Georgia Tech Computer Science

CS2200
Systems and Networks
Spring 2024

Lecture 28:
File Systems

Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

Lecture slides adapted from Bill Leahy, Charles Lively, Umakishore Ramachandran of Georgia Tech

CIOS is open

- Your feedback helps me improve cs2200!
  - Your future peers will be thankful
  - What worked well? Suggestions for improvement?

- CIOS participation incentive: entire class gets
  - 0.5% bonus to total grade if we reach 85% participation
  - 1% bonus to total grade if we reach 95% participation

Agenda

Agenda

- ~~Disk capacity, performance, and scheduling~~
  - ~~Chapter 10~~

- File Systems
  - Chapter 11
  - Last cs2200 topic 🙌🏻

- It's Pet-a-Disk day!

In this chapter

- Disk space allocation policies

In this chapter

- Disk space allocation policies
  - Figures of merit

In this chapter

- Disk space allocation policies
  - Figures of merit
  - Unix file system

In this chapter

- Disk space allocation policies
  - Figures of merit
  - Unix file system
- Similarities to memory management

In this chapter

- Disk space allocation policies
  - Figures of merit
  - Unix file system
- Similarities to memory management

	Namespace	Physical entity
Virtual memory	Virtual address space	Physical memory
File system	File name	Disk blocks

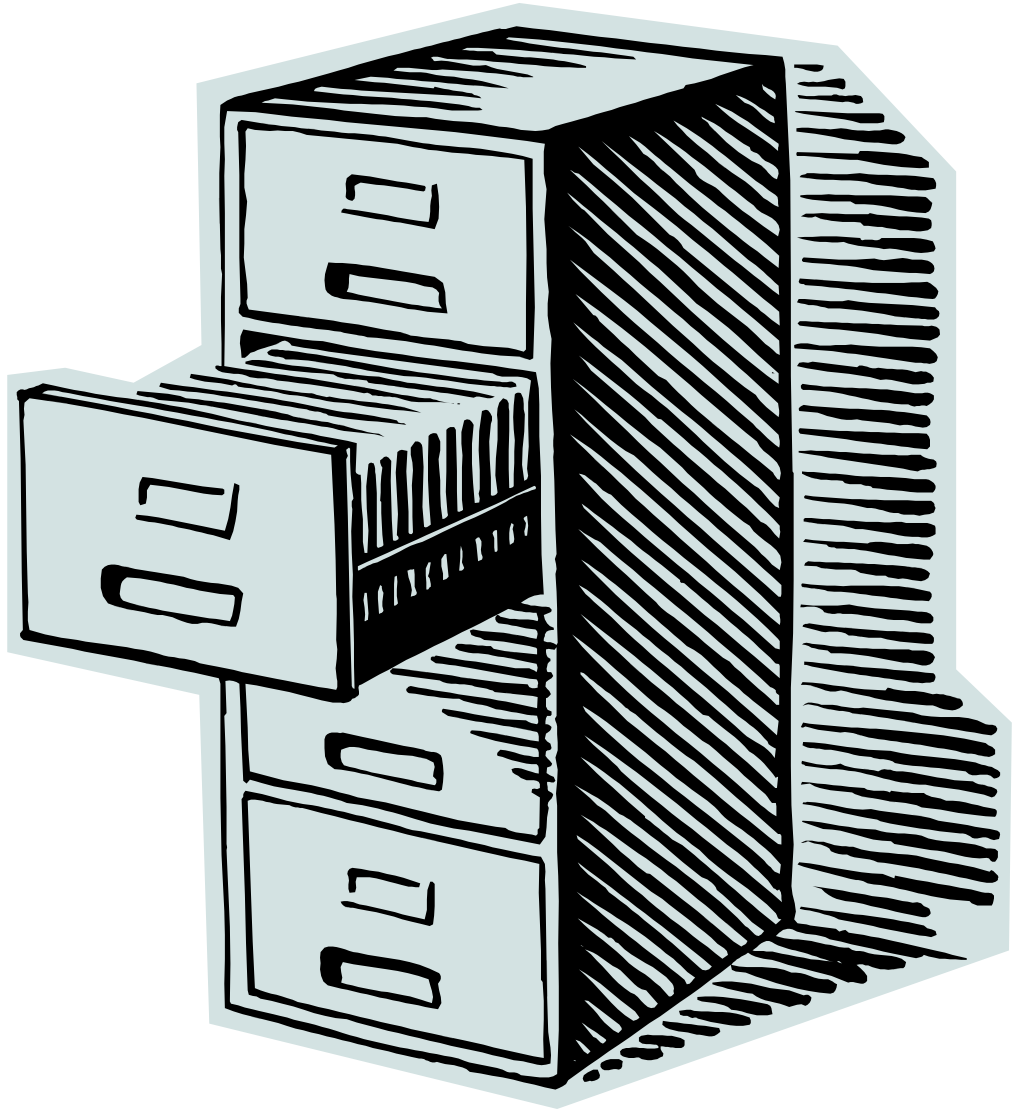
In this chapter

- Disk space allocation policies
  - Figures of merit
  - Unix file system
- Similarities to memory management

	Namespace	Physical entity
Virtual memory	Virtual address space	Physical memory
File system	File name	Disk blocks

- Disk block: unit of disk management for the OS
  - May consist of one or more disk sectors

File System Abstraction



Logical block addresses

- We're going to adopt the modern convention of converting cylinder/track/sector into a logical block address (LBA)
- That is, we calculate an integer 0 to n-1 so that the disk appears to be a simple array of disk blocks
- Modern disk controllers present this abstraction to the CPU to prevent it from having to know the geometry of the physical disk drives

File allocation schemes on the disk

- Contiguous allocation
- Contiguous allocation with overflow
- Linked allocation
- FAT
- Indexed allocation
- Multilevel indexed
- Hybrid indexed

File allocation schemes on the disk

- Contiguous allocation
- Contiguous allocation with overflow
- Linked allocation
- FAT
- Indexed allocation
- Multilevel indexed
- Hybrid indexed

Functionality is similar to memory management

Implemented by "storage manager" or "file system" part of OS

Most OSs support more than one file system.

Figures of merit

Figures of merit

- Fast **sequential** access

Figures of merit

- Fast **sequential** access
- Fast **random** access

Figures of merit

- Fast **sequential** access
- Fast **random** access
- Ability to **grow** the file

Figures of merit

- Fast **sequential** access
- Fast **random** access
- Ability to **grow** the file
- **Easy allocation** of storage

Figures of merit

- Fast **sequential** access
- Fast **random** access
- Ability to **grow** the file
- **Easy allocation** of storage
- Efficiency of **space utilization** on the disk

Figures of merit

- Fast **sequential** access
- Fast **random** access
- Ability to **grow** the file
- **Easy allocation** of storage
- Efficiency of **space utilization** on the disk

→ Use case?

