CS4414 Recitation 7

Prelim1 solution

10/11/2024

Alicia Yang

Logistics

- HW 3 on Gradescope
- Due date:
  - Part 1. **10/11 (Friday, today)**
  - Part 2. **10/27 (Sunday)**
- **START EARLY**
  - This assignment takes more time than hw1 and hw2. Make sure to start early.
- Late submission
  - -5 points per day, maximum -15 (3 days late submission)

File System

How are files and directories organized?

What happens when you read a file?

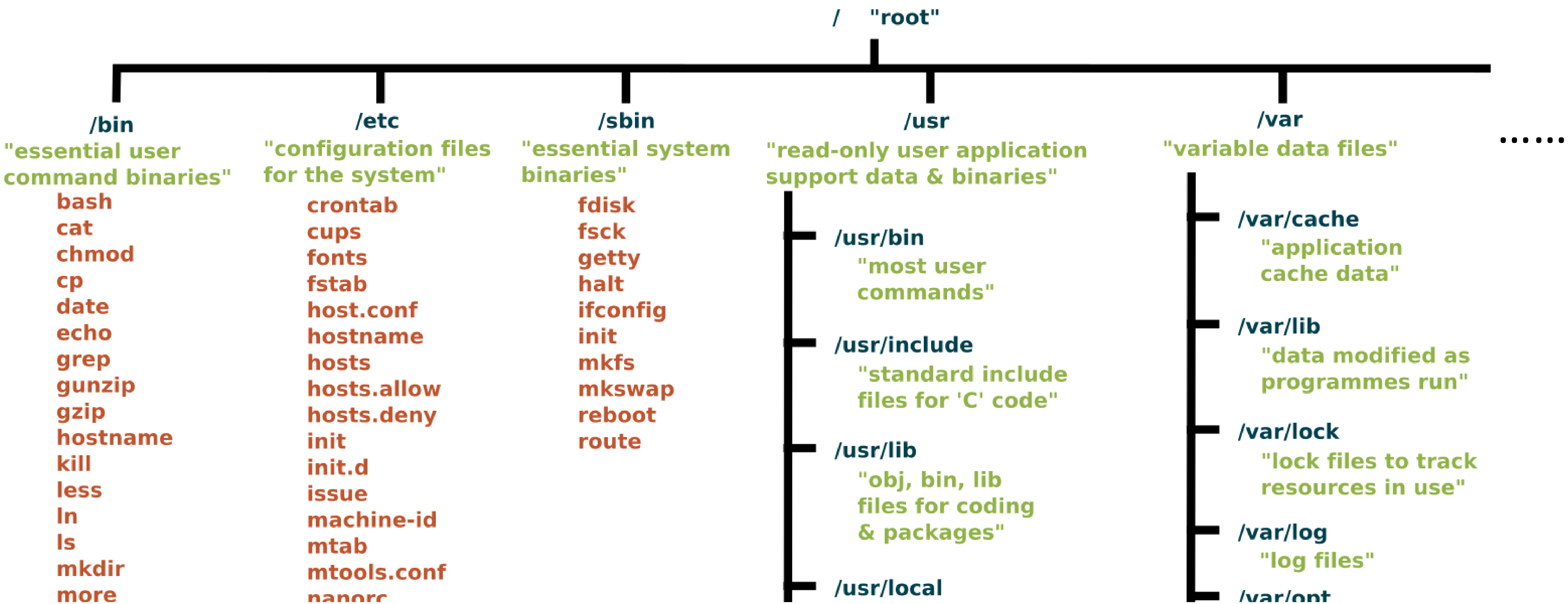
What happens when you delete a file?

Linux file system structure (simplified from organize and access perspectives)

- Pathname
- Inode
- Data blocks

Pathname

e.g. "/usr/local/alicia/doc1.txt"



/ "root"

/bin
"essential user command binaries"
- bash
- cat
- chmod
- cp
- date
- echo
- grep
- gunzip
- gzip
- hostname
- kill
- less
- ln
- ls
- mkdir
- more

/etc
"configuration files for the system"
- crontab
- cups
- fonts
- fstab
- host.conf
- hostname
- hosts
- hosts.allow
- hosts.deny
- init
- init.d
- issue
- machine-id
- mtab
- mtools.conf
- nanorc

/sbin
"essential system binaries"
- fdisk
- fsck
- getty
- halt
- ifconfig
- init
- mkfs
- mkswap
- reboot
- route

/usr
"read-only user application support data & binaries"
- /usr/bin "most user commands"
- /usr/include "standard include files for 'C' code"
- /usr/lib "obj, bin, lib files for coding & packages"
- /usr/local

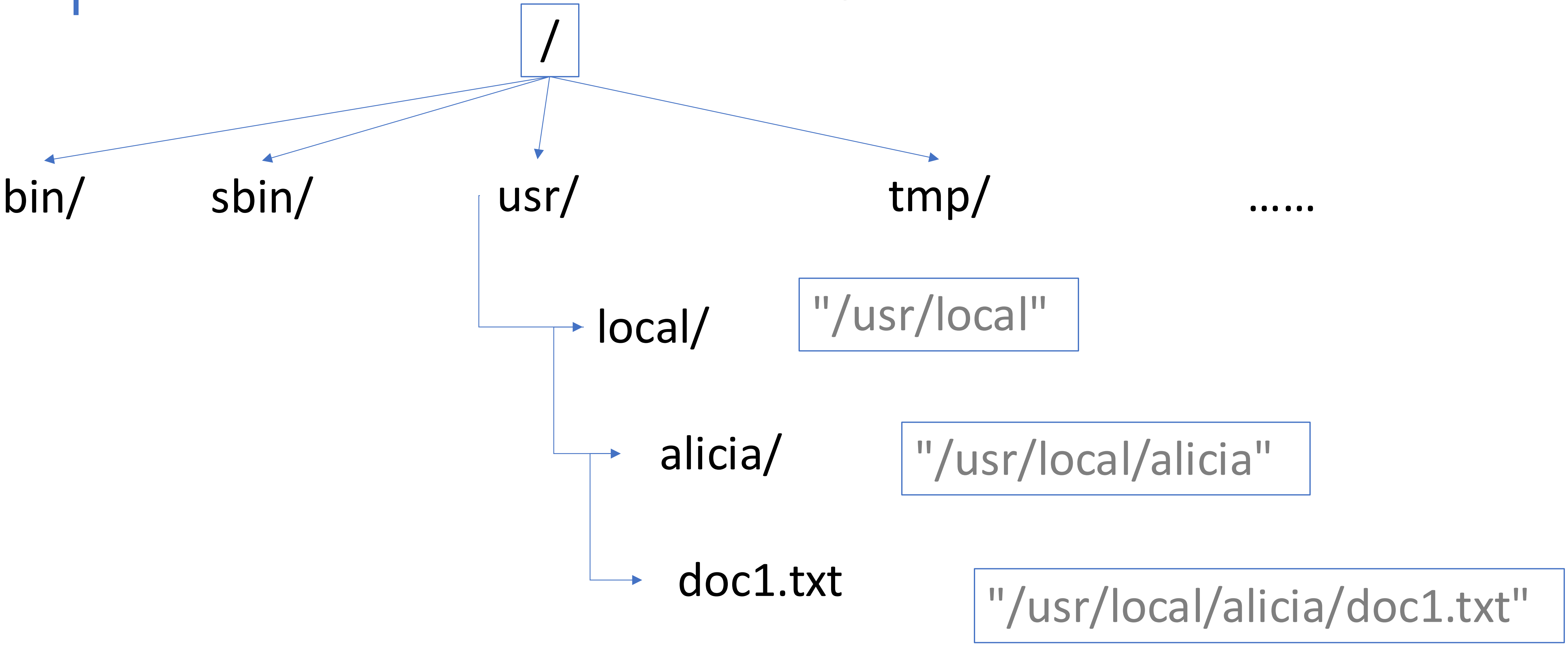
/var
"variable data files"
- /var/cache "application cache data"
- /var/lib "data modified as programmes run"
- /var/lock "lock files to track resources in use"
- /var/log "log files"
- /var/opt

......

https://www.linuxfoundation.org/blog/blog/classic-sysadmin-the-linux-filesystem-explained

Pathname

e.g. "/usr/local/alicia/doc1.txt"



/

bin/

sbin/

usr/

tmp/

......

local/

"/usr/local"

alicia/

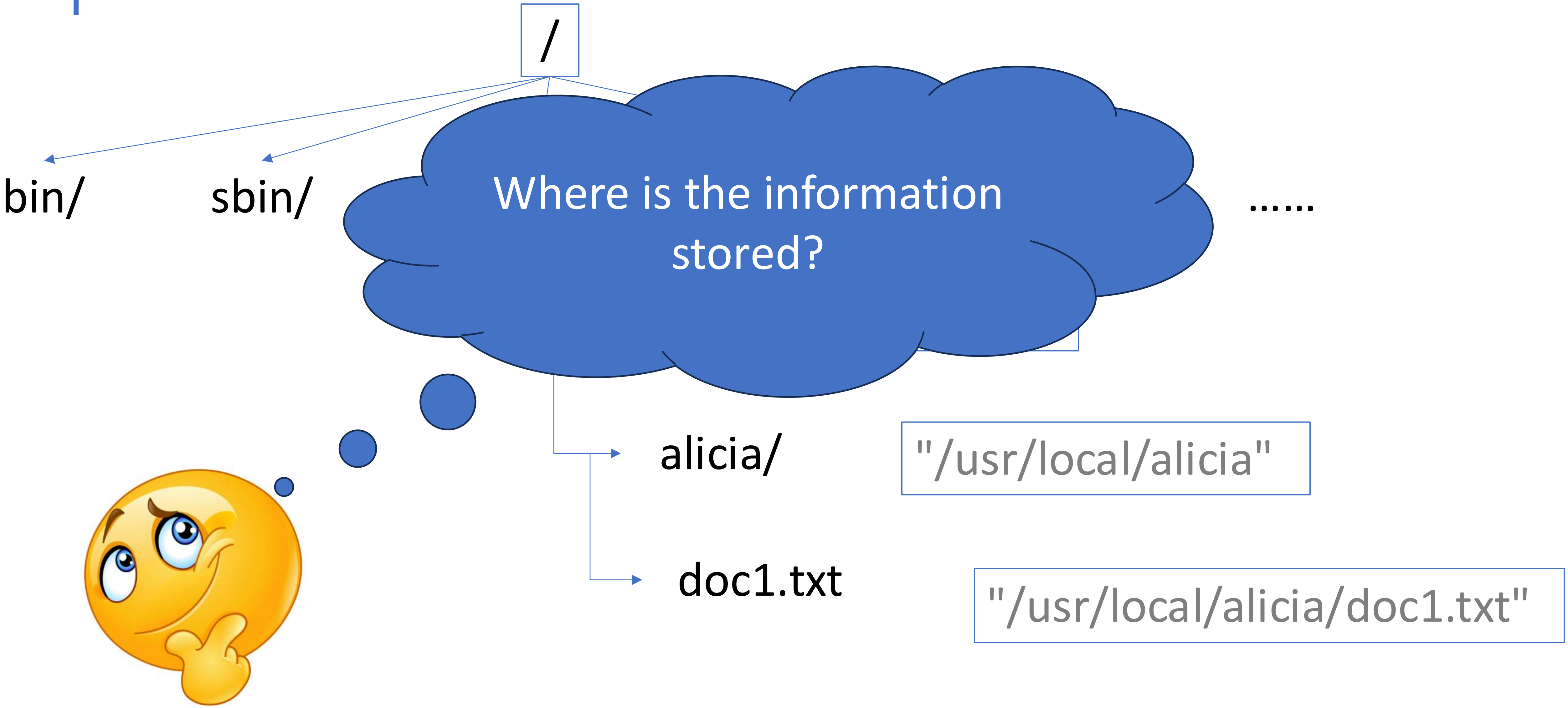
"/usr/local/alicia"

doc1.txt

"/usr/local/alicia/doc1.txt"

Pathname

e.g. "/usr/local/alicia/doc1.txt"



/

bin/

sbin/

......

Where is the information stored?

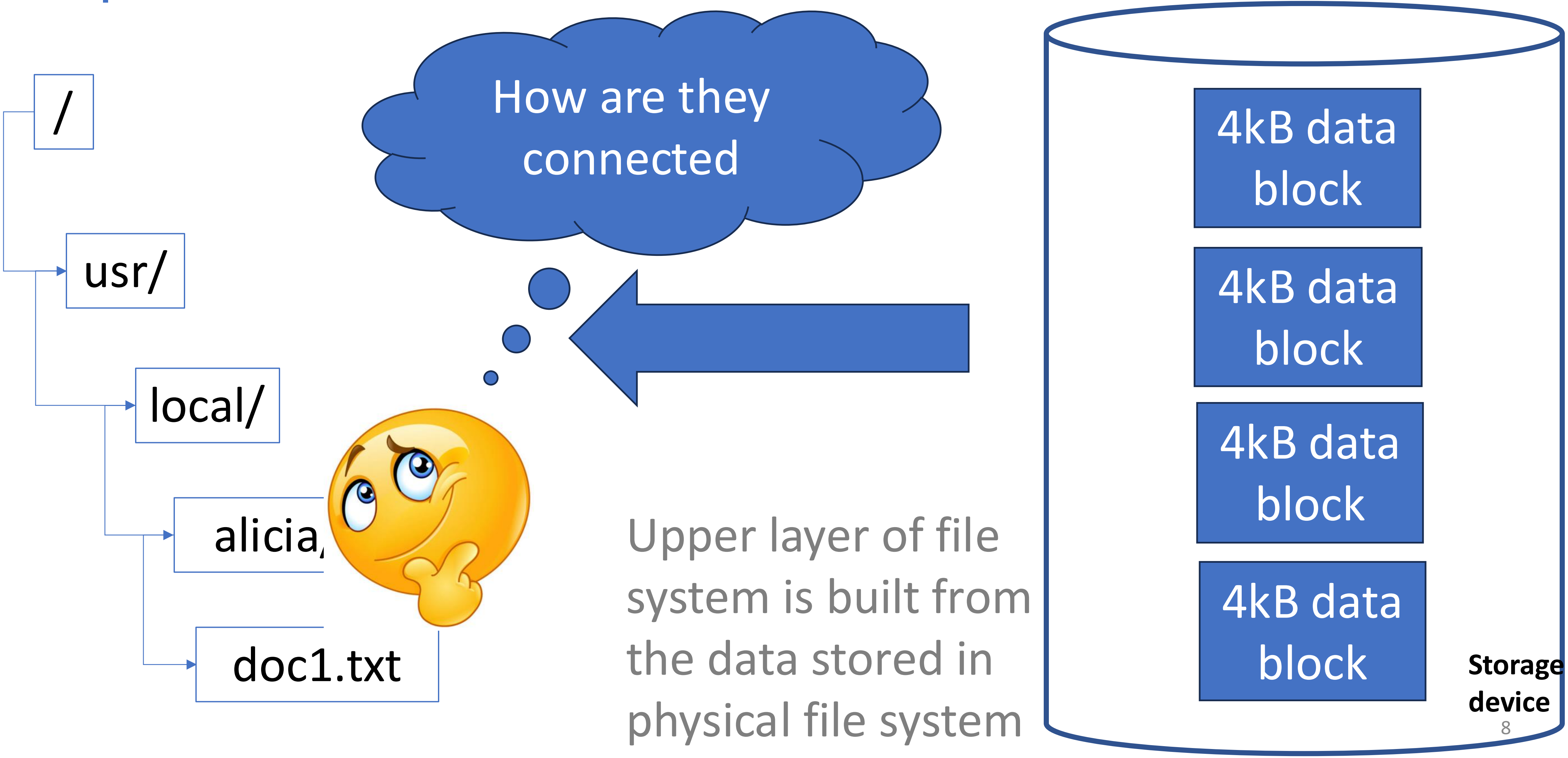
alicia/

"/usr/local/alicia"

