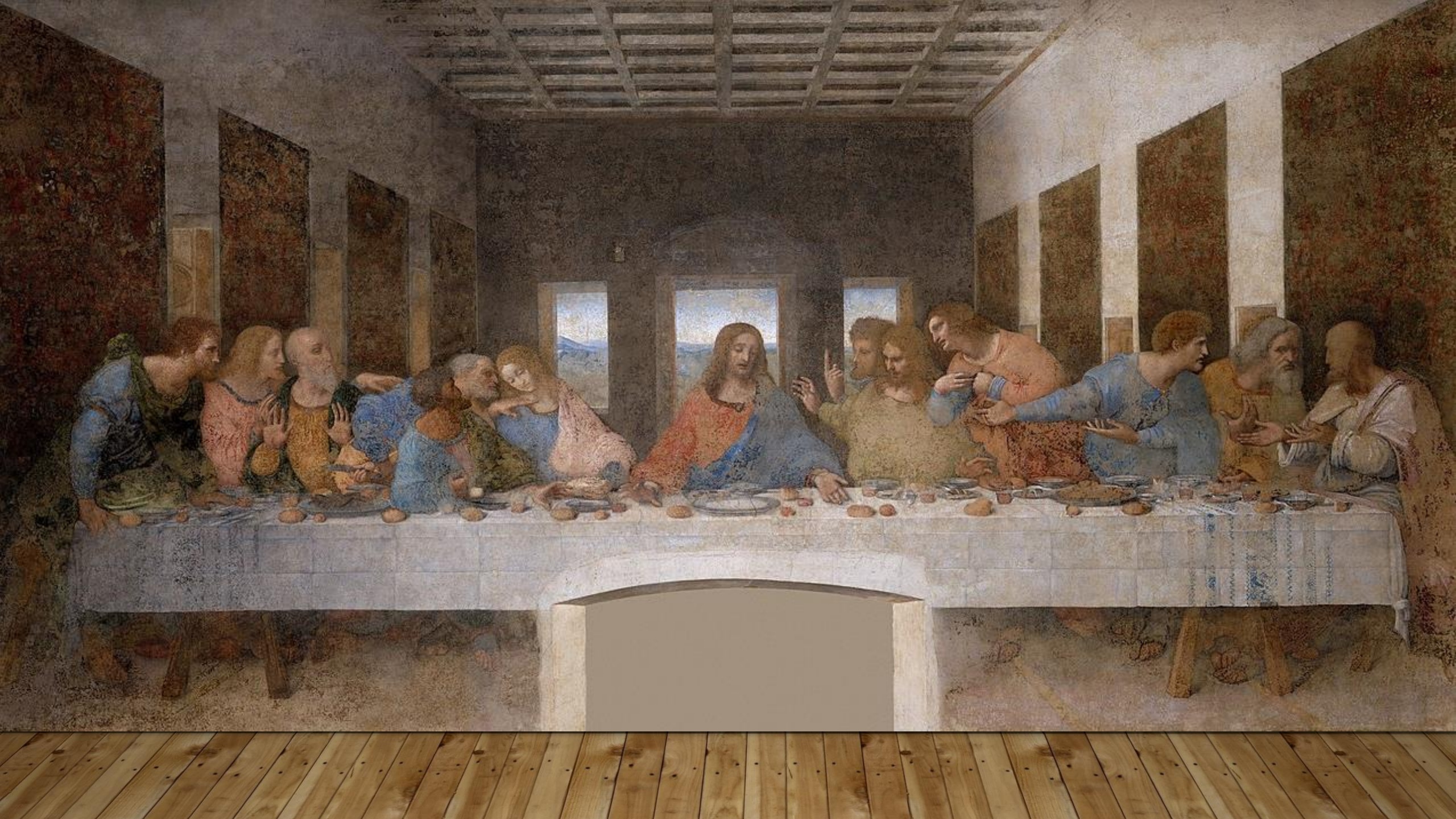
```
HANDLE file = CreateFile("huge_log.txt", GENERIC_READ,
  FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

HANDLE mapping = CreateFileMapping(file, NULL, PAGE_READONLY, 0, 0,
  NULL);

LPVOID buf = MapViewOfFile(mapping, FILE_MAP_READ, 0, 0, 0);
```

FILE IO IN C++ FROM FSTREAM TO THE OS SDK



FILE IO IN C++ FROM FSTREAM TO THE OS SDK





THE 64-BIT ADDRESS SPACE

- https://wg21.link/p1040
- "I'm very keen on std::embed. I've been hand-embedding data in executables for NEARLY FORTY YEARS now. — Guy "Hatcat" Davidson, June 15, 2018"

```
int main (int, char*[]) {
  constexpr span<const byte> fxaa_binary =
    embed("fxaa.spirv");
  …
}
```

THE 64-BIT ADDRESS SPACE

- https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3017.htm

```
/* default is unsigned char */
const unsigned char icon_display_data[] = {
    #embed "art.png"
};
```

THE 64-BIT ADDRESS SPACE

THE 64-BIT ADDRESS SPACE

- https://thephd.dev/finally-embed-in-c23
- https://developercommunity.visualstudio.com/t/Add-support-for-embed-as-voted-into-the/10451640

THE 64-BIT ADDRESS SPACE

- https://wg21.link/p1031
- 4KB -> RAM = 5µs
- 4KB HDD -> RAM = 26,000µs
- 4KB SSD -> RAM = 800µs
- 4KB NVMe -> RAM = 300µs

THE 64-BIT ADDRESS SPACE

- https://wg21.link/p1883
- DirectStorage

SUMMARY

- Why do we have files?
- What is a filesystem?
- Why should we avoid buffered file IO?
- How do we optimise unbuffered file IO?
- How might the standard help us in future?

SUMMARY

- https://wg21.link/p1040
- https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3017.htm
- https://thephd.dev/finally-embed-in-c23
- https://developercommunity.visualstudio.com/t/Add-support-for-embed-as-voted-into-the/10451640
- https://wg21.link/p1031
- https://wg21.link/p1883
- @hatcat01