Figures of merit

- Fast **sequential** access → Use case?
- Fast **random** access → Use case?
- Ability to **grow** the file
- **Easy allocation** of storage
- Efficiency of **space utilization** on the disk

Figures of merit

- Fast **sequential** access → Use case?
- Fast **random** access → Use case?
- Ability to **grow** the file → Flexibility
- **Easy allocation** of storage
- Efficiency of **space utilization** on the disk

Figures of merit

- Fast **sequential** access → Use case?
- Fast **random** access → Use case?
- Ability to **grow** the file → Flexibility
- **Easy allocation** of storage → Time
- Efficiency of **space utilization** on the disk

Figures of merit

- Fast **sequential** access → Use case?
- Fast **random** access → Use case?
- Ability to **grow** the file → Flexibility
- **Easy allocation** of storage → Time
- Efficiency of **space utilization** on the disk
  - → Concerns of internal and external fragmentation

Figures of merit

- Fast **sequential** access → Use case?
- Fast **random** access → Use case?
- Ability to **grow** the file → Flexibility
- **Easy allocation** of storage → Time
- Efficiency of **space utilization** on the disk → Concerns of internal and external fragmentation

Common concerns to VM and file system

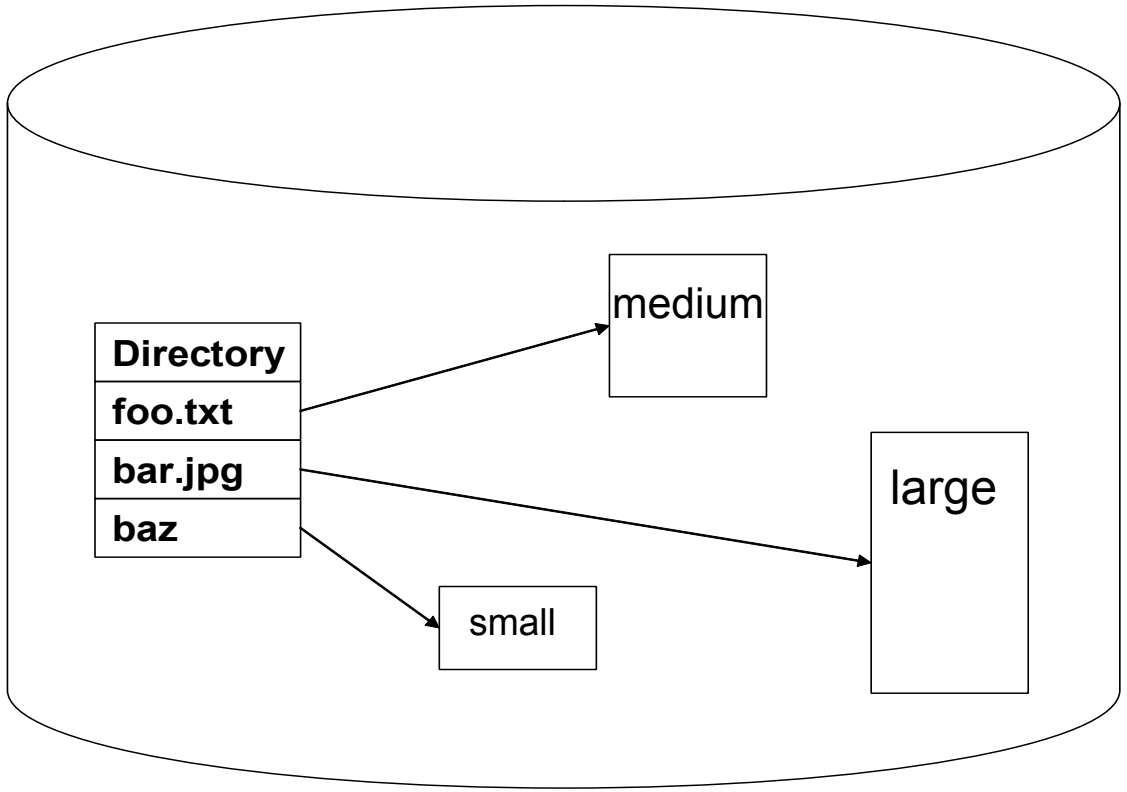
Figures of merit

Specific to the file system

- Fast **sequential** access → Use case?
- Fast **random** access → Use case?
- Ability to **grow** the file → Flexibility
- **Easy allocation** of storage → Time
- Efficiency of **space utilization** on the disk → Concerns of internal and external fragmentation