doc1.txt

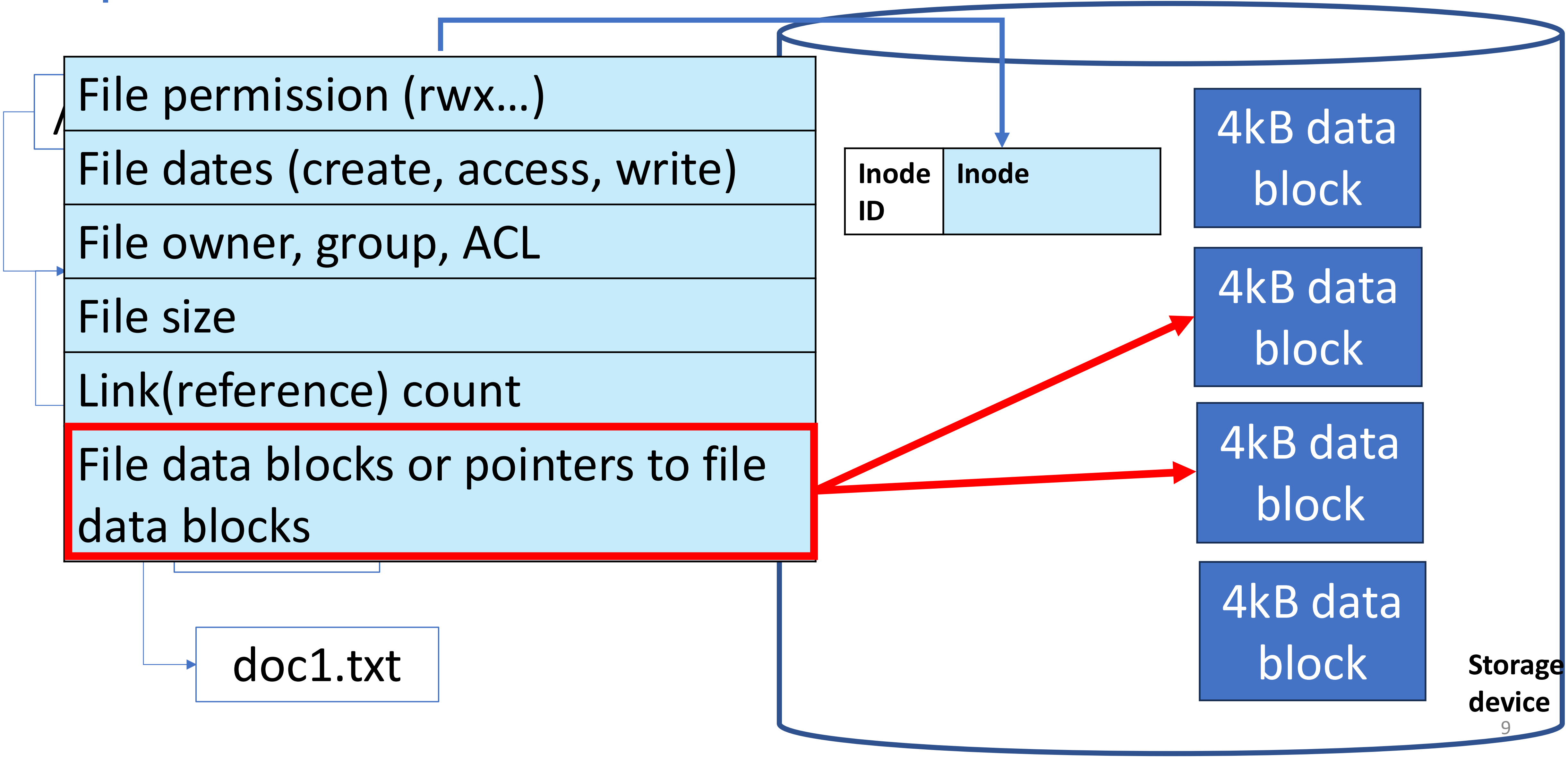
"/usr/local/alicia/doc1.txt"

Actual data are stored in data blocks



Upper layer of file system is built from the data stored in physical file system

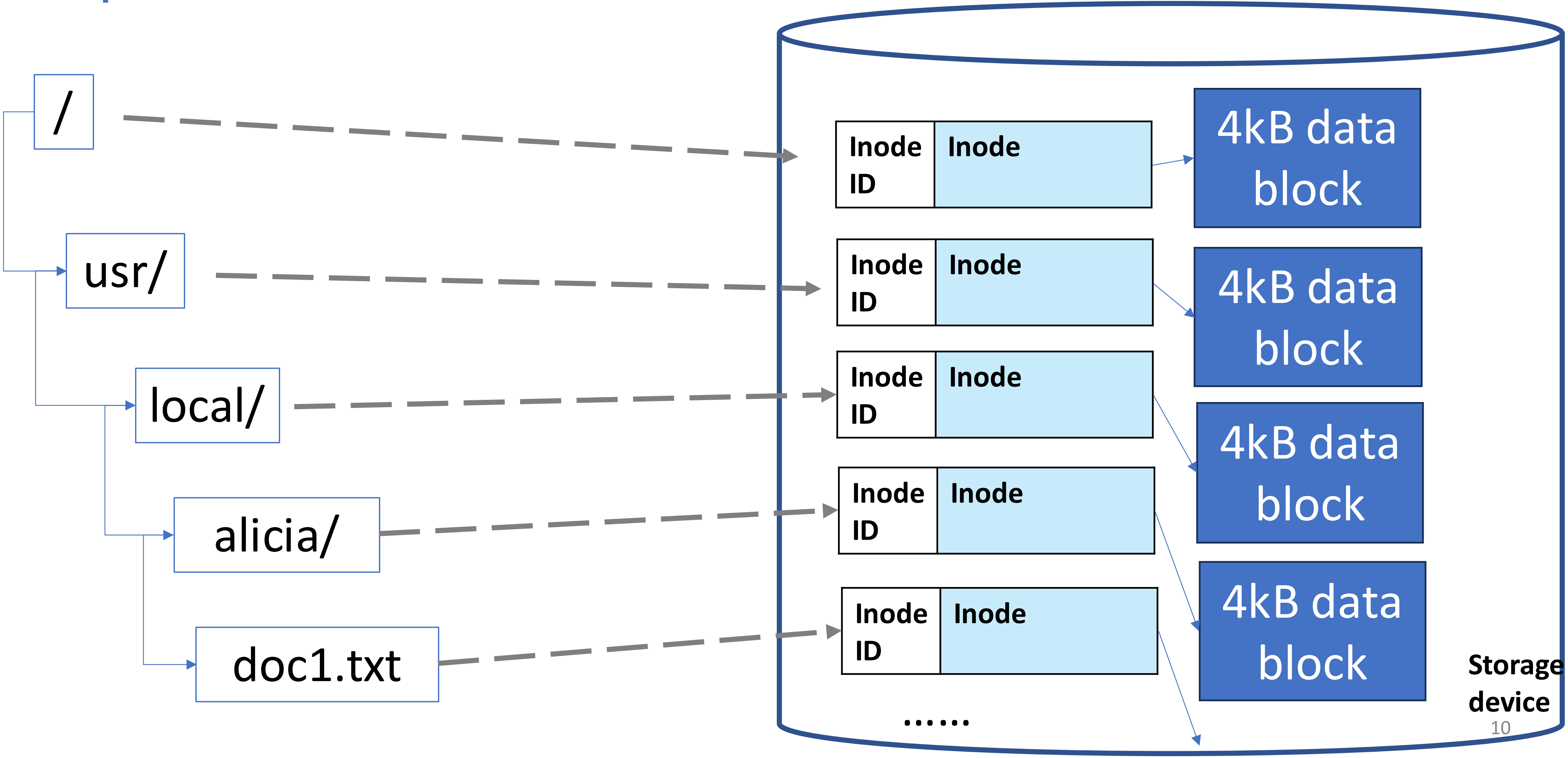
Inode: describe a file system object (directory/file)



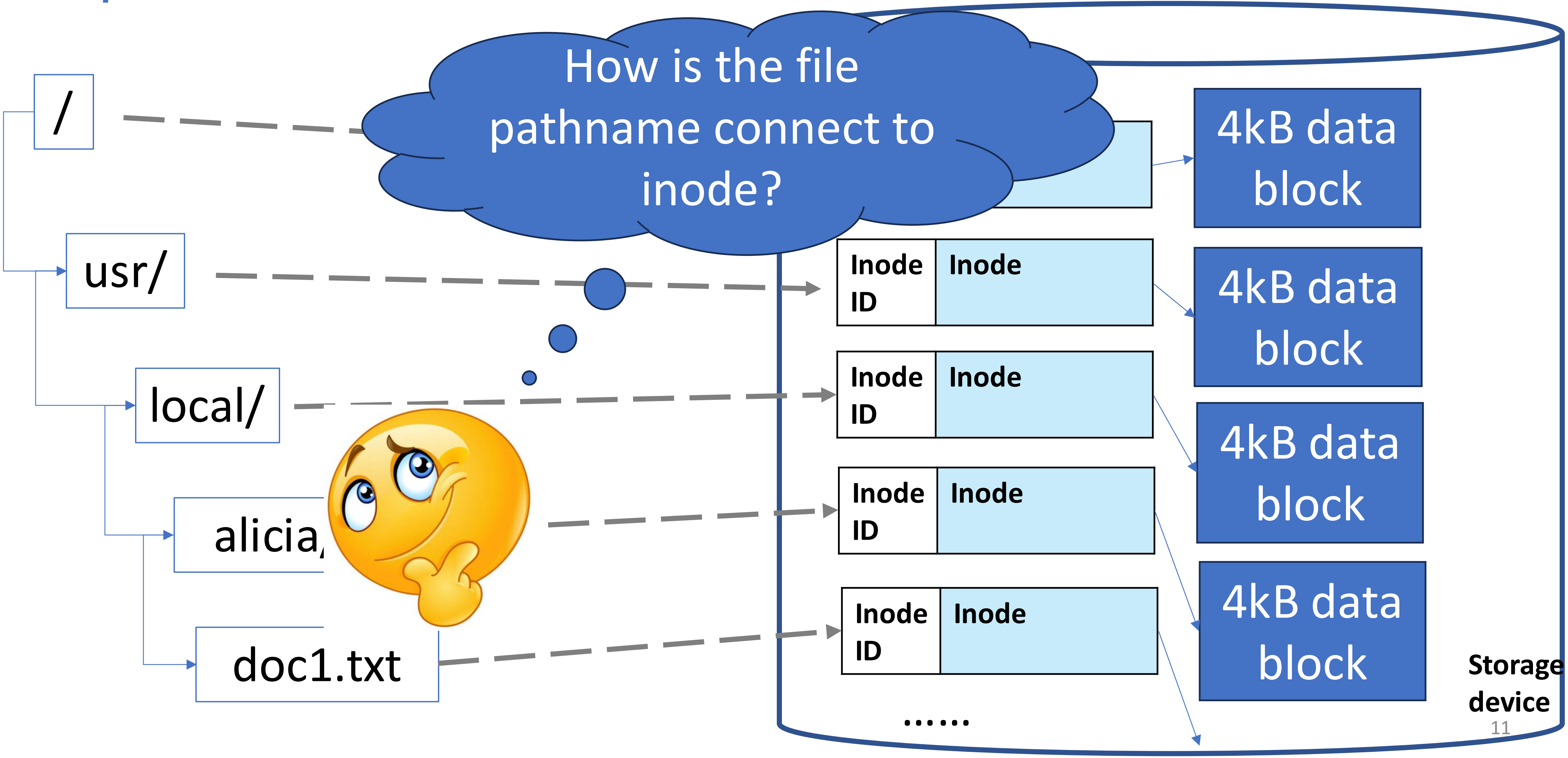
- File permission (rwx…)
- File dates (create, access, write)
- File owner, group, ACL
- File size
- Link(reference) count
- File data blocks or pointers to file data blocks

InodeTable

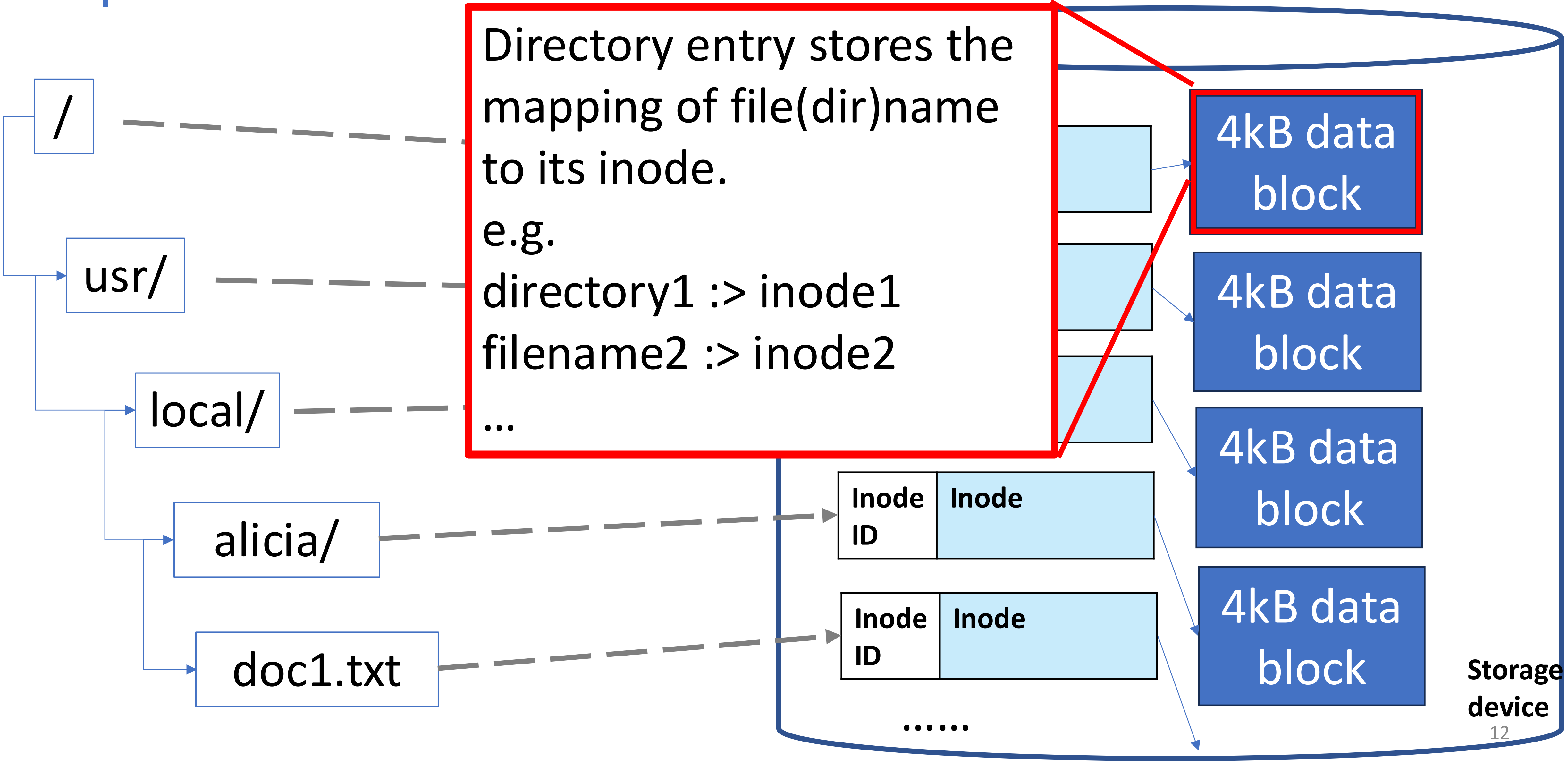
Datablock layout is different for different file systems, e.g. unix, ext, …



InodeTable


How is the file pathname connect to inode?
/
usr/
local/
alicia/
doc1.txt
Inode ID
Inode
Inode ID
Inode
Inode ID
Inode
Inode ID
Inode
……
4kB data block
4kB data block
4kB data block
4kB data block
Storage device