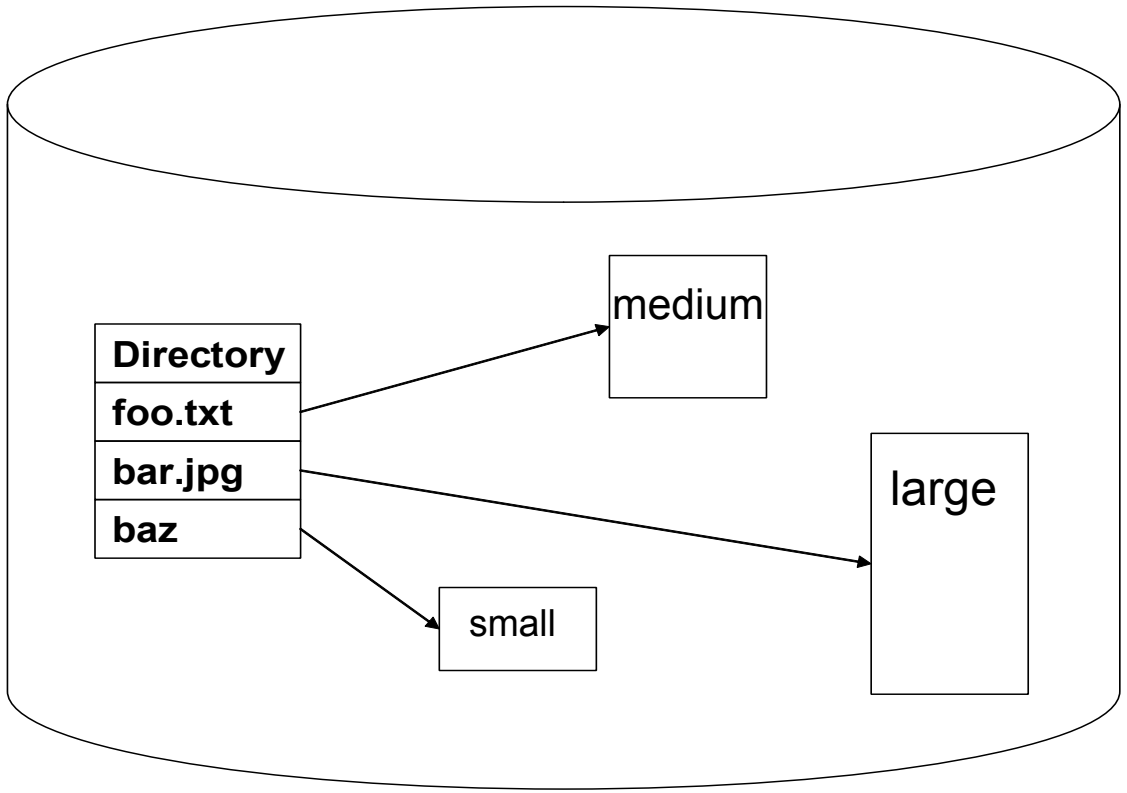
Common concerns to VM and file system

Contiguous allocation

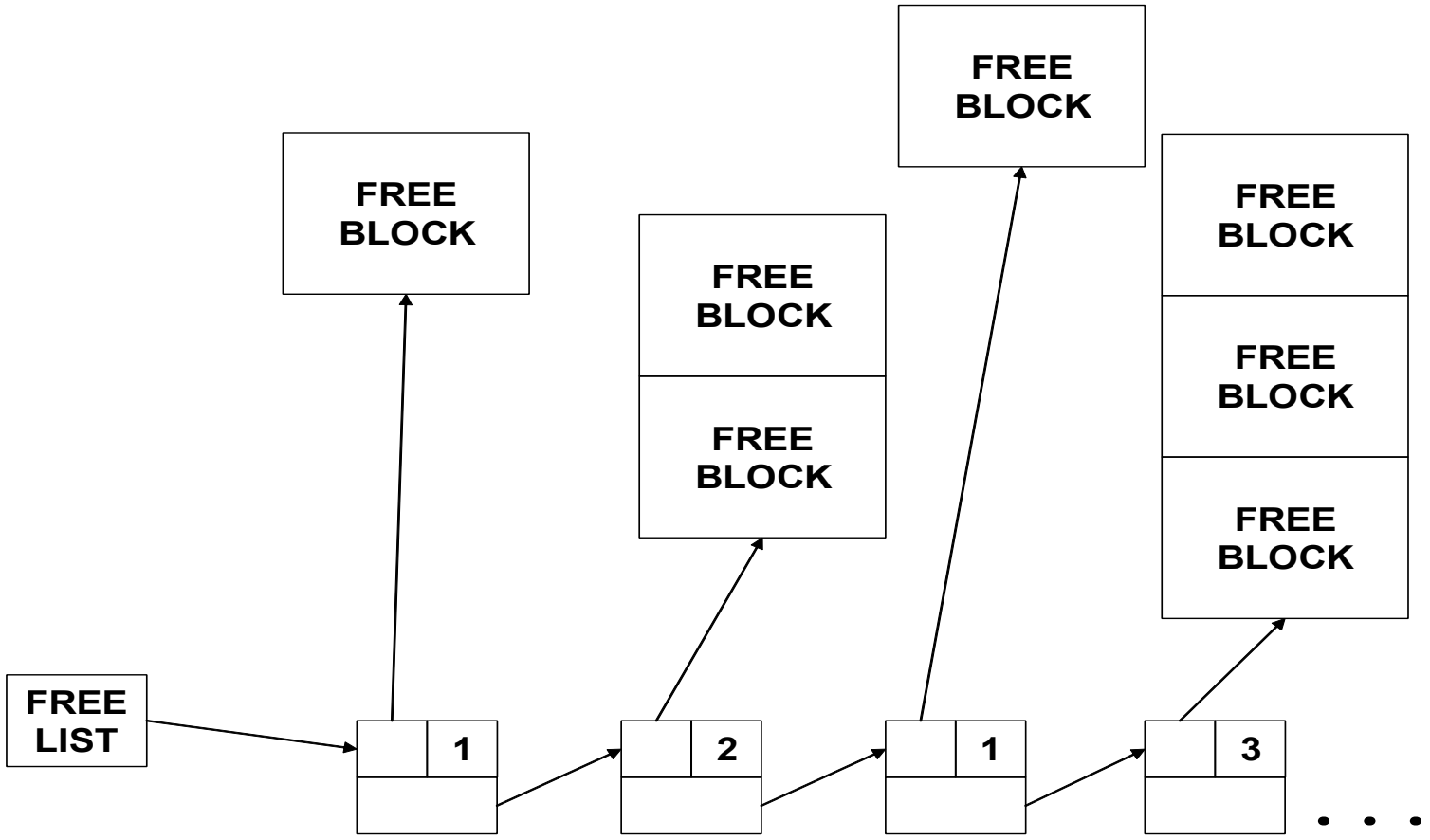


Contiguous allocation

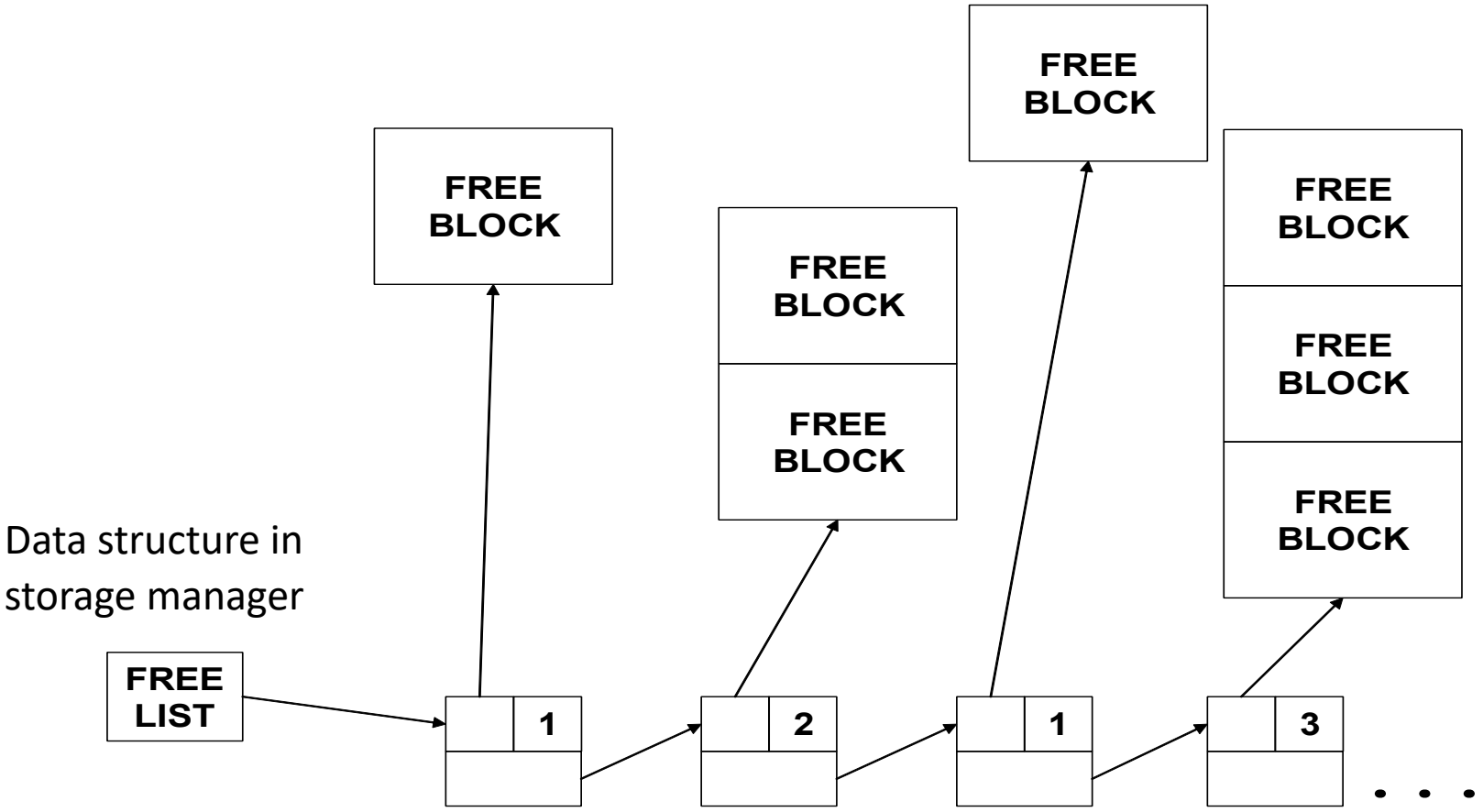
Similar to variable-sized partitioning in memory management



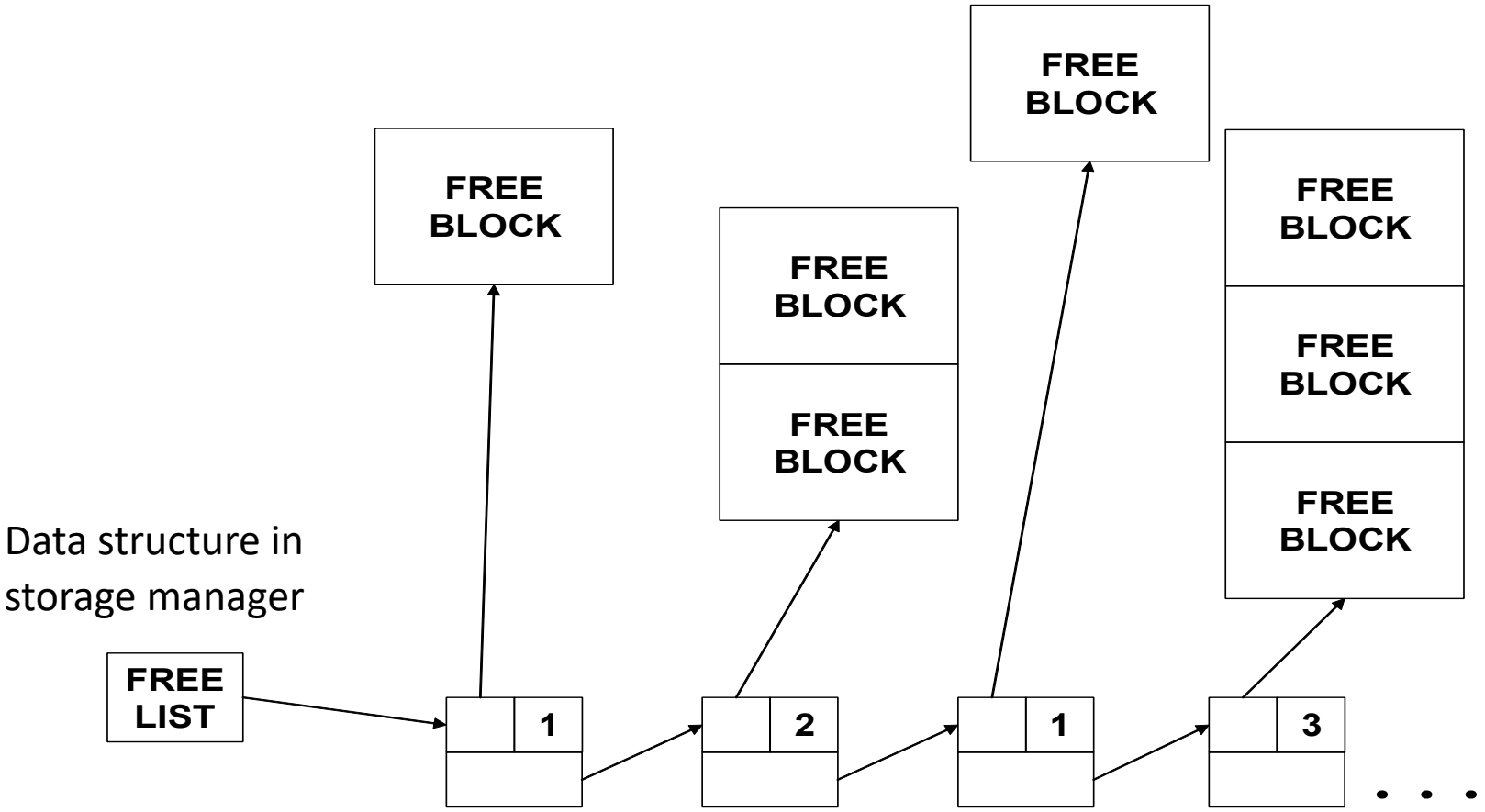
Free list for contiguous allocation



Free list for contiguous allocation

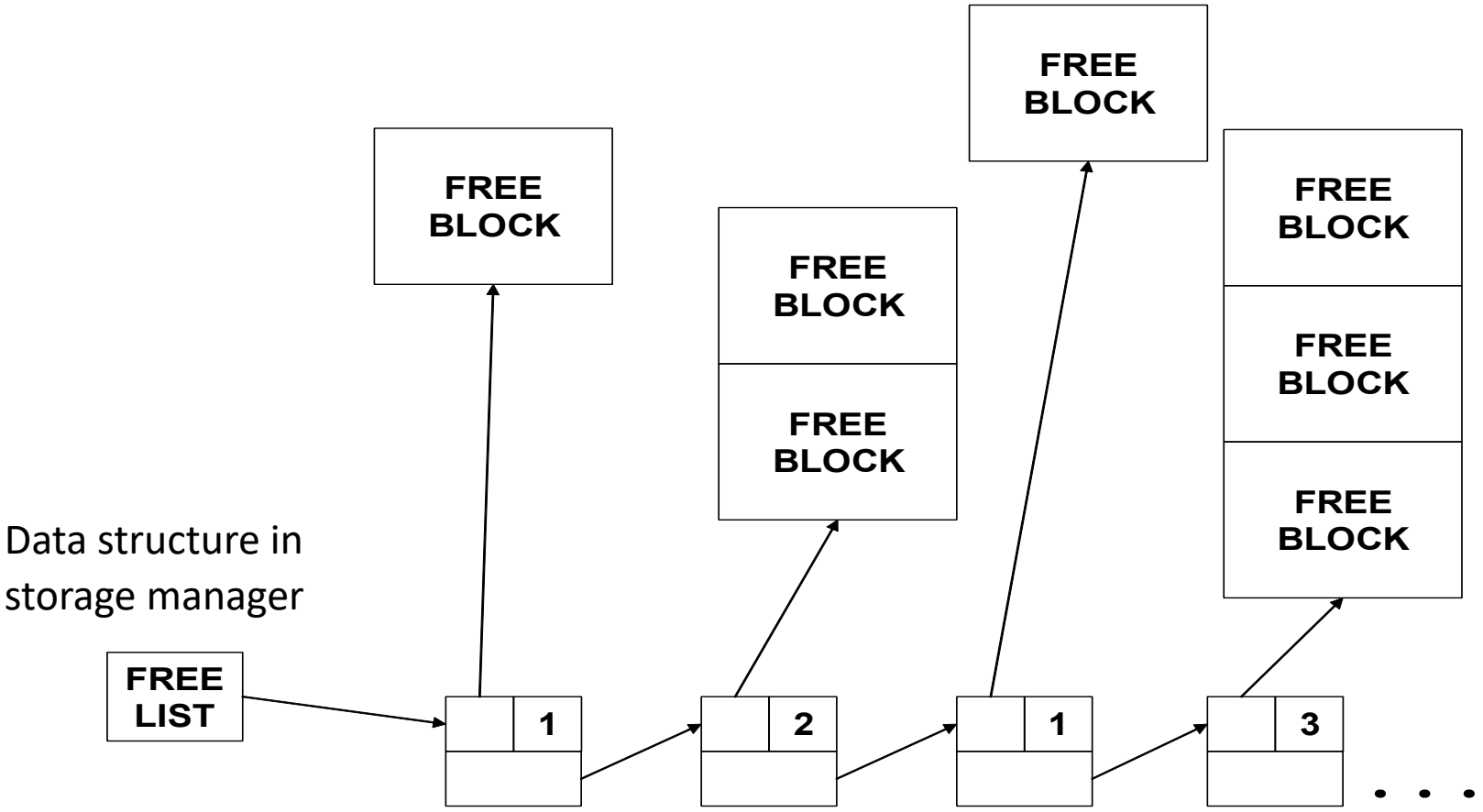


Free list for contiguous allocation



Where does this data structure live?