Directory entry



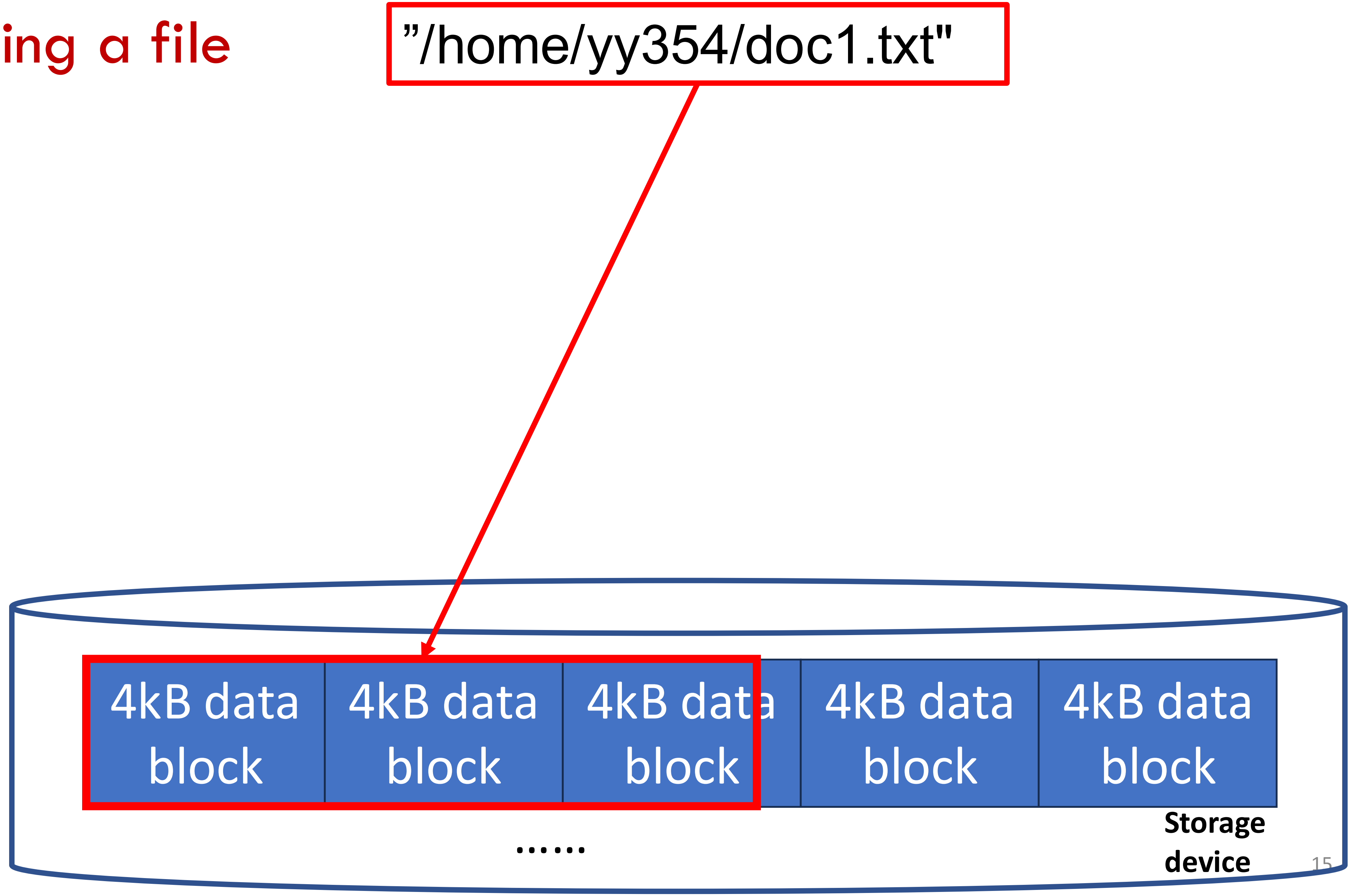
Directory entry stores the mapping of file(dir)name to its inode.
e.g.
directory1 :> inode1
filename2 :> inode2
…

File System

How are files and directories organized?

What happens when you read a file?

Reading a file



”/home/yy354/doc1.txt"

4kB data block

4kB data block

4kB data block

4kB data block

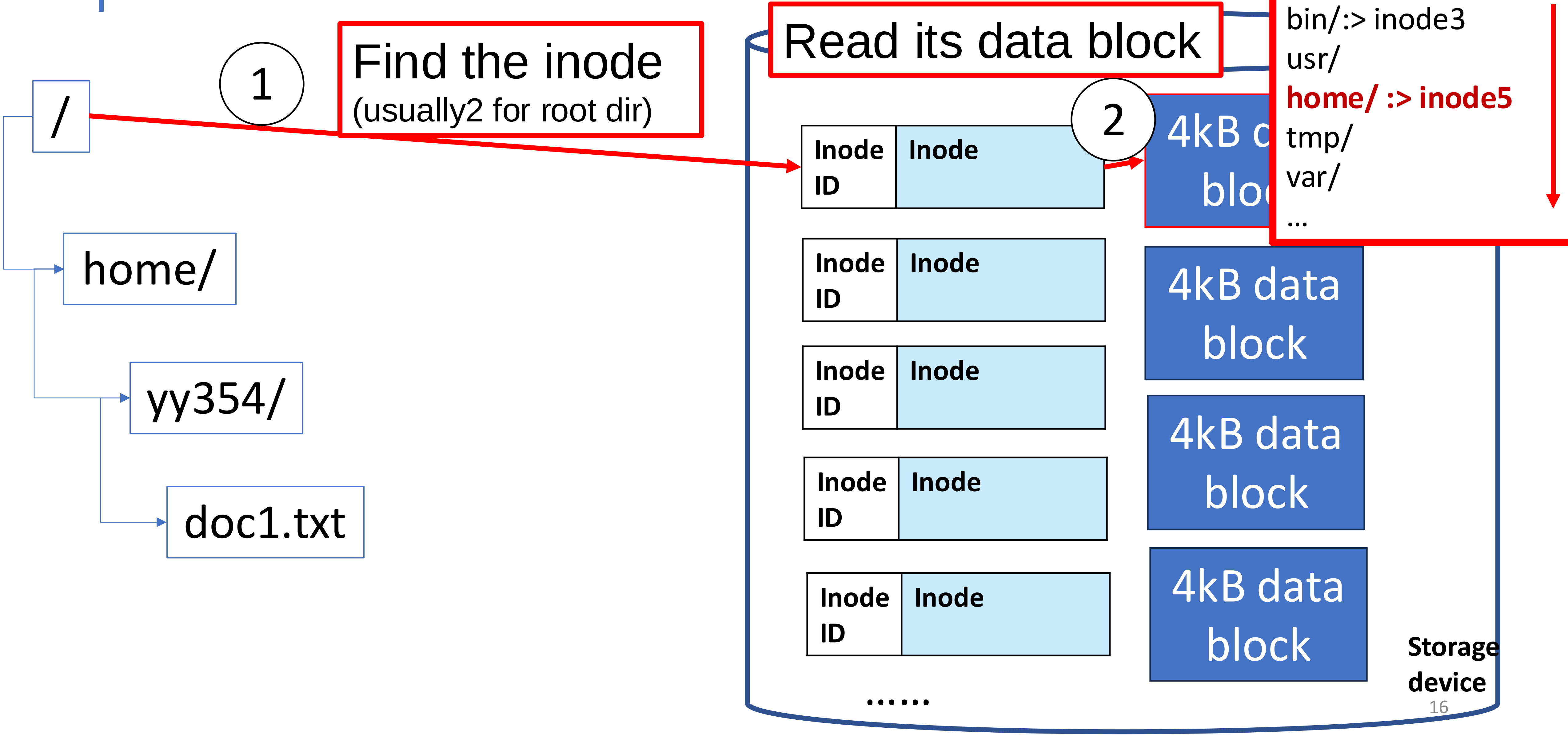
4kB data block

……

Storage device

Start from root

"/"



Find the inode (usually2 for root dir)

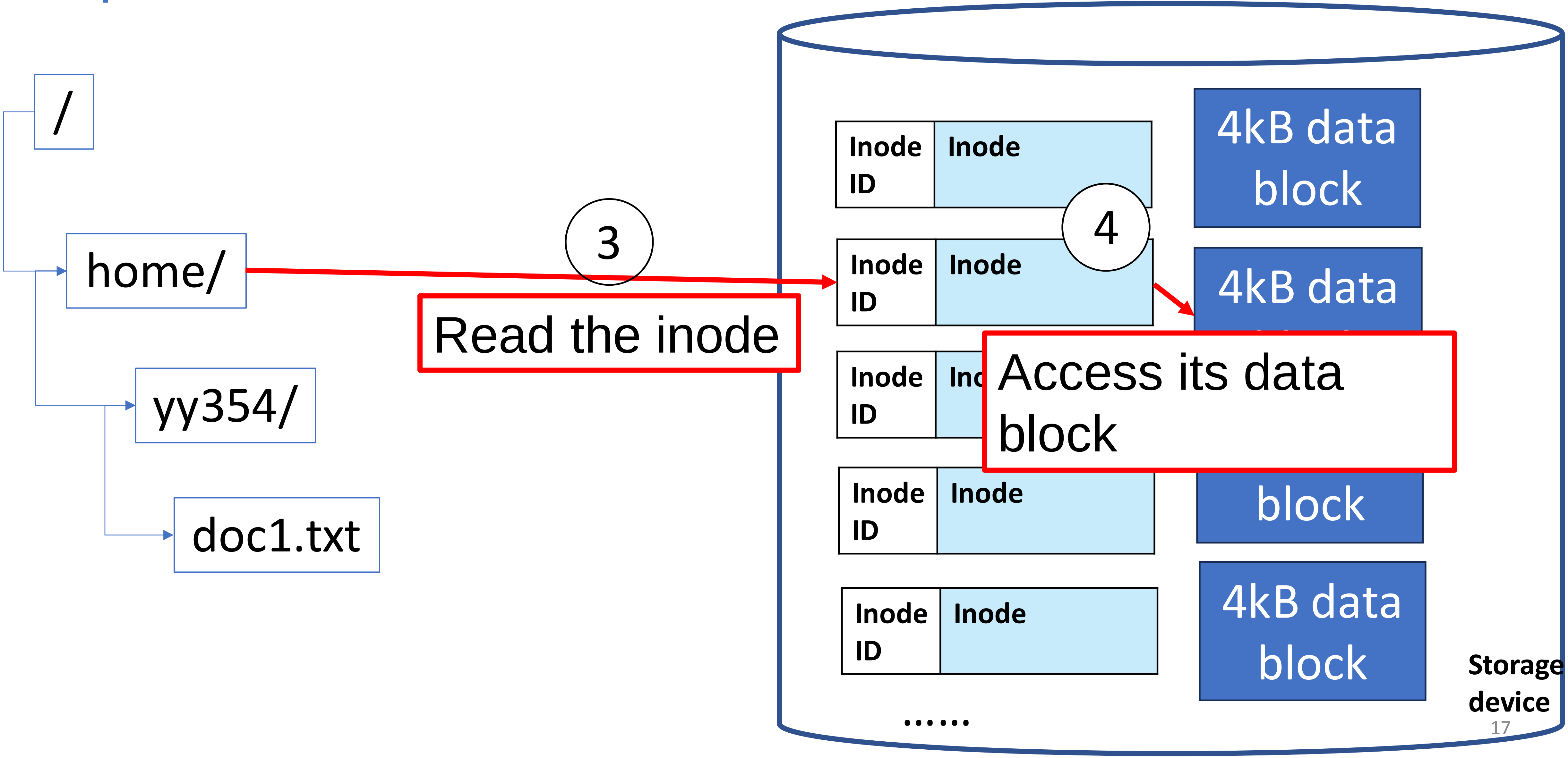
Read its data block

bin/:> inode3
usr/
home/ :> inode5
tmp/
var/
…

Follow the directory

"/home"

Similar for ”/home/yy354/"

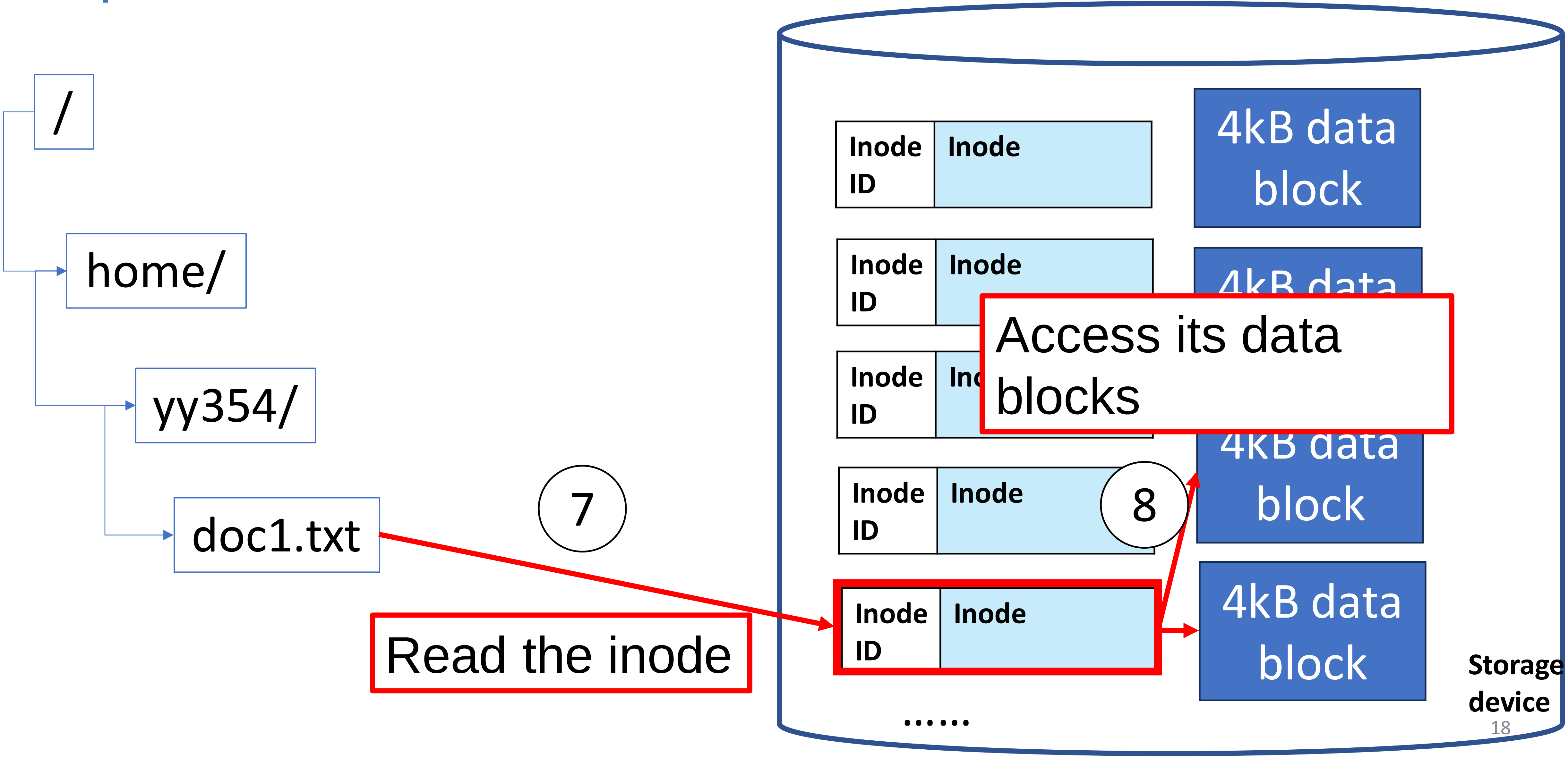


Read the inode

Access its data block

Read the data

"/home/yy354/doc1.txt"



Access its data blocks

Read the inode

File System

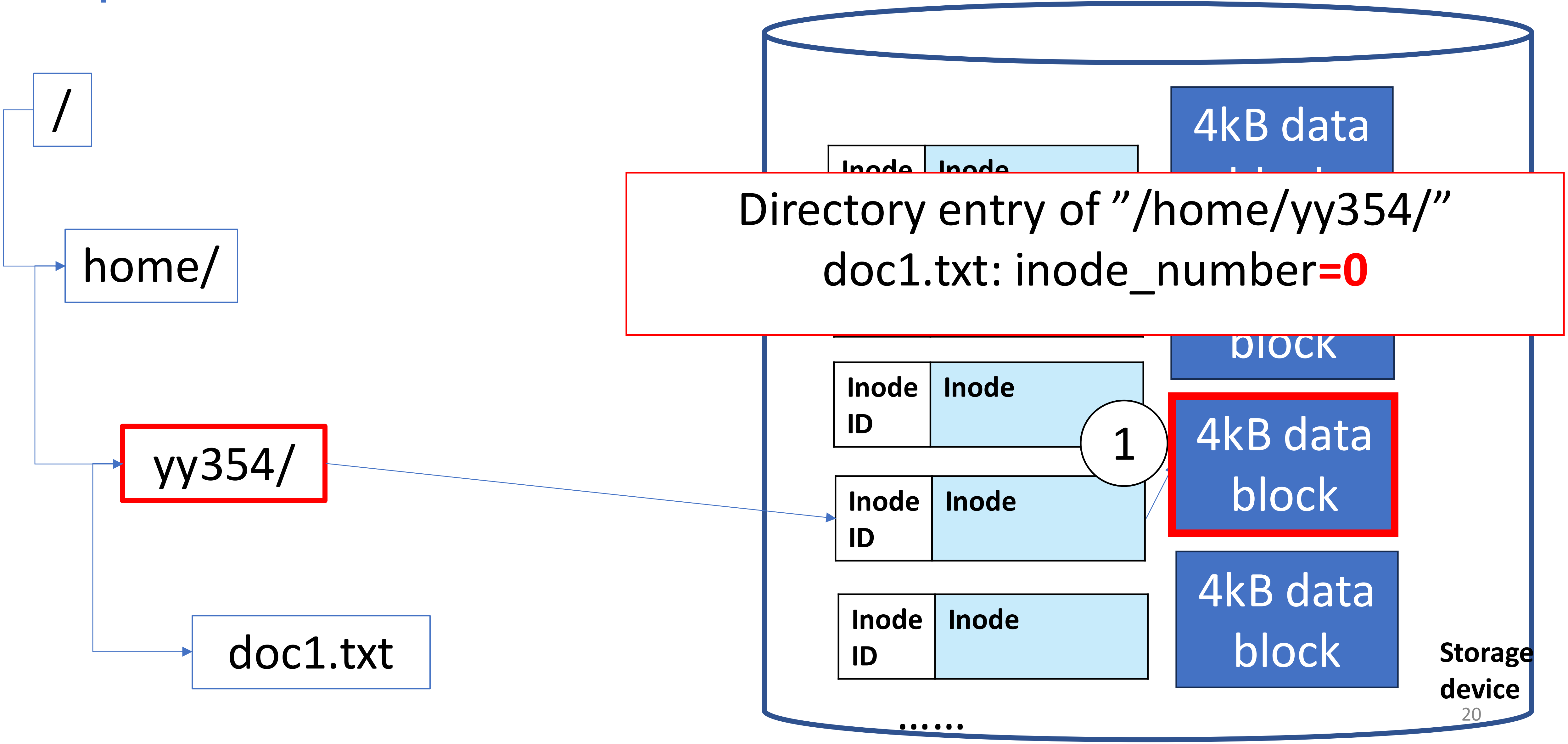
How are files and directories organized?