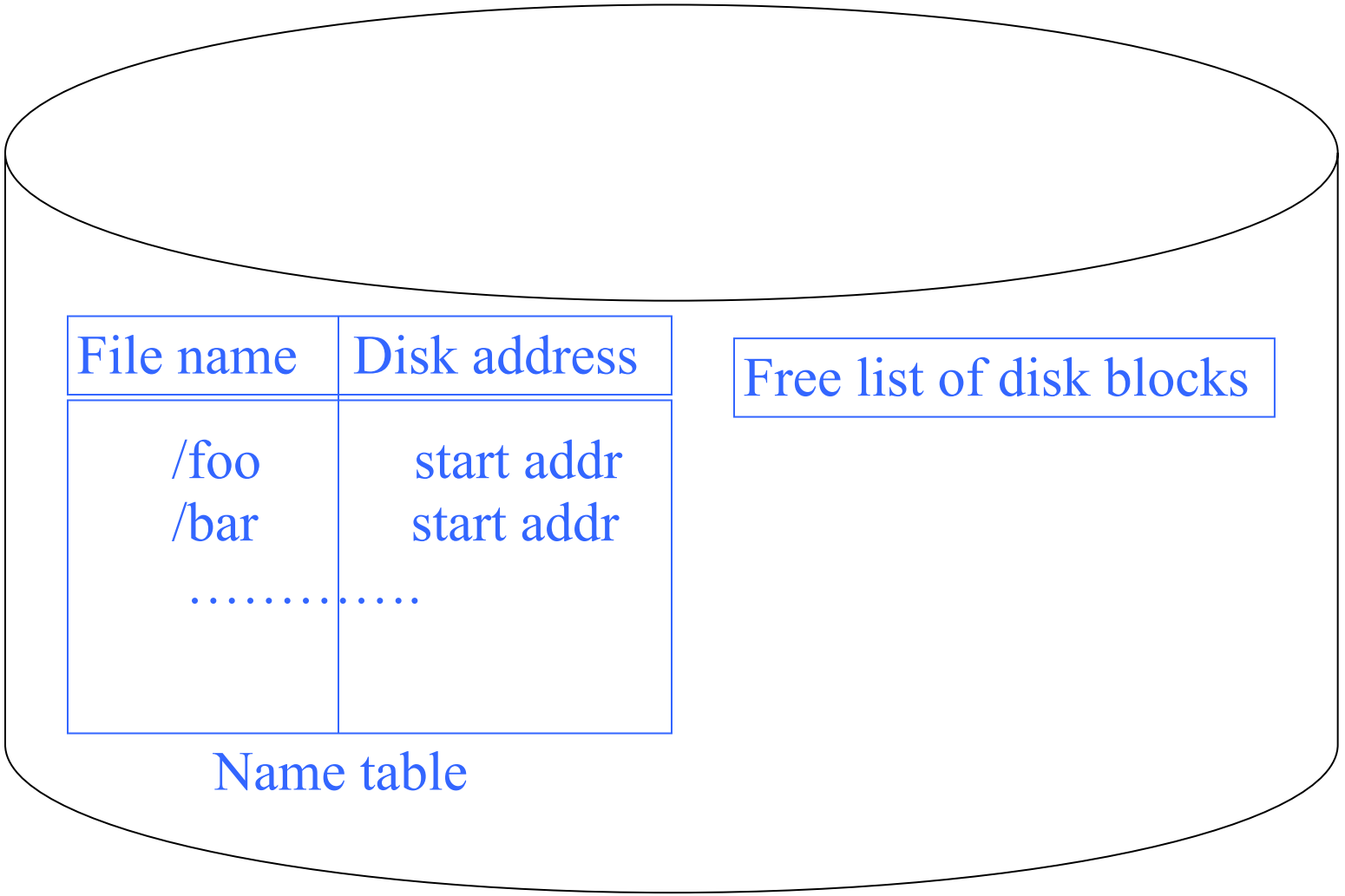
Free list for contiguous allocation



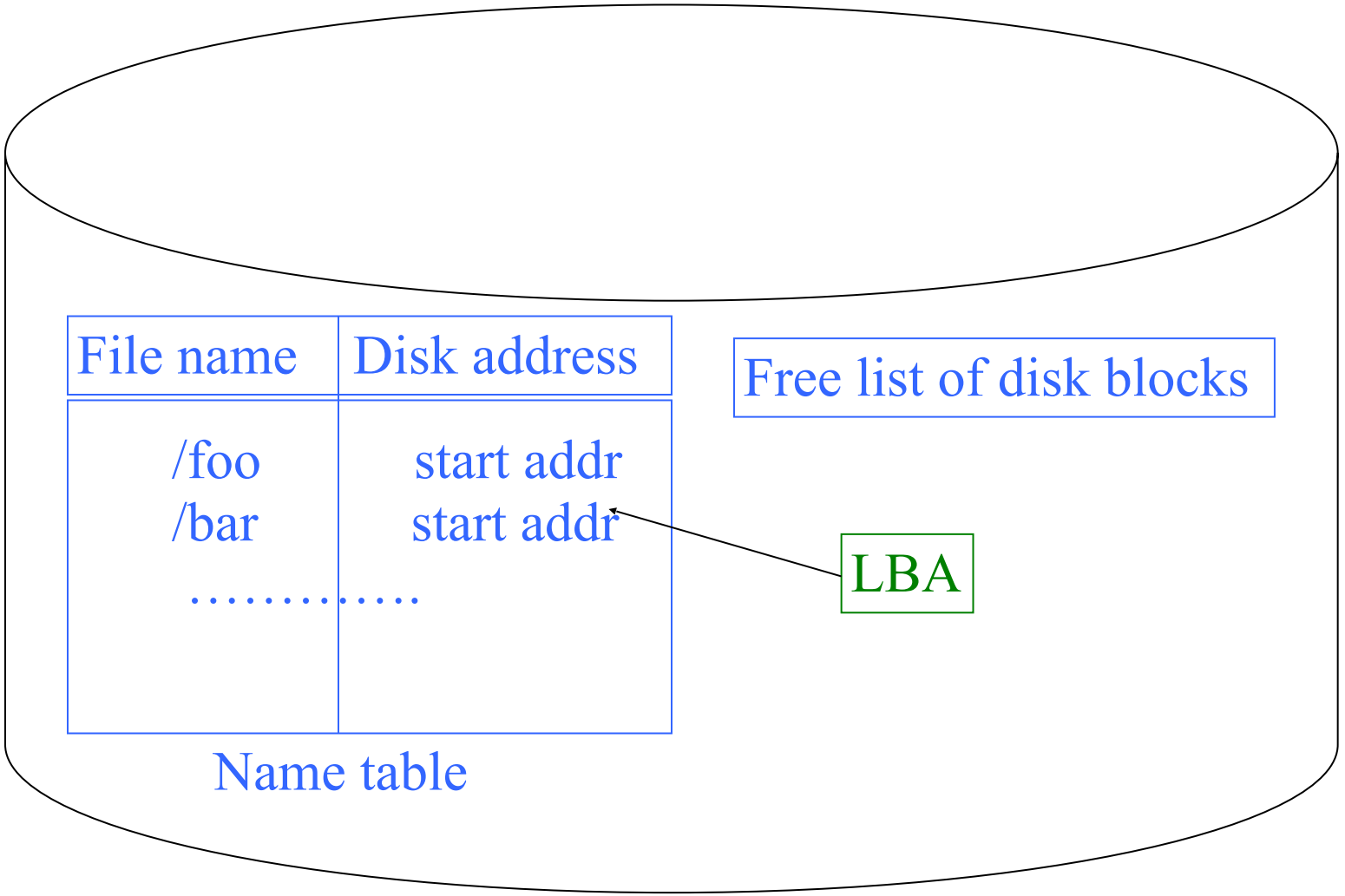
Where does this data structure live?

➔On disk but "cached" in physical memory

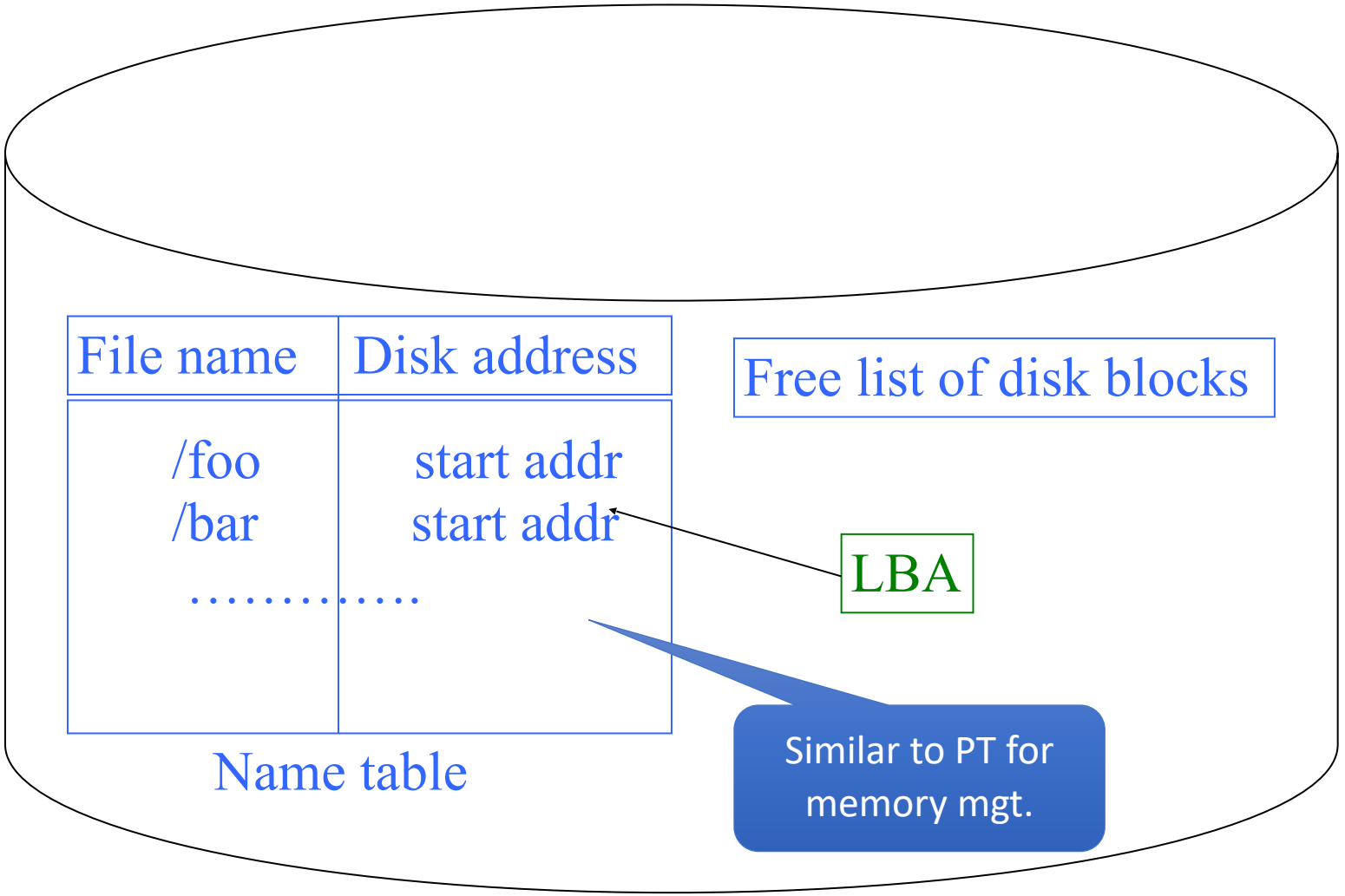
Data structures for storage manager



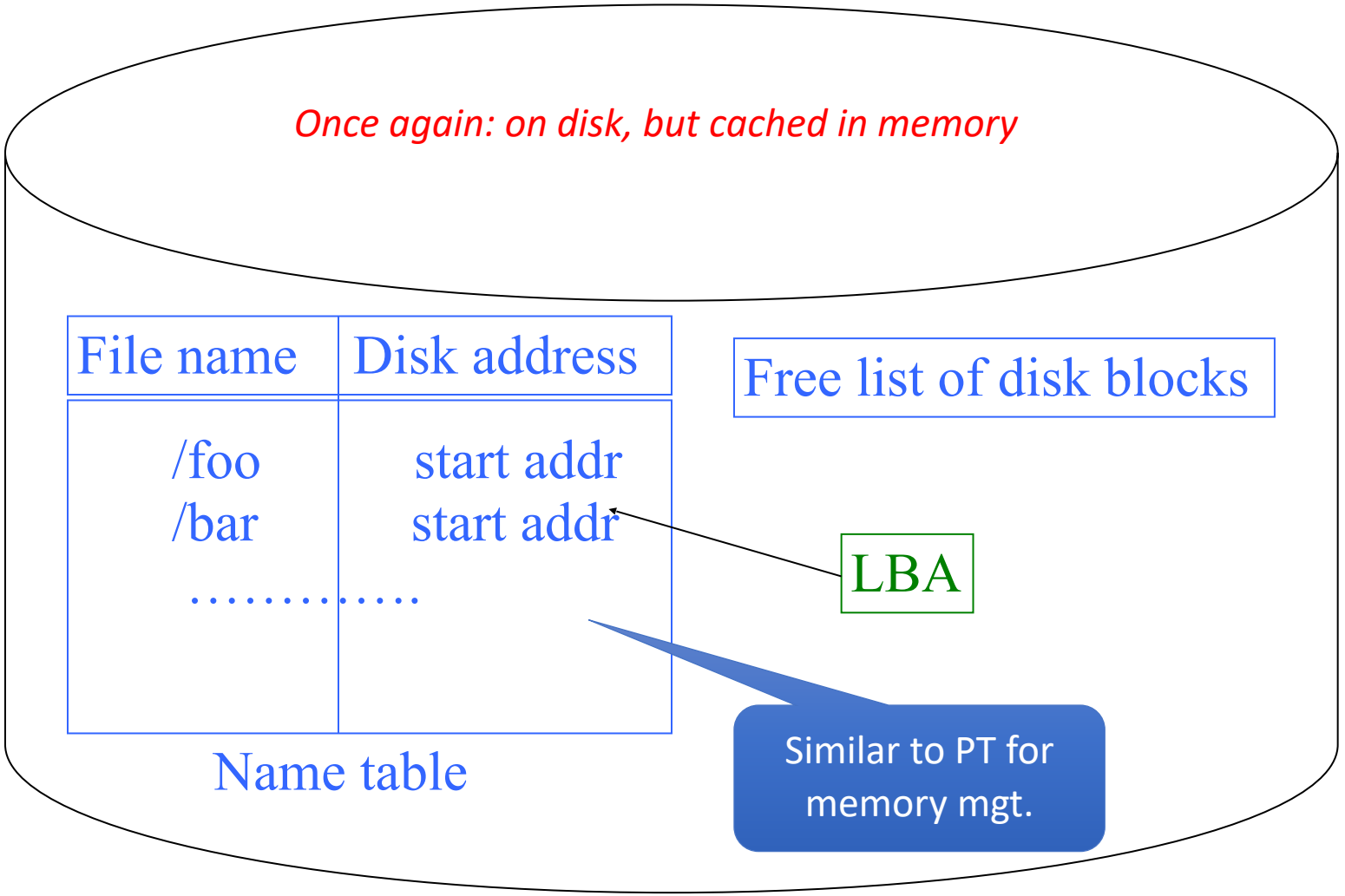
Data structures for storage manager



Data structures for storage manager



Data structures for storage manager



Contiguous allocation

- Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Contiguous allocation

X

- Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Contiguous allocation

X

X

- Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Similar issues to variable-sized partitioned memory management

- First fit
- Best fit
- Compaction

Contiguous allocation

X
- Growth?

X
- Quickness of allocation (time)?

X
- Fragmentation (space)?

Both internal and external

- Sequential access?
- Random access?

Similar issues to variable-sized partitioned memory management
- First fit
- Best fit
- Compaction

Contiguous allocation

X
- Growth?

X
- Quickness of allocation (time)?

X
- Fragmentation (space)?

Both internal and external

✓
- Sequential access?
- Random access?

Similar issues to variable-sized partitioned memory management
- First fit
- Best fit
- Compaction

Contiguous allocation

- X Growth?
- X Quickness of allocation (time)?
- X Fragmentation (space)? Both internal and external
- ✓ Sequential access?
- ✓ Random access?

Similar issues to variable-sized partitioned memory management

- First fit
- Best fit
- Compaction

Contiguous allocation

X
- Growth?

Can be fixed with "overflow" areas…

X
- Quickness of allocation (time)?

X
- Fragmentation (space)?

Both internal and external

✓
- Sequential access?

✓
- Random access?