What happens when you read a file?

What happens when you delete a file?

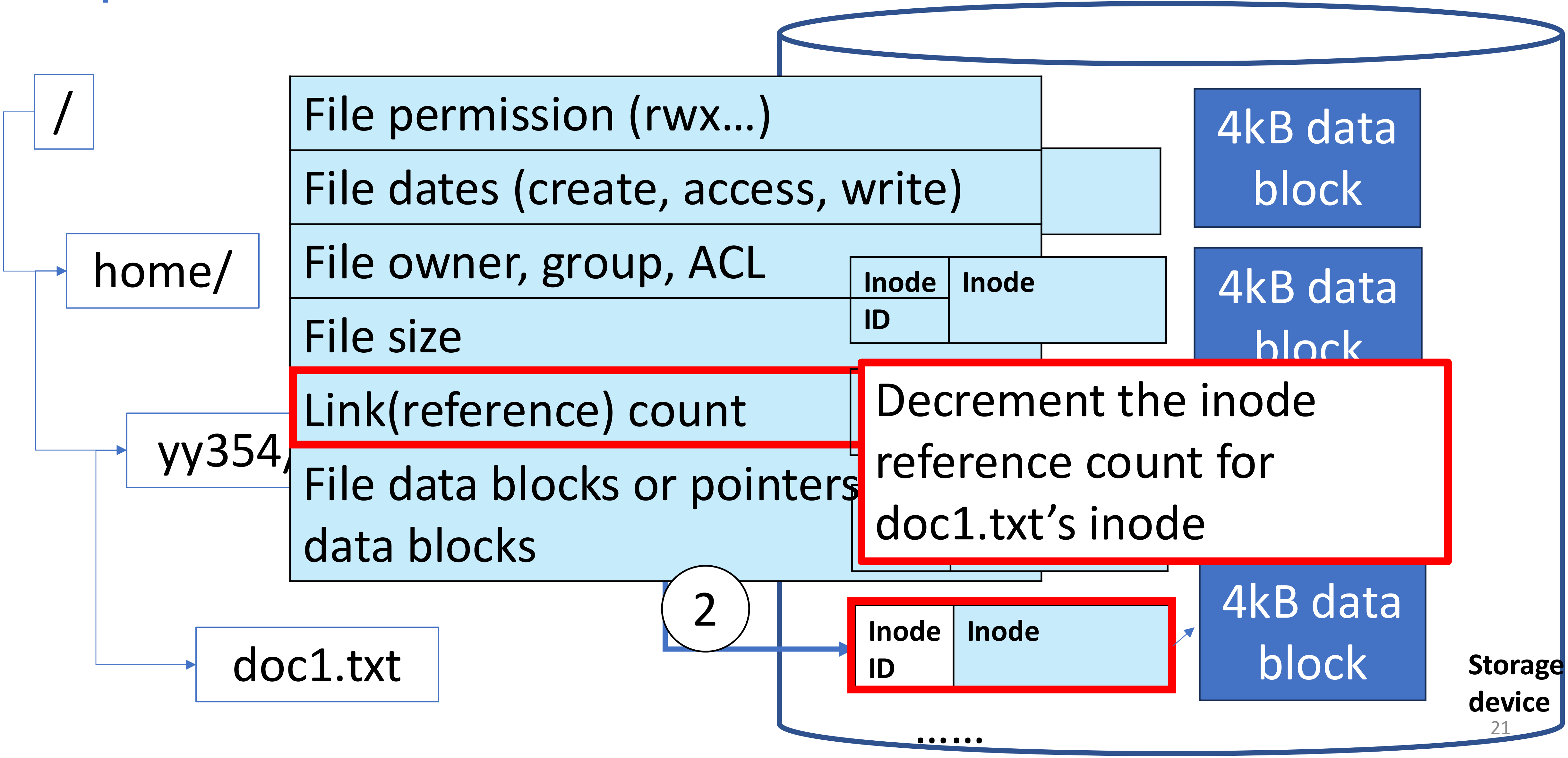
Deleting a file

"/home/yy354/doc1.txt"



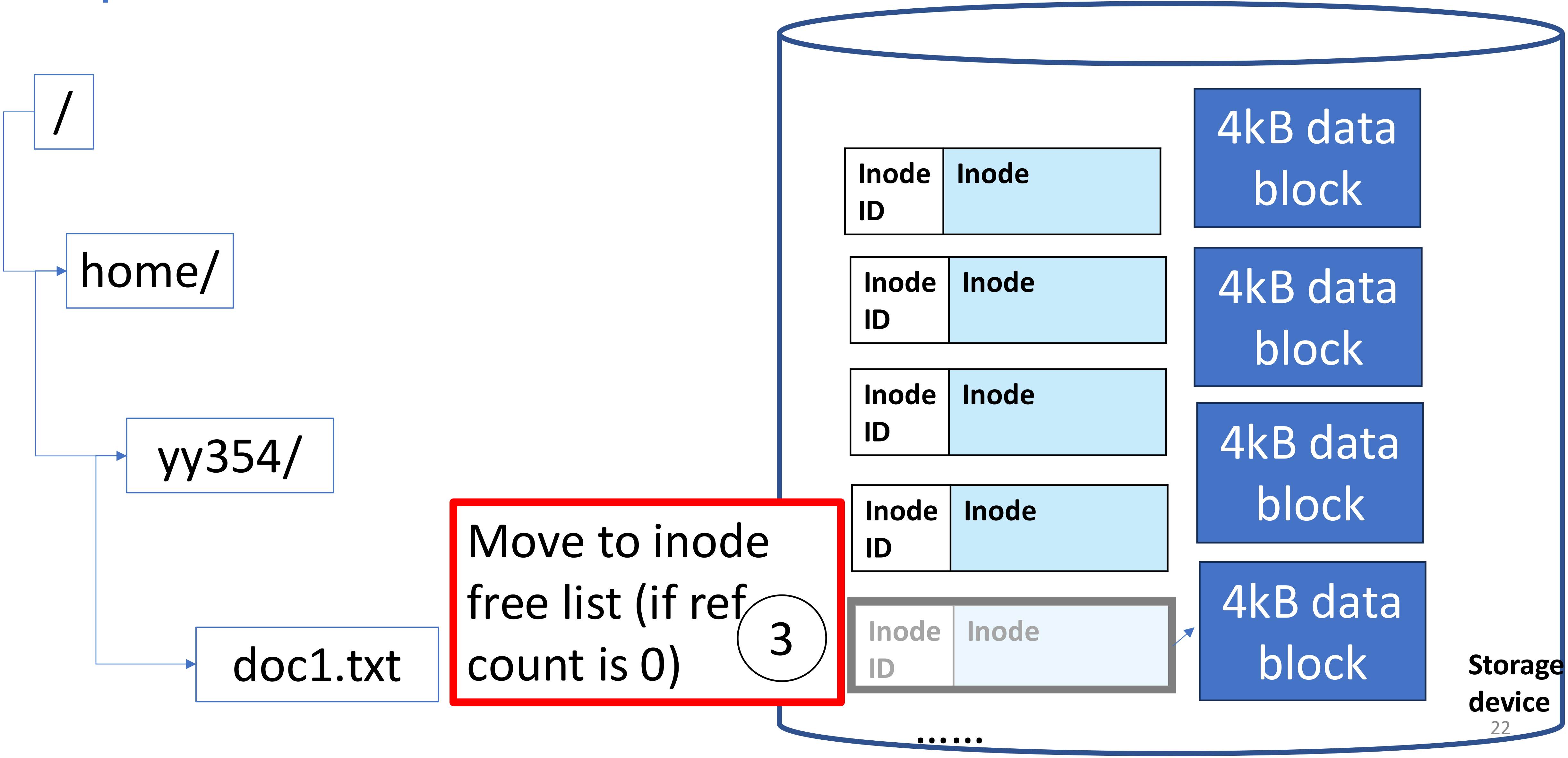
Directory entry of ”/home/yy354/”
doc1.txt: inode_number=0

Decrement inode reference count



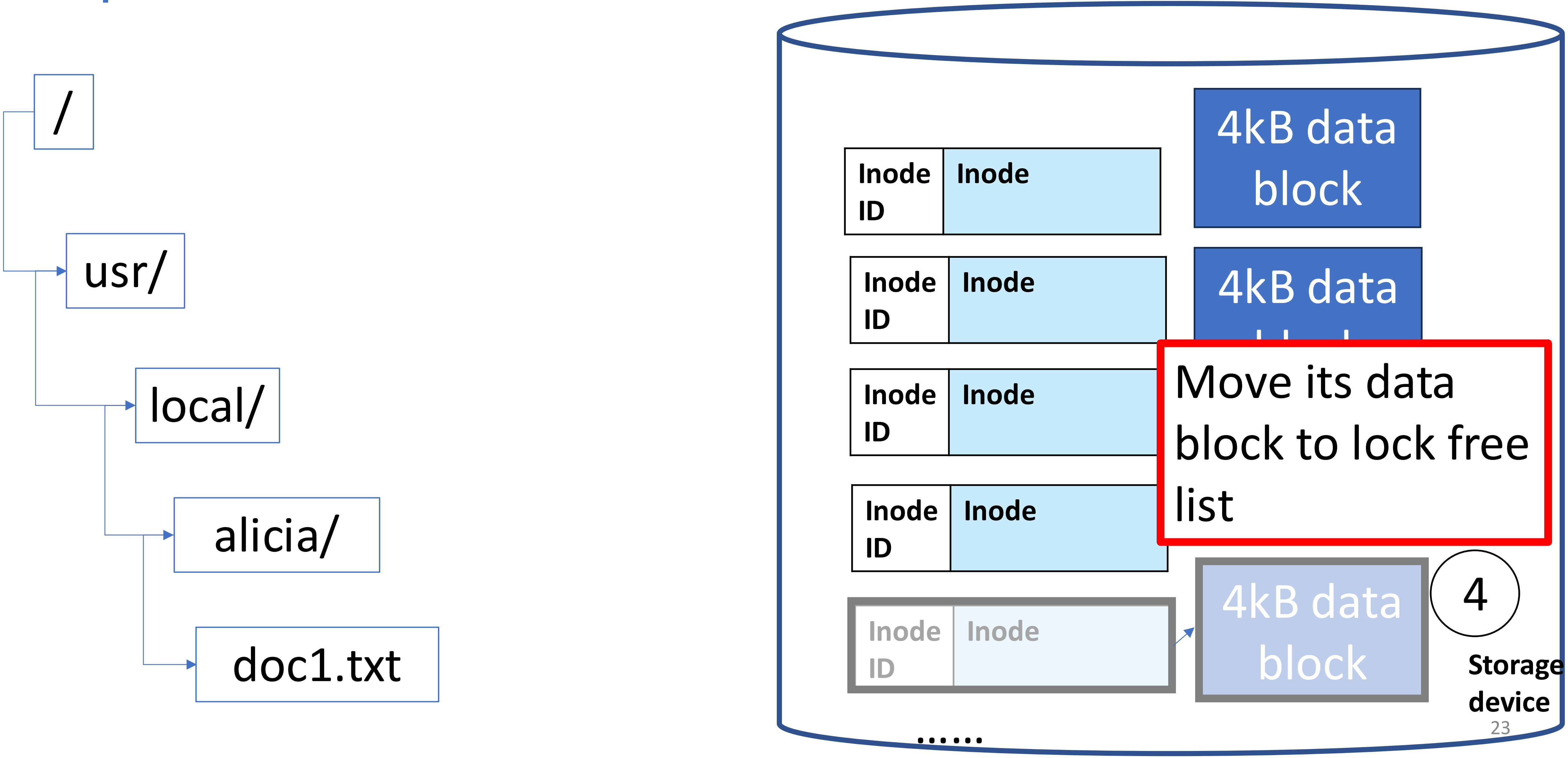
Decrement the inode reference count for doc1.txt's inode

Move inode to inode free list



Move to inode free list (if ref count is 0)

Move inode to inode free list



Move its data block to lock free list

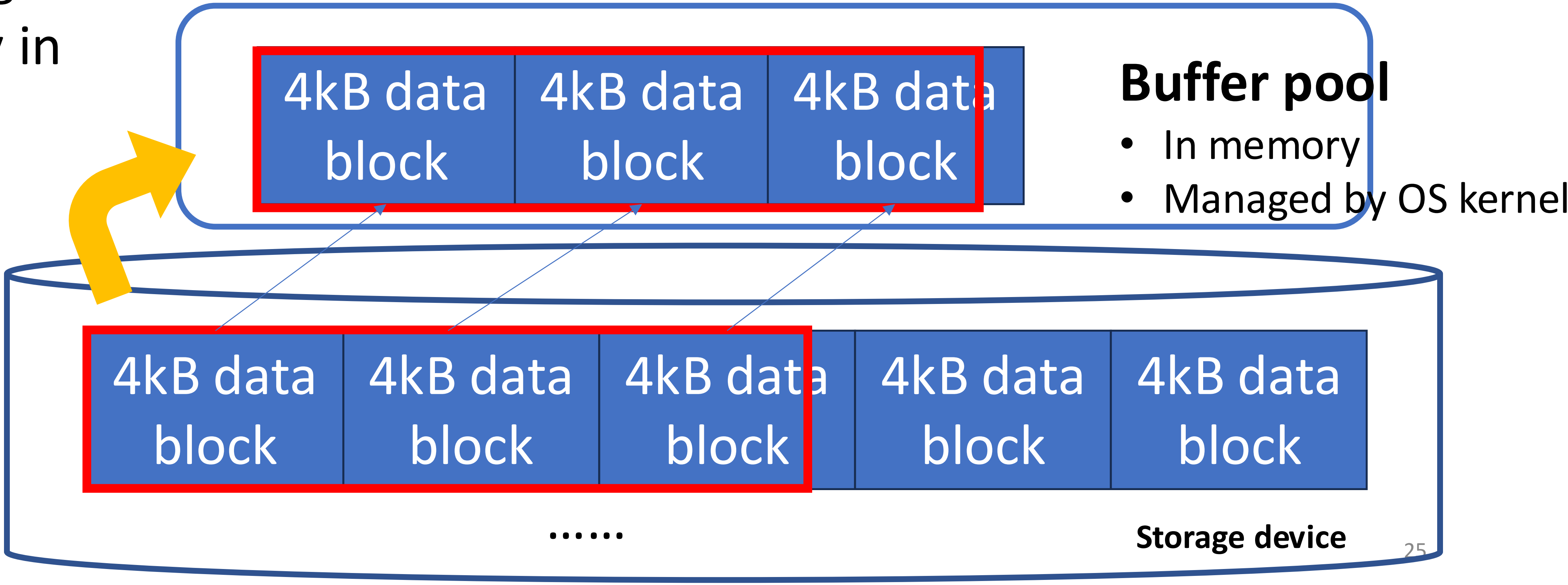
Memory accessing

Reading a file

”/home/yy354/doc1.txt"

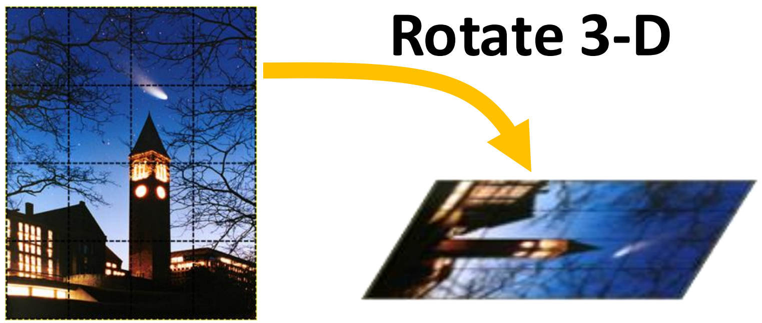
```
std::ifstream file(”/home/yy354/doc1.txt");
std::string line;
std::getline(file, line);
```

Data is fetched from storage to memory in block-sized chunk



Example from lecture

- Consider this photo rotation:



- Does it have embarrassing parallelism in the task?

Photo rotation

Cpp implementation:
https://github.com/aliciayuting/CS4414Demo
Below is pseudo-code for understanding

```
void rotate90Clockwise(src, dst, width, height){
    for (y in height){
        for (x in width){
            dst_x = height - 1 - y
            dst_y = x
            dst_index = (dst_y * new_width + dst_x) * 3

            src_index = (y * width + x) * 3

            dst[dst_index] = src[src_index]
            dst[dst_index + 1] = src[src_index + 1]
            dst[dst_index + 2] = src[src_index + 2]
}}}}
```

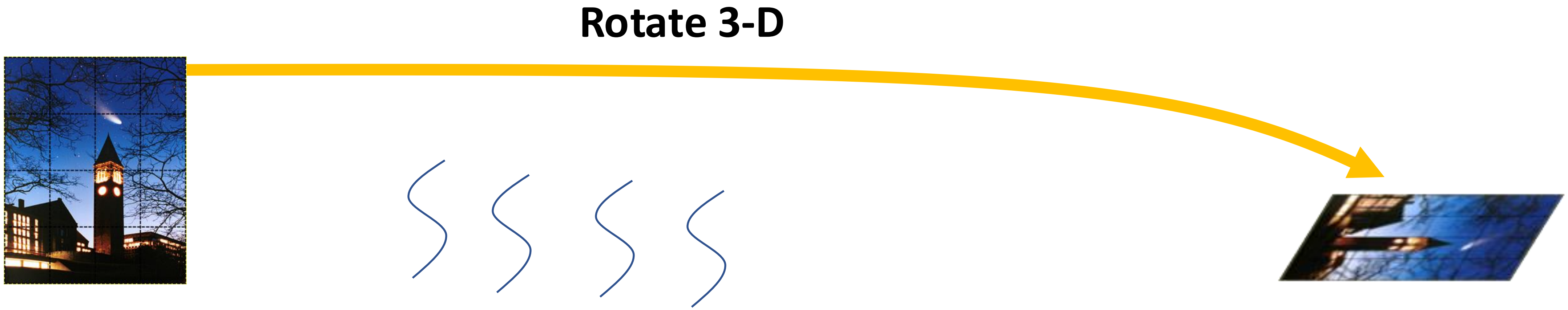
// Calculate the destination coordinates for a 90-degree rotation

// Calculate the source index

// Store rotated RGB values in dst at calculated position

Photo rotation

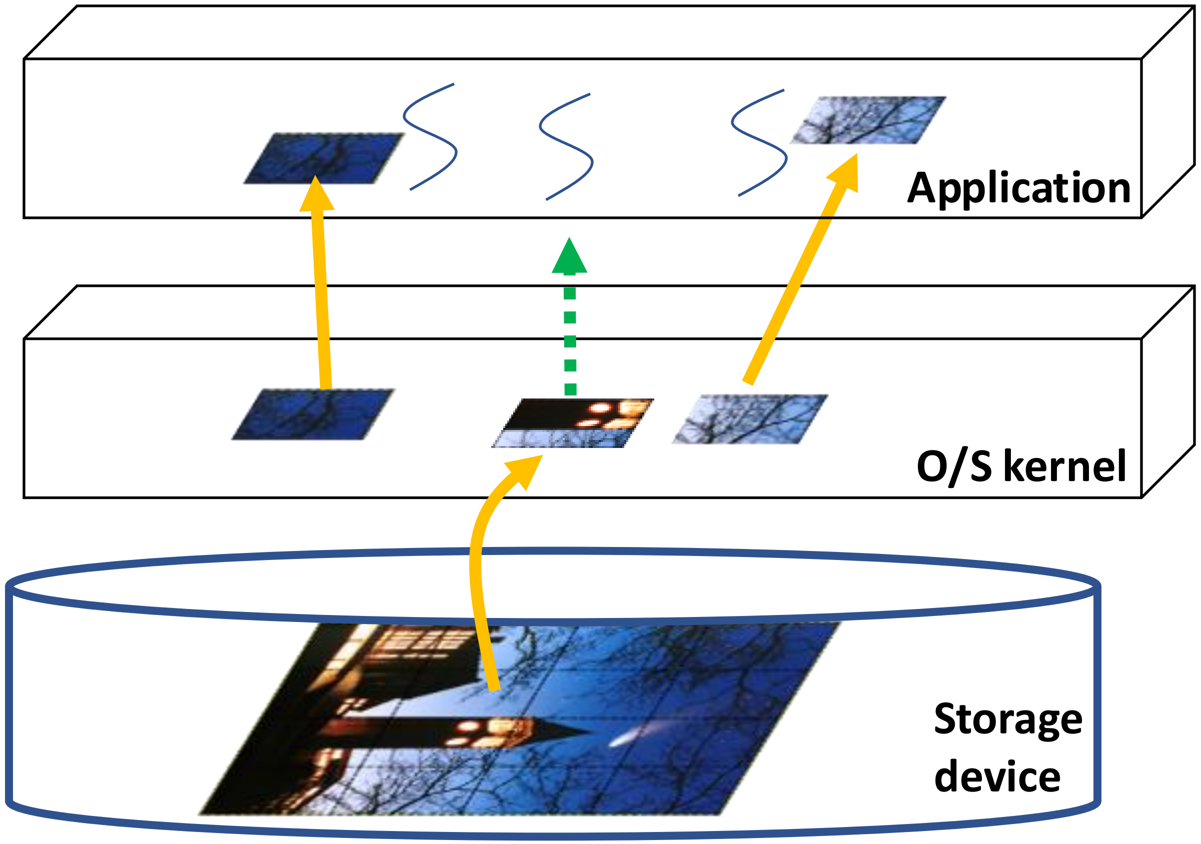
Demo



Rotate 3-D

Multithreading
accessing different part of the memory to parallelize the computation

Opportunity for parallelism



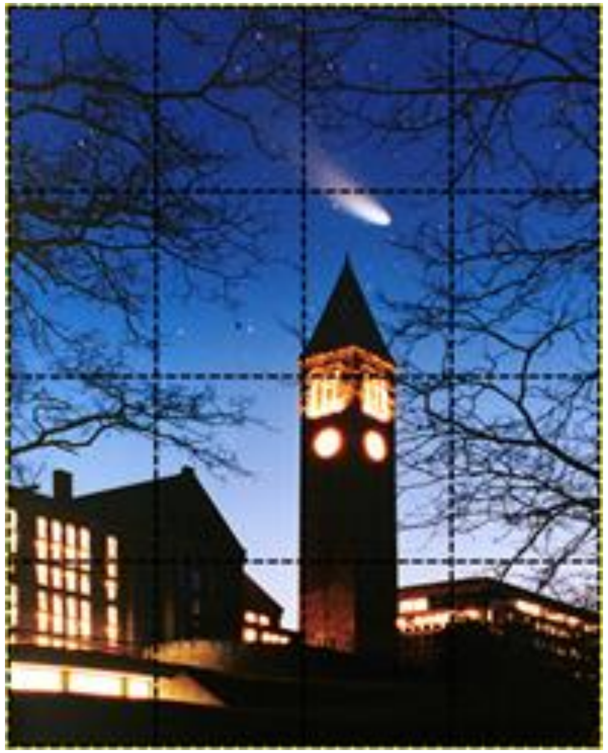
The application has **multiple threads** and they are processing different **blocks**.

The blocks themselves are arrays of pixels. We need to multiply each pixel against a small 4x4 tensor describing the rotation

File system could be doing prefetching

On disk, photo spans many blocks

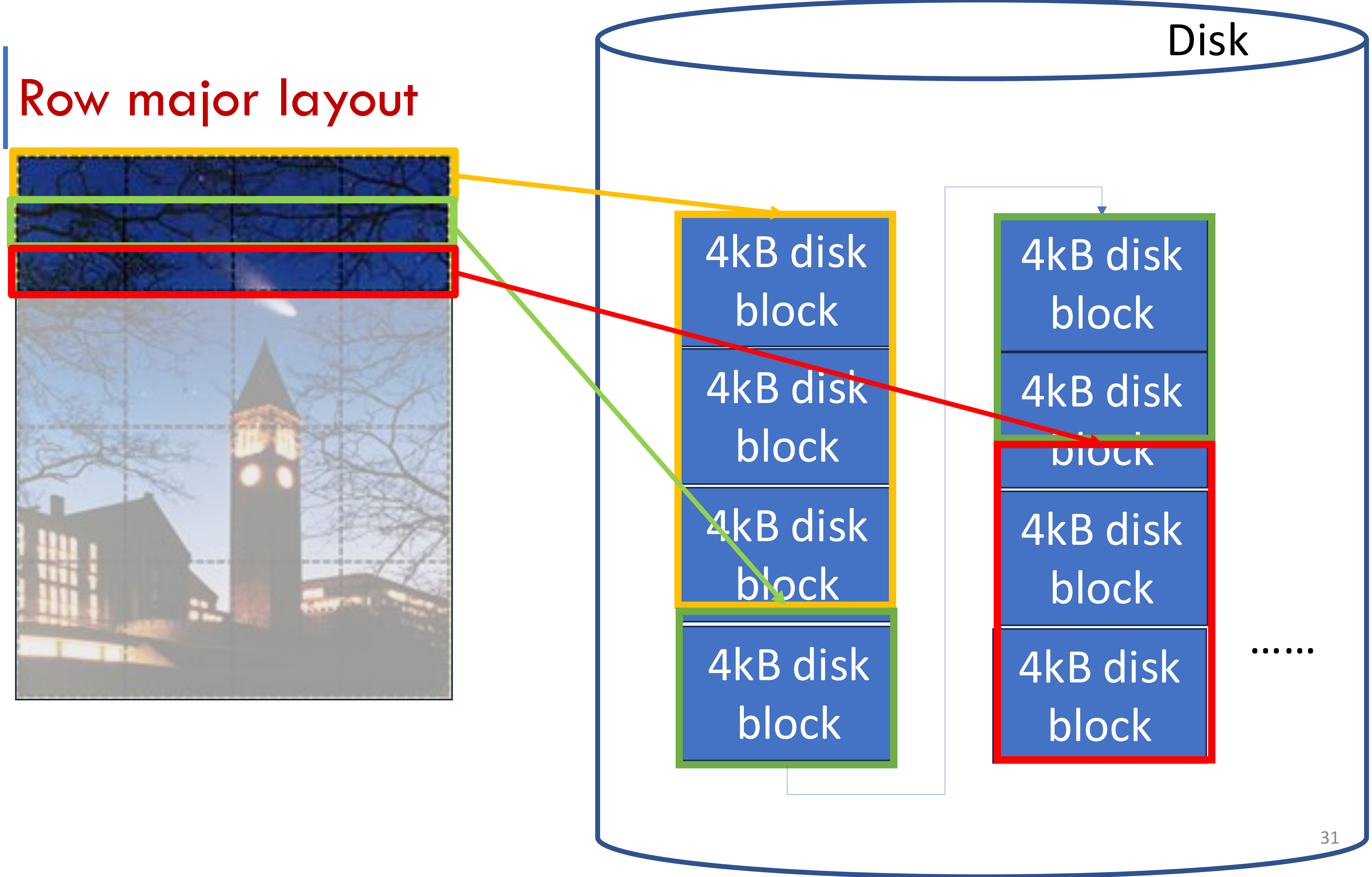
But the example as shown has a gotcha



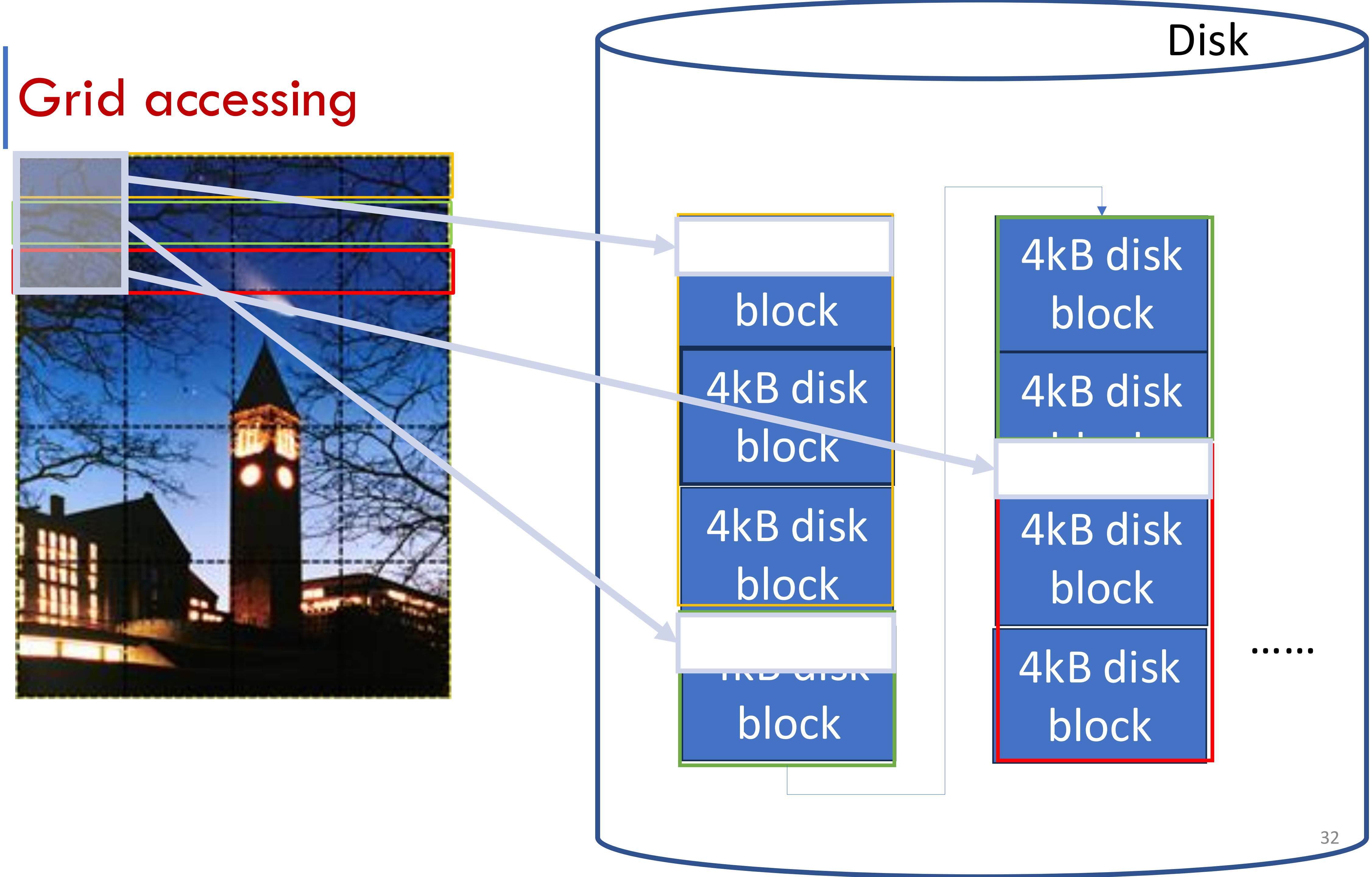
- Are these submatrices actually adjacent data, in the image as held in memory?