Similar issues to variable-sized partitioned memory management

- First fit
- Best fit
- Compaction

Contiguous allocation

- X Growth?
- X Quickness of allocation (time)?
- X Fragmentation (space)?
- ✓ Sequential access?
- ✓ Random access?

Can be fixed with "overflow" areas…

➔ Contiguous allocation with overflow

Both internal and external

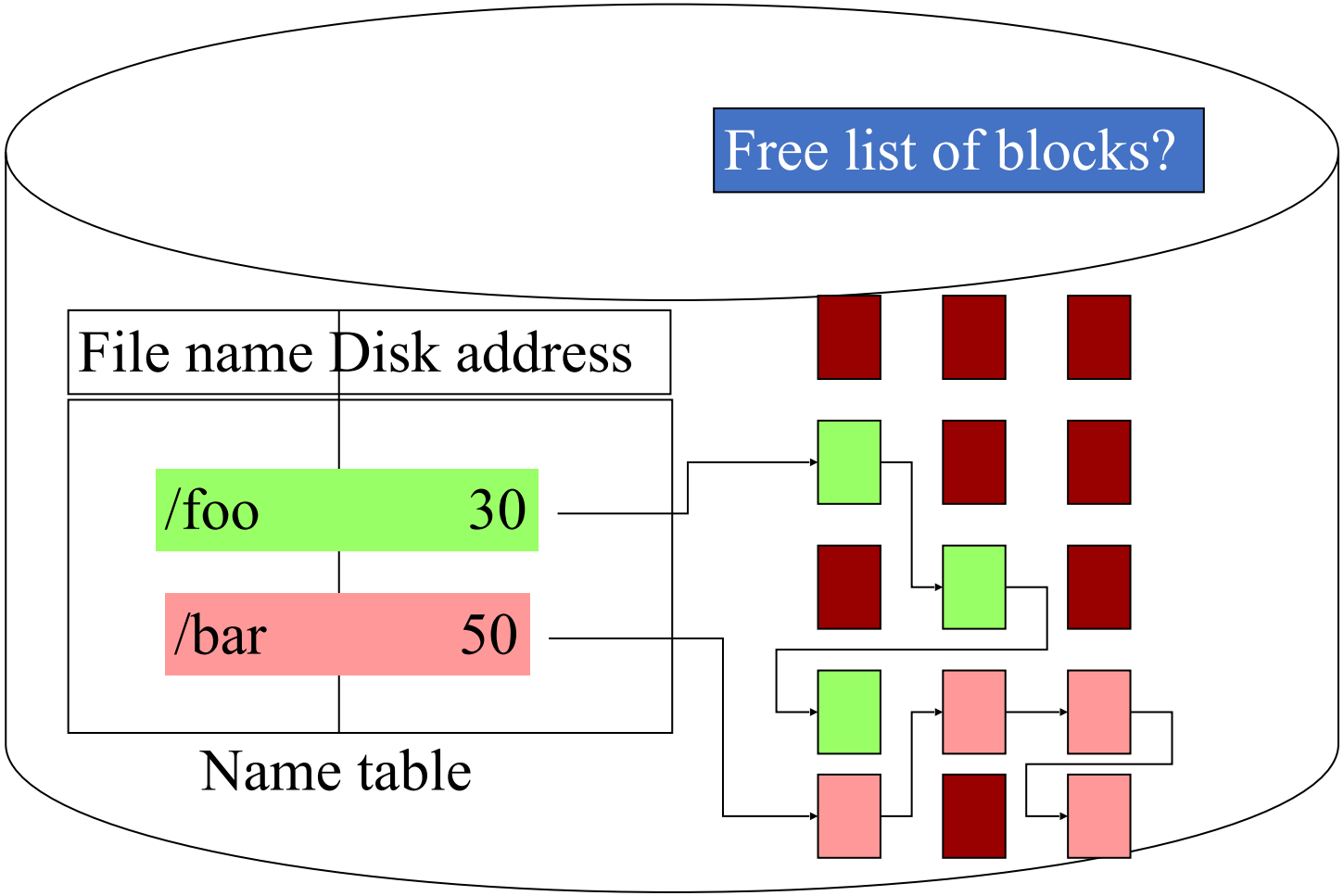
Similar issues to variable-sized partitioned memory management

- First fit
- Best fit
- Compaction

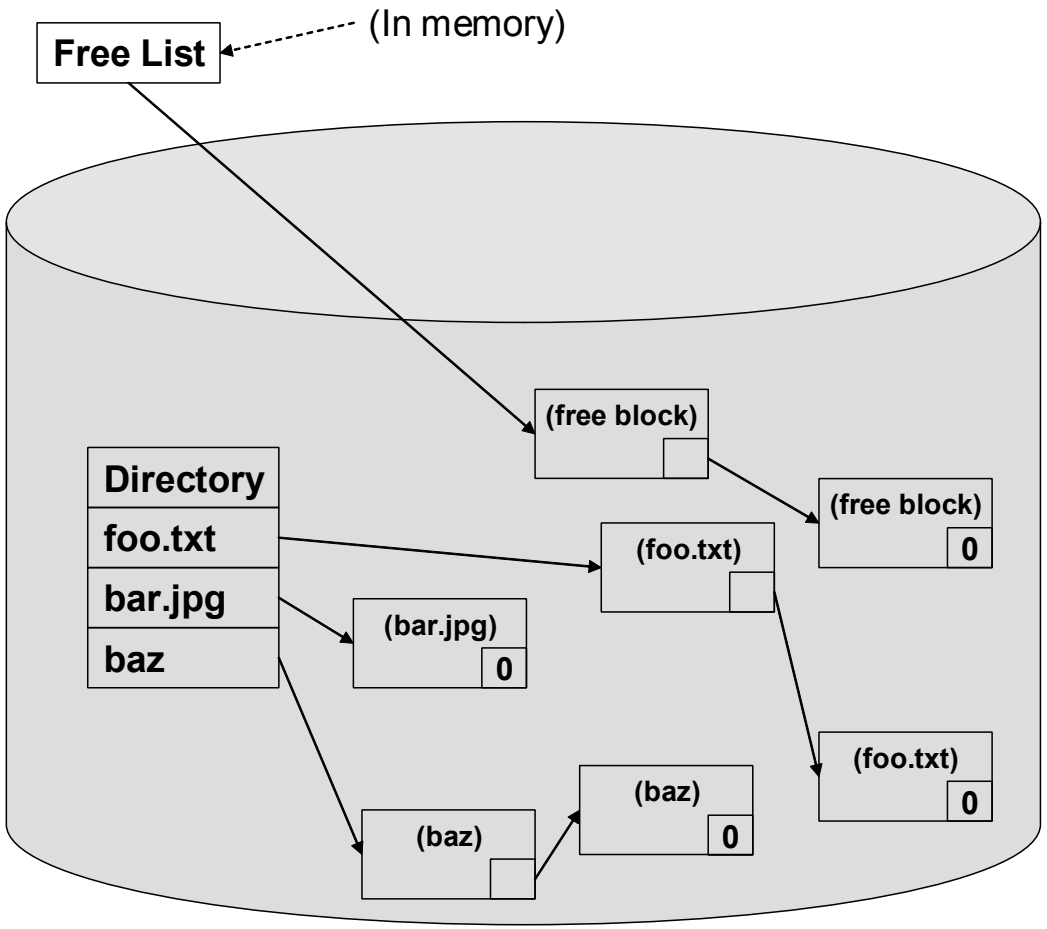
File allocation schemes on the disk

- ✓ Contiguous allocation
- ✓ Contiguous allocation with overflow
- ➔ Linked allocation
  - FAT
  - Indexed allocation
  - Multilevel indexed
  - Hybrid indexed