- In C++ (like most languages), a matrix is represented in “row major” layout: first all the data in row 0, sequentially, then row 1, etc.

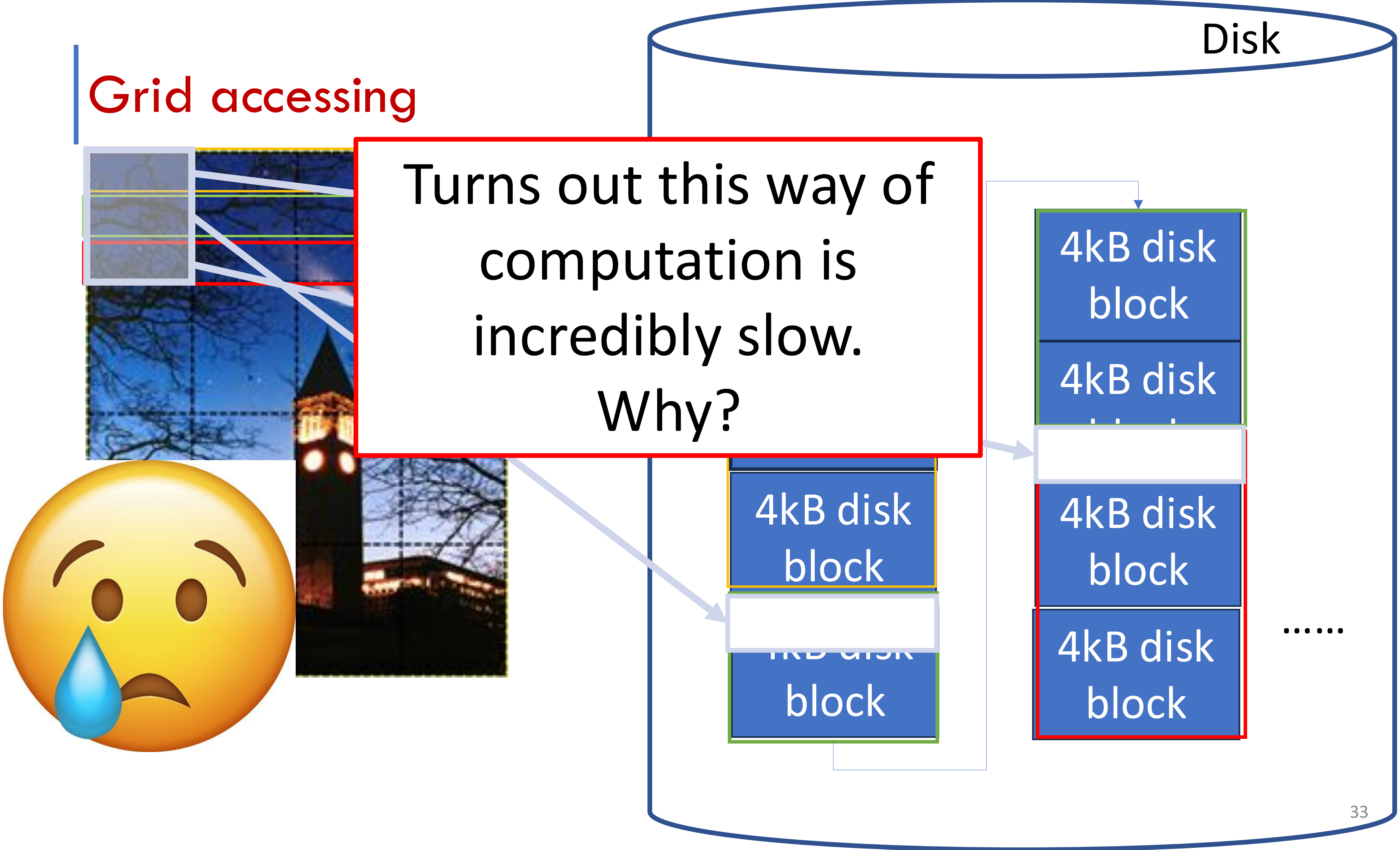
Row major layout



Grid accessing

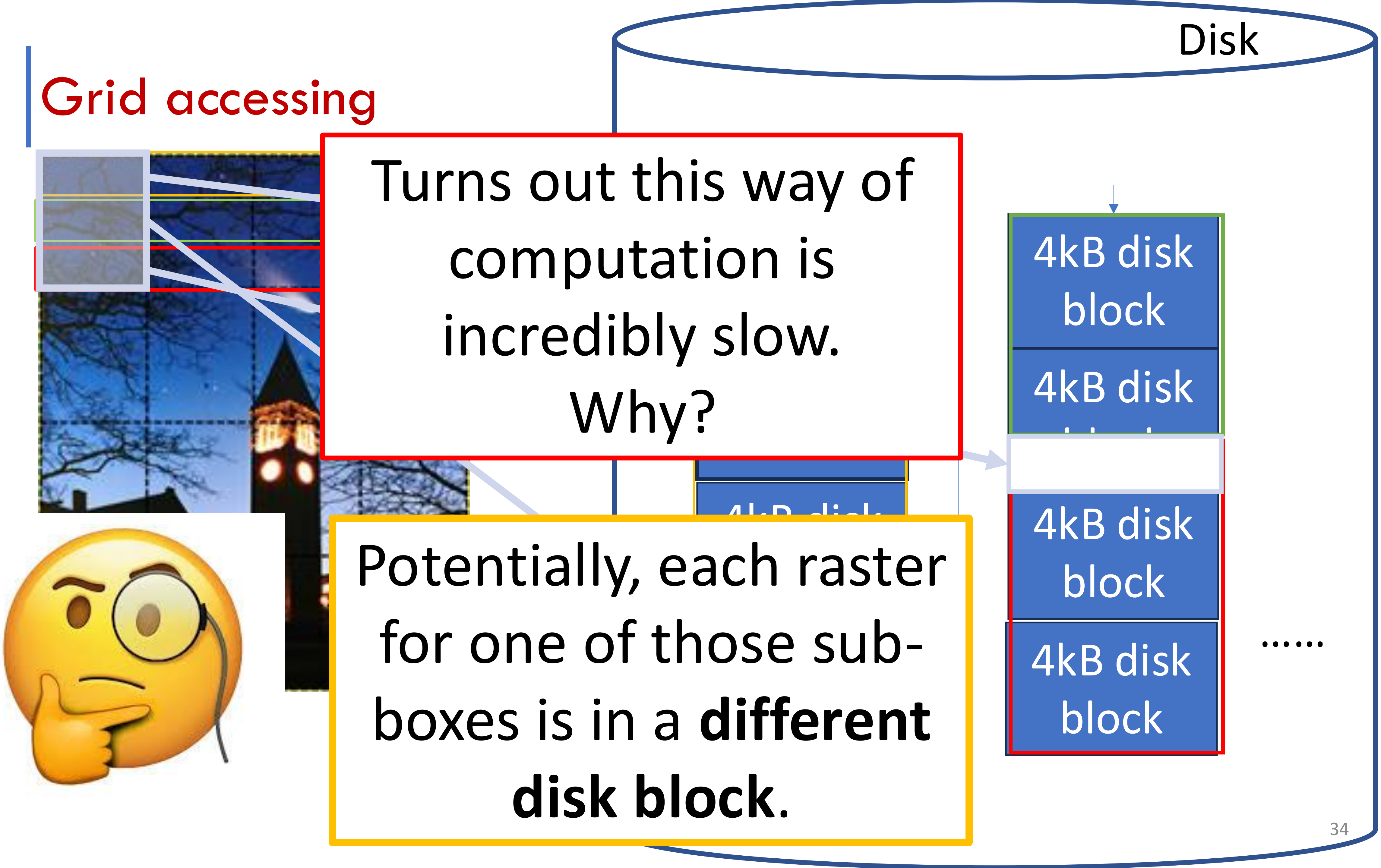


Grid accessing



Turns out this way of computation is incredibly slow. Why?

Grid accessing

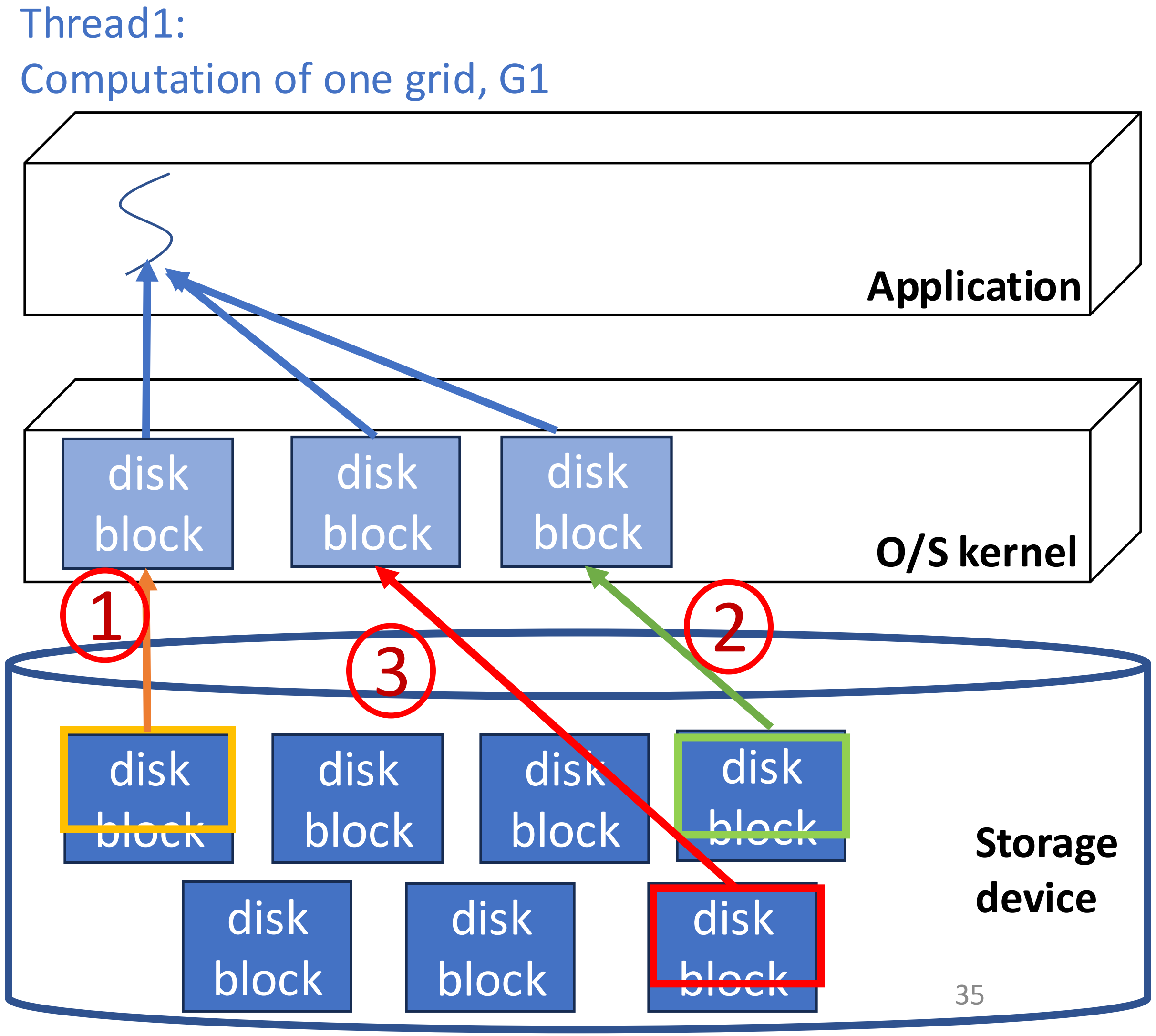


Turns out this way of computation is incredibly slow. Why?

Potentially, each raster for one of those sub-boxes is in a **different disk block**.

Computation --- by Grid

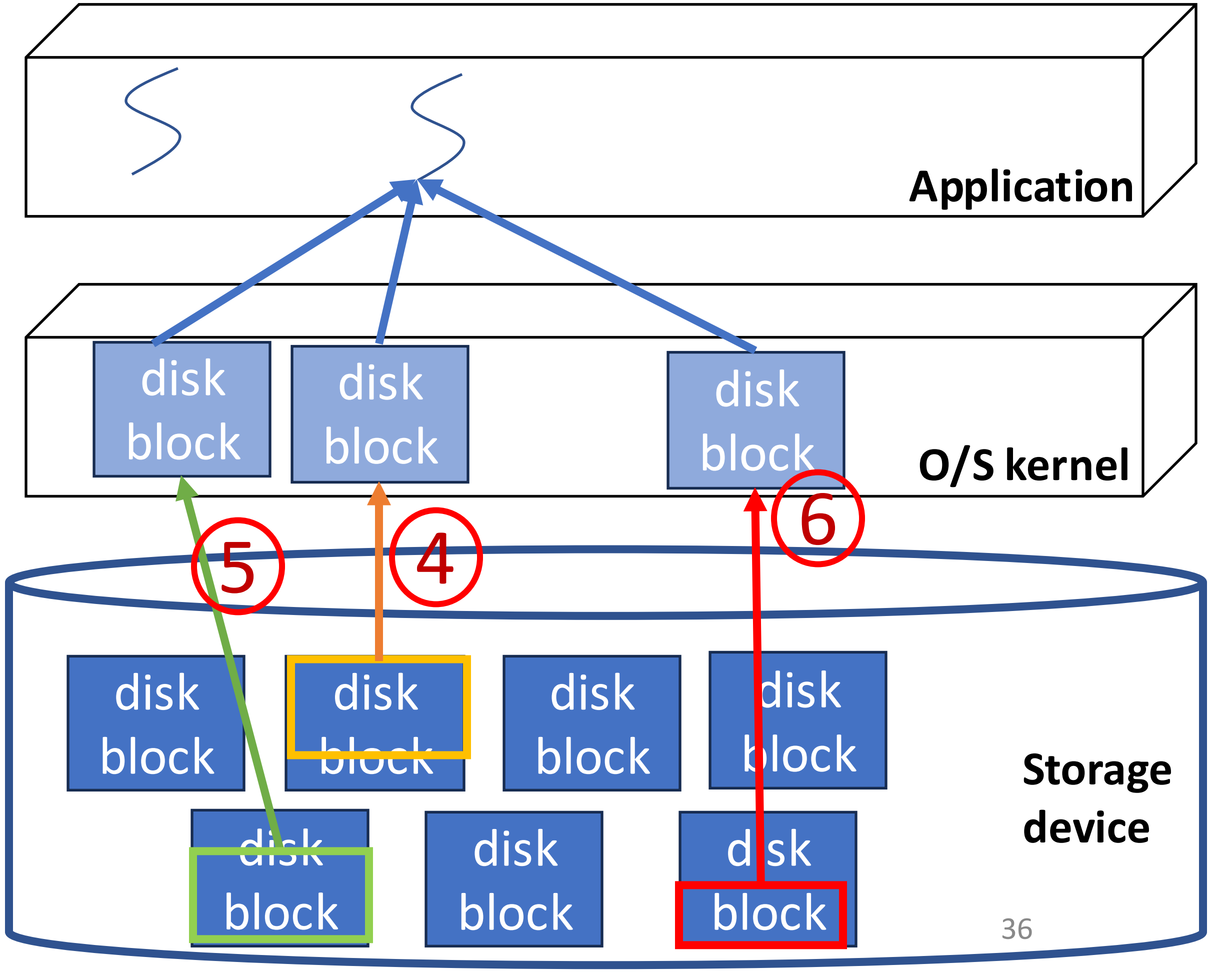
- Hard to prefetch: across multiple non-consecutive disk blocks



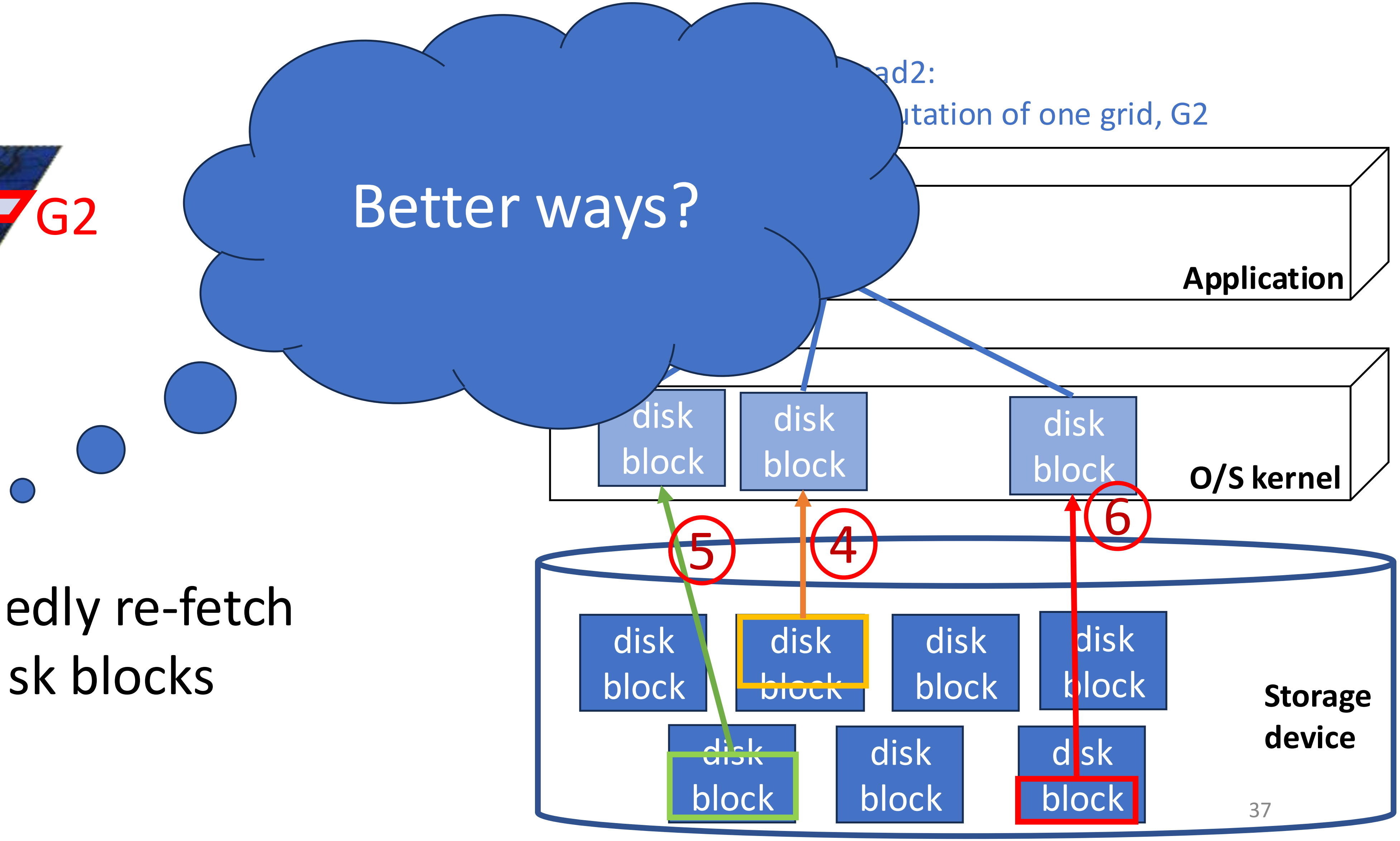
Computation --- by Grid

- Lots of I/O
  May repeatedly re-fetch the same disk blocks

Thread2:
Computation of one grid, G2



Computation --- by Grid

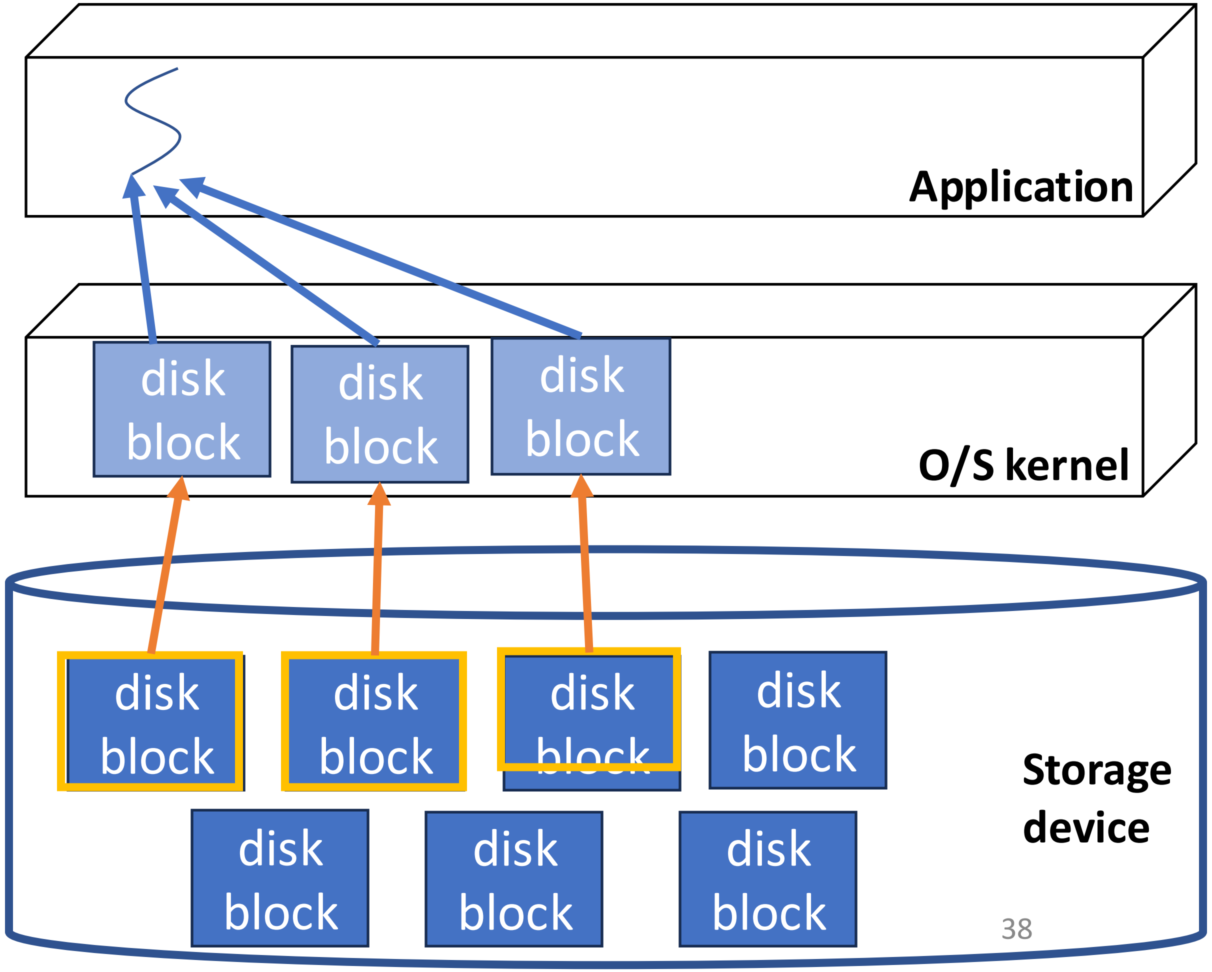


edly re-fetch
sk blocks

Computation --- by **row**-Slice

- Contiguous memory accessing

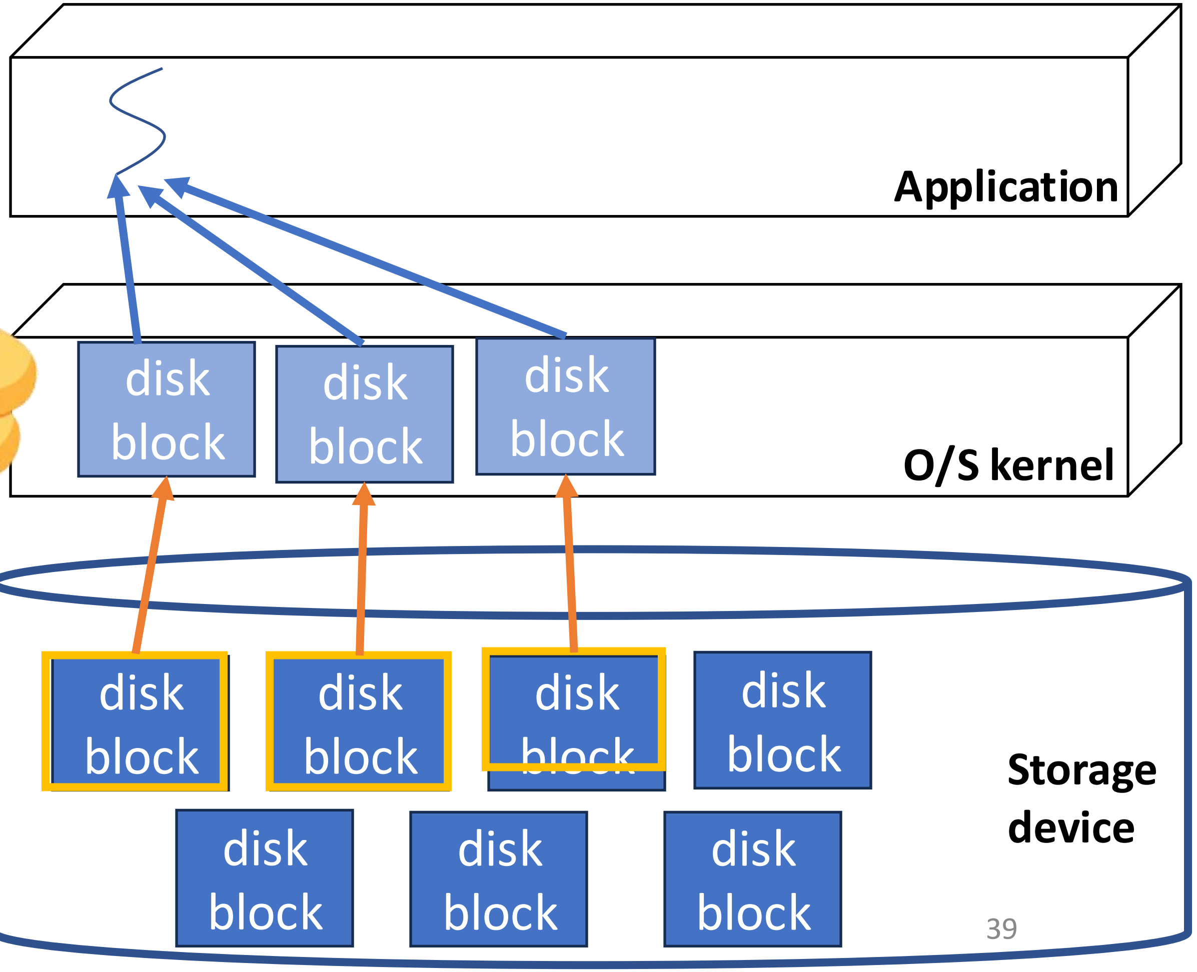
Thread1:
Computation of one row, R1



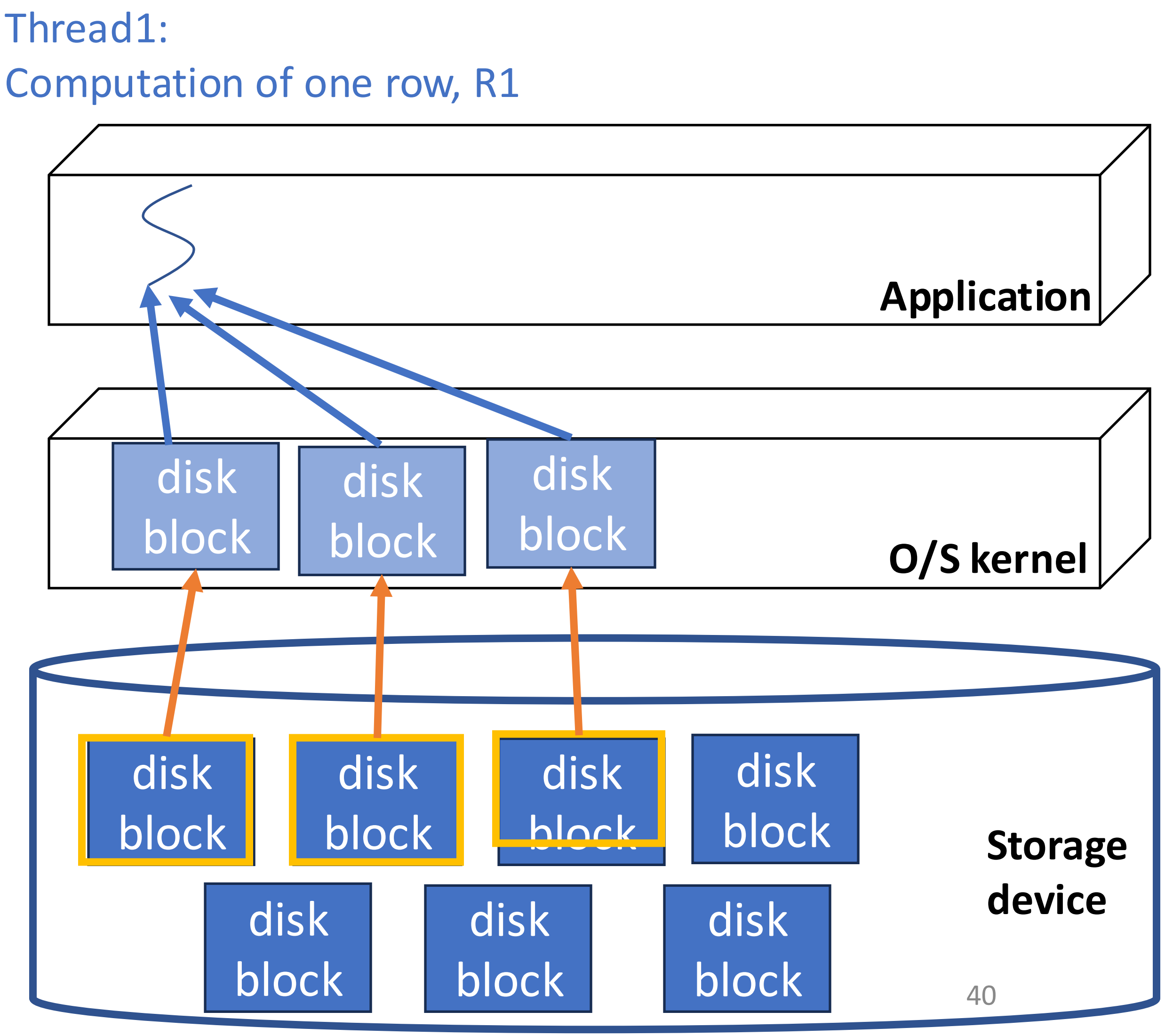
Computation --- by **row**-Slice

- Contiguous memory accessing

Thread1:
Computation of one row, R1



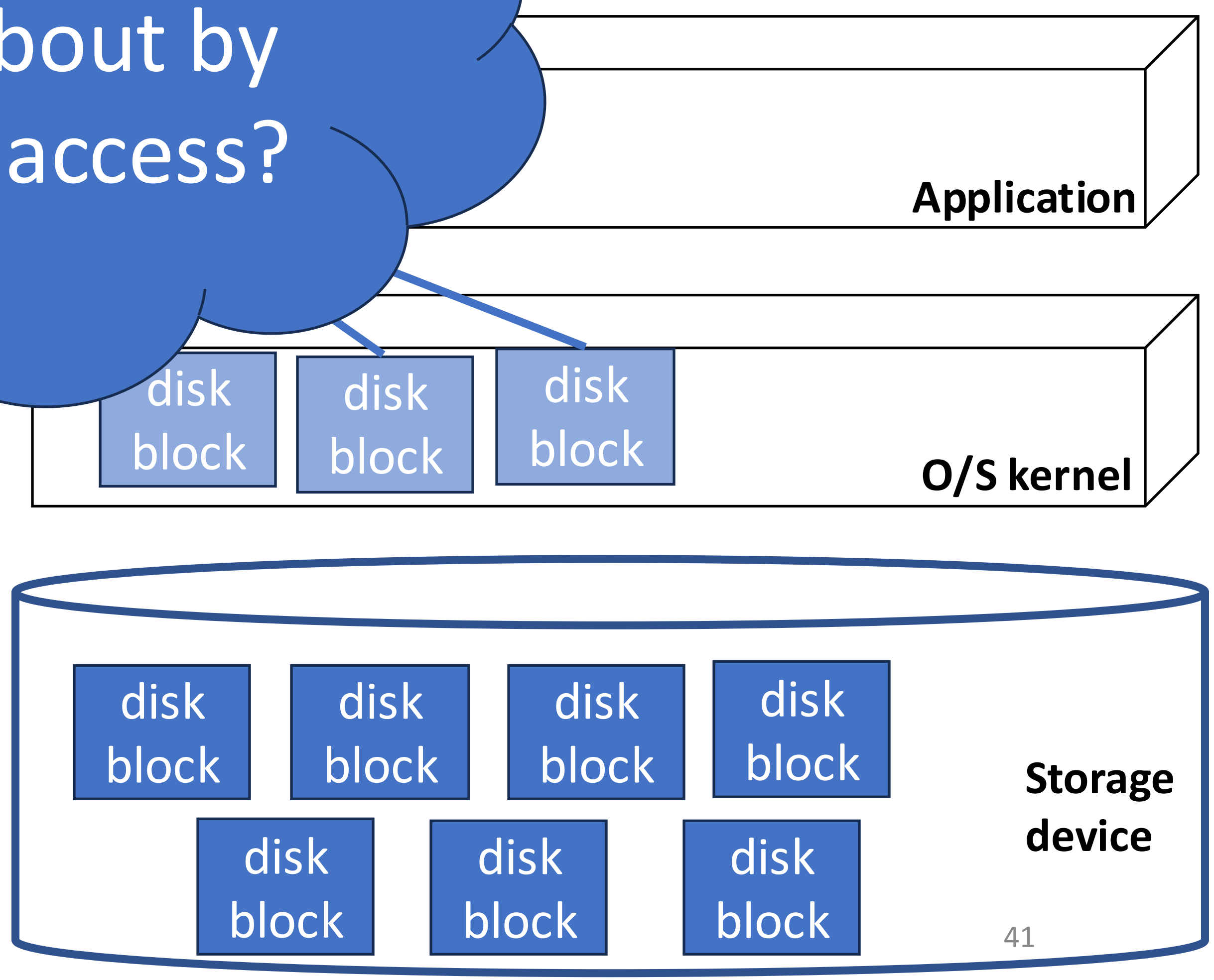
Computation --- by **row**-Slice



- Prefetch the data block
    - While it is processing b, Linux would prefetch b+1 and b+2

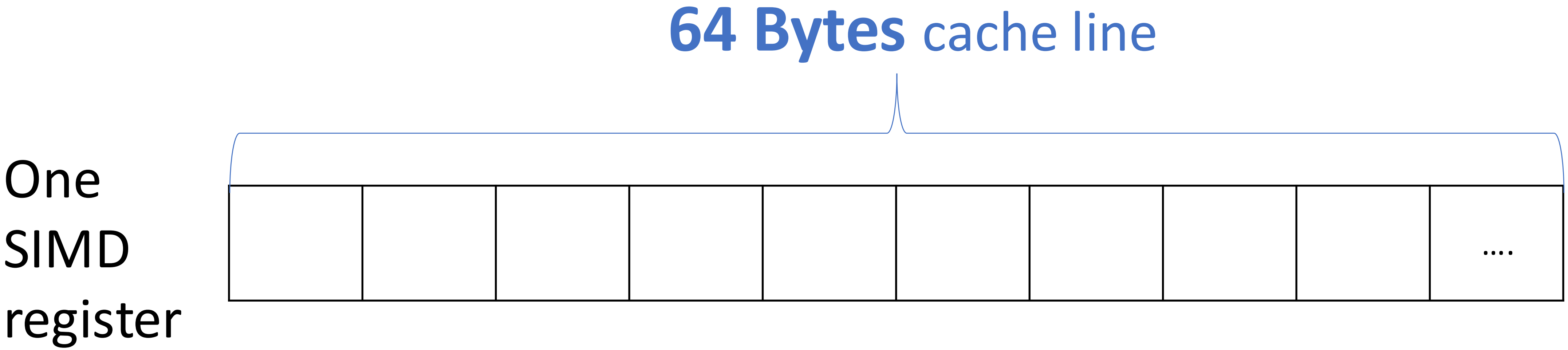
Computation --- by **column**-Slice

What about by column access?



MMX

Bool matrix with SIMD



64 Bytes cache line

One SIMD register

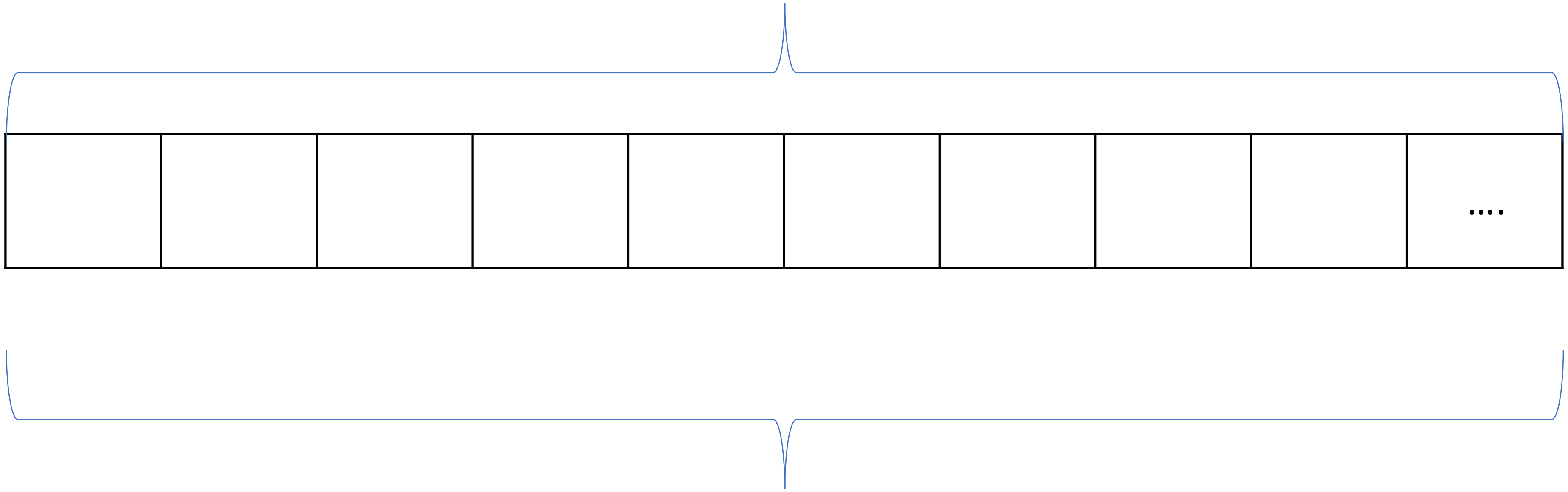
....

Bool = {0,1}
could be represented using 1 bit

Bool matrix with SIMD

64 Bytes cache line

One SIMD register



64 * 8 = 512 bit

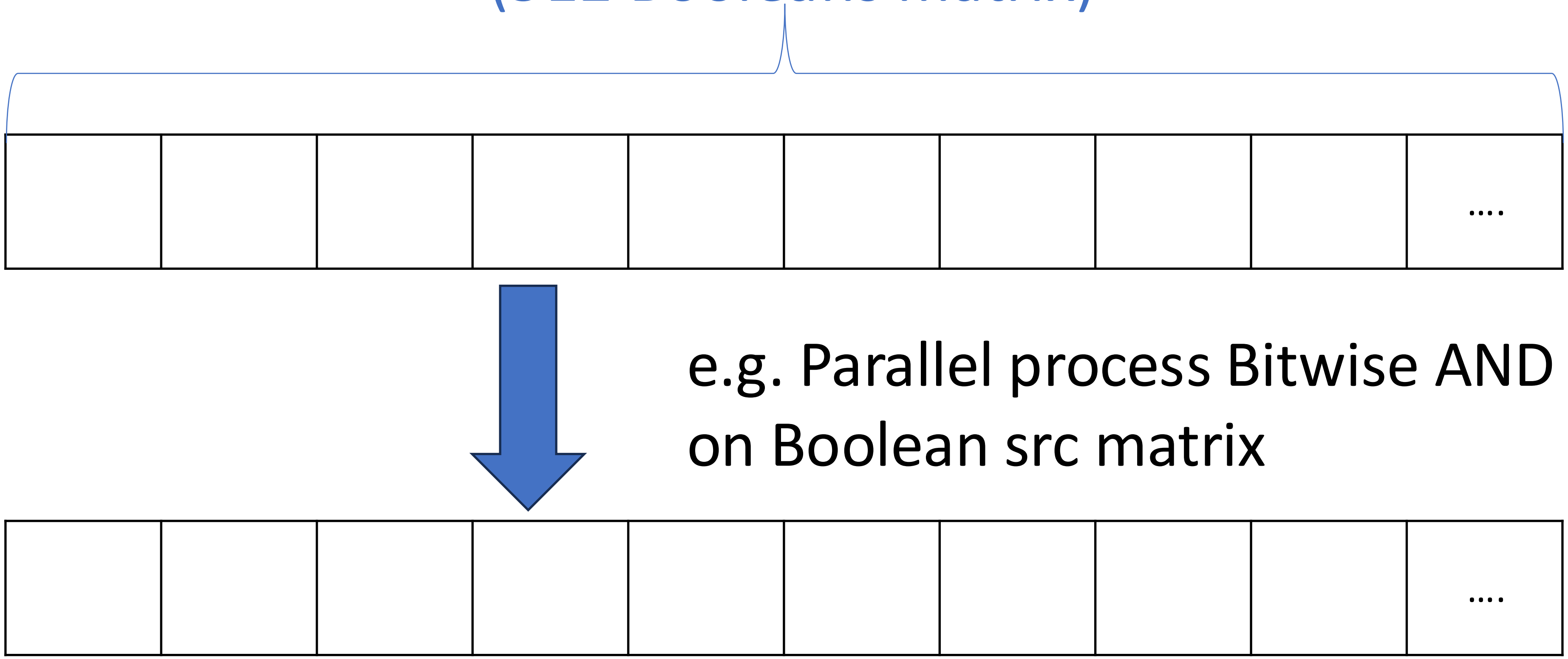
Could represent 512 Booleans at a time

Bool = {0,1}
could be represented using 1 bit

Bool matrix with SIMD

64 Bytes cache line
(512 Booleans matrix)

SIMD
R1 (src)



e.g. Parallel process Bitwise AND operation
on Boolean src matrix

SIMD
R2 (dst)

x 512 Speedup

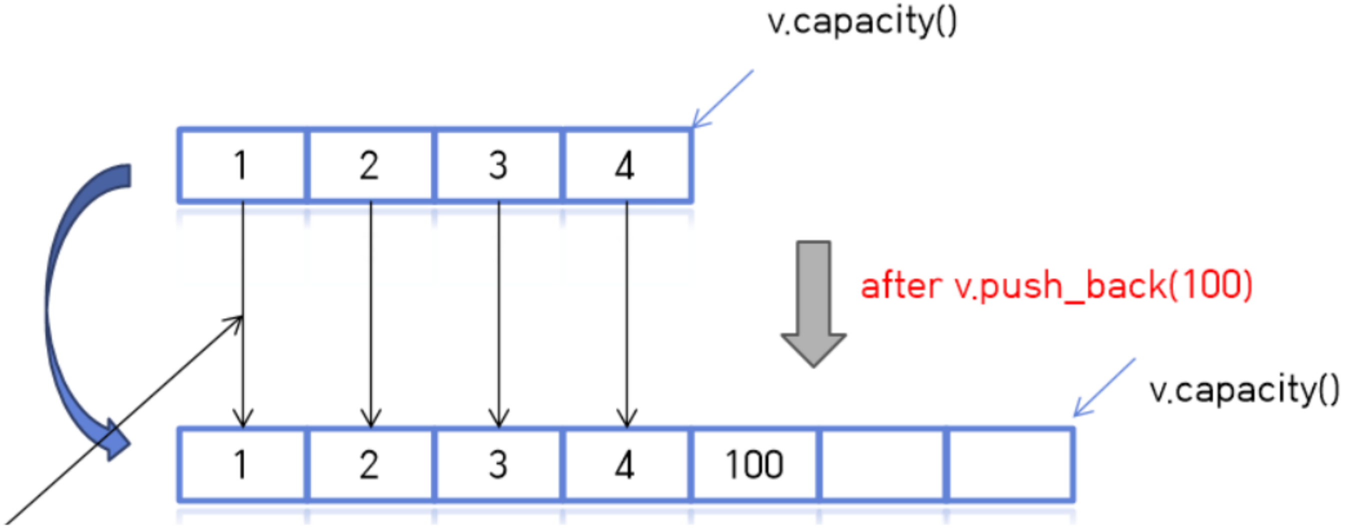
Bool matrix with SIMD

```
void bitwise_and_512bit(const uint8_t* a, const uint8_t* b, uint8_t* result) {
for (int i = 0; i < 64; ++i) {
        result[i] = a[i] & b[i];      // Iterate over each byte (64 bytes total)
    }                                 and perform bitwise AND
}
int main() {
    uint8_t a[64] = {….}; // 512 bits packed into 64 bytes
    uint8_t b[64] = {….}; // 512 bits packed into 64 bytes
    uint8_t result[64]
    bitwise_and_512bit(a, b, result);
…}
```

Why SIMD with std::vector could get a bit tricky?

std::vector<T> - A dynamic-sized array

- Concept of size vs. capacity (std::vector capacity >= size)
- Reallocates elements when capacity is exceeded



Reference & pointer with container

Data copy in code

- Explicit calling copy-constructor (copy-assignment)
- Function parameter pass by value
- Iterate over values in std containers
- …

Copy constructor

```
std::vector<Cat> cat_vec;

cat_vec.emplace_back(”fluffy”, 2);

std::vector<Cat> cat_vec2;

cat_vec2.emplace_back(”sally”, 2);
```

How many catsin cat_vec?
What about cat_vec2?

Prelim1 Question 5

```
class Kitten{
public:
        std::string name;
        …
};
```

```
class Cat{
public:
        std::vector<Kitten> litter;
        Cat();
        Cat(std::vector<Kitten> l);
        ~Cat();
        …
};
```

Function Parameter

--- Passing value

- When a vector value is passed to a function, a copy of the vector is created.

```
void add_to_litter(Cat c, Kitten k){
        c.litter.push_back(k);
}

int main(){
        Cat c1;
        Kitten k1;
        c1.add_to_litter(k1);
…}
```

← Passing a copy of Cat object to a function:

changes made inside the function are not reflected outside

Function Parameter

--- Passing reference

```
void add_to_litter(Cat& c, Kitten& k){
        c.litter.push_back(k);
}

int main(){
        Cat c1;
        Kitten k1;
        c1.add_to_litter(k1);
…}
```

← Passing a reference of Cat object to

a function:

changes made inside the function

persist to the argument that passed in

Member function

```
class Cat{
public:
        std::vector<Kitten> litter;
        Cat(std::vector<Kitten> l){…}
        void add_to_litter(Kitten k){
                litter.push_back(k);
        }
};
```

← A copy of the Kitten object from argument is added to this Cat object

Iterate in std::container --- value

```
std::vector<Cat> cat_vec;
cat_vec.emplace_back(”fluffy”, 2);
cat_vec.emplace_back(”sally”, 2);
for (Cat cat: cat_vec){
        Kitten k;
        cat.add_to_litter(k);
}
```

Will this add Kitten k to each cat in cat_vec?

Iterate in std::container ---reference

```
std::vector<Cat> cat_vec;
cat_vec.emplace_back(”fluffy”, 2);
cat_vec.emplace_back(”sally”, 2);
for (Cat& cat: cat_vec){
        Kitten k;
        cat.add_to_litter(k);
}
```

What about this?

Iterate in std::container --- index

```
std::vector<Cat> cat_vec;
cat_vec.emplace_back(”fluffy”, 2);
cat_vec.emplace_back(”sally”, 2);
for (size_t i=0; i<cat_vec.size(); ++i){
        Kitten k;
        cat[i].add_to_litter(k);
}
```

What about this?

Iterate in std::container --- iterator

```
std::vector<Cat> cat_vec;
cat_vec.emplace_back(”fluffy”, 2);
cat_vec.emplace_back(”sally”, 2);
for (auto it=cat_vec.begin(); it!=cat_vec.end(); ++it){
        Kitten k;
        it->add_to_litter(k);
}
```

What about this?