Linked List Allocation

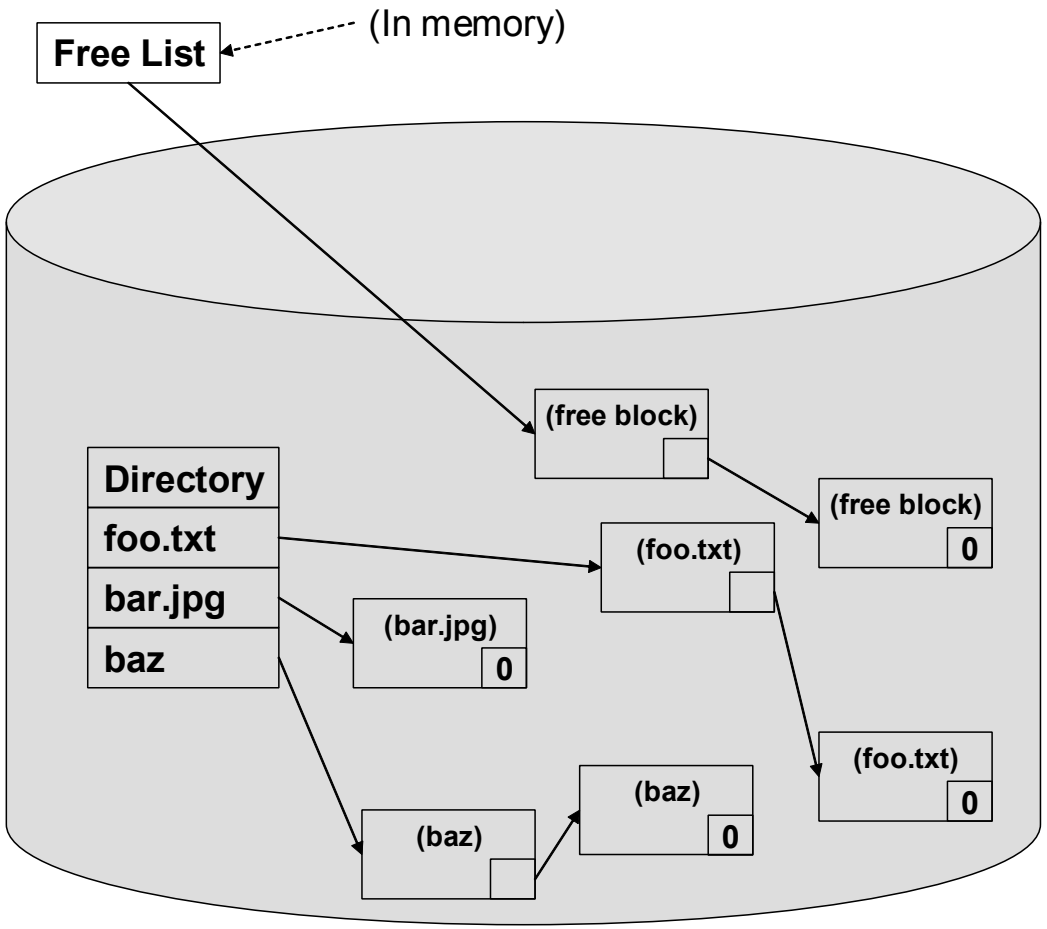


Linked allocation



Linked allocation

As before, data structures are on disk but cached in memory



Linked list allocation

- Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Linked list allocation

- ✓ Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Linked list allocation

- ✓ Growth?
- ✓ Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

One block at a time

Linked list allocation

- ✓ Growth?
- ✓ Quickness of allocation (time)? One block at a time
- ✓ Fragmentation (space)? No external
- Sequential access?
- Random access?

Linked list allocation

- ✓ Growth?
- ✓ Quickness of allocation (time)? One block at a time
- ✓ Fragmentation (space)? No external
- ✓ Sequential access?
- Random access?

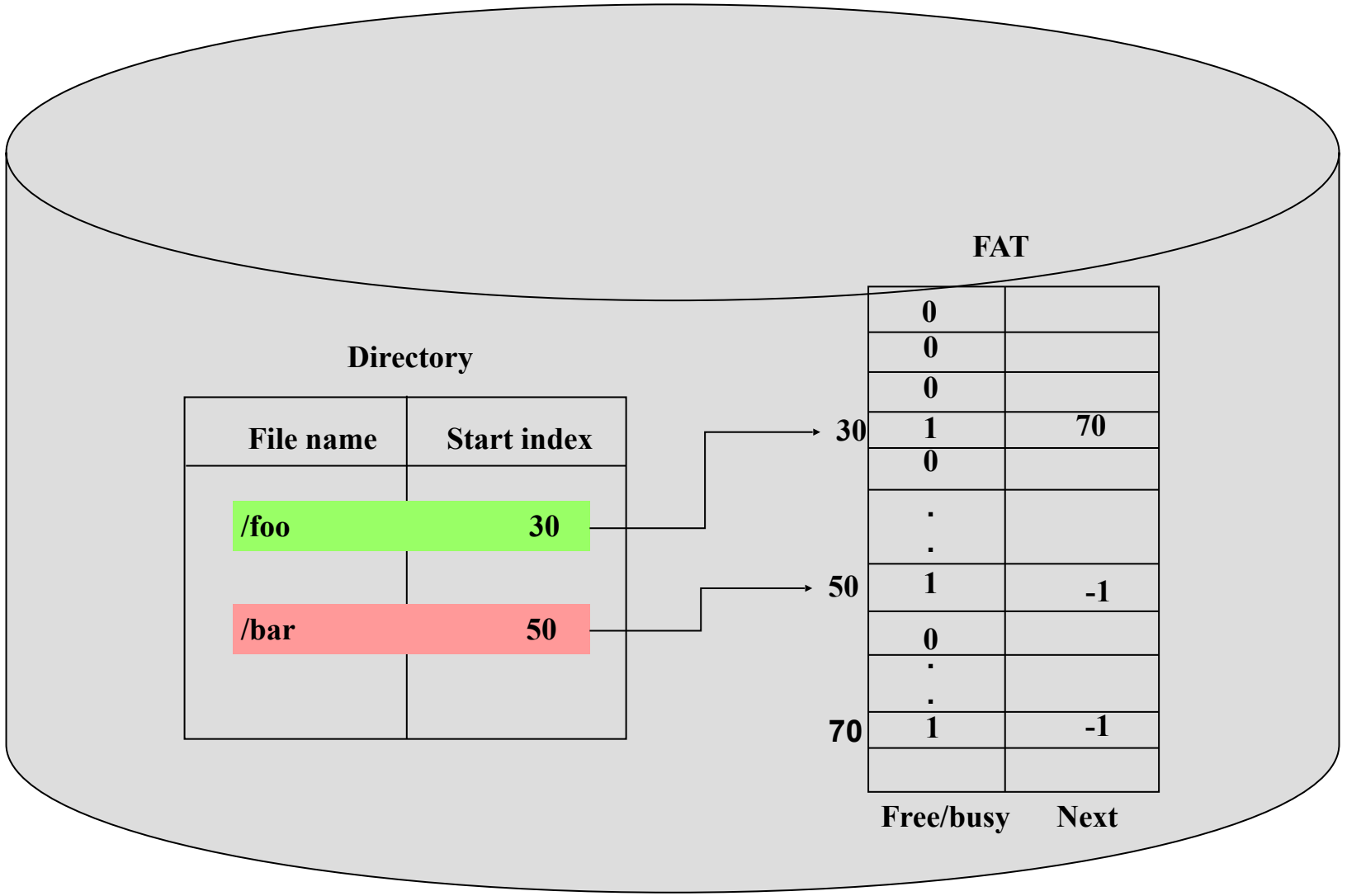
Linked list allocation

- ✓ Growth?
- ✓ Quickness of allocation (time)? One block at a time
- ✓ Fragmentation (space)? No external
- ✓ Sequential access?
- X Random access? Very dependent on seek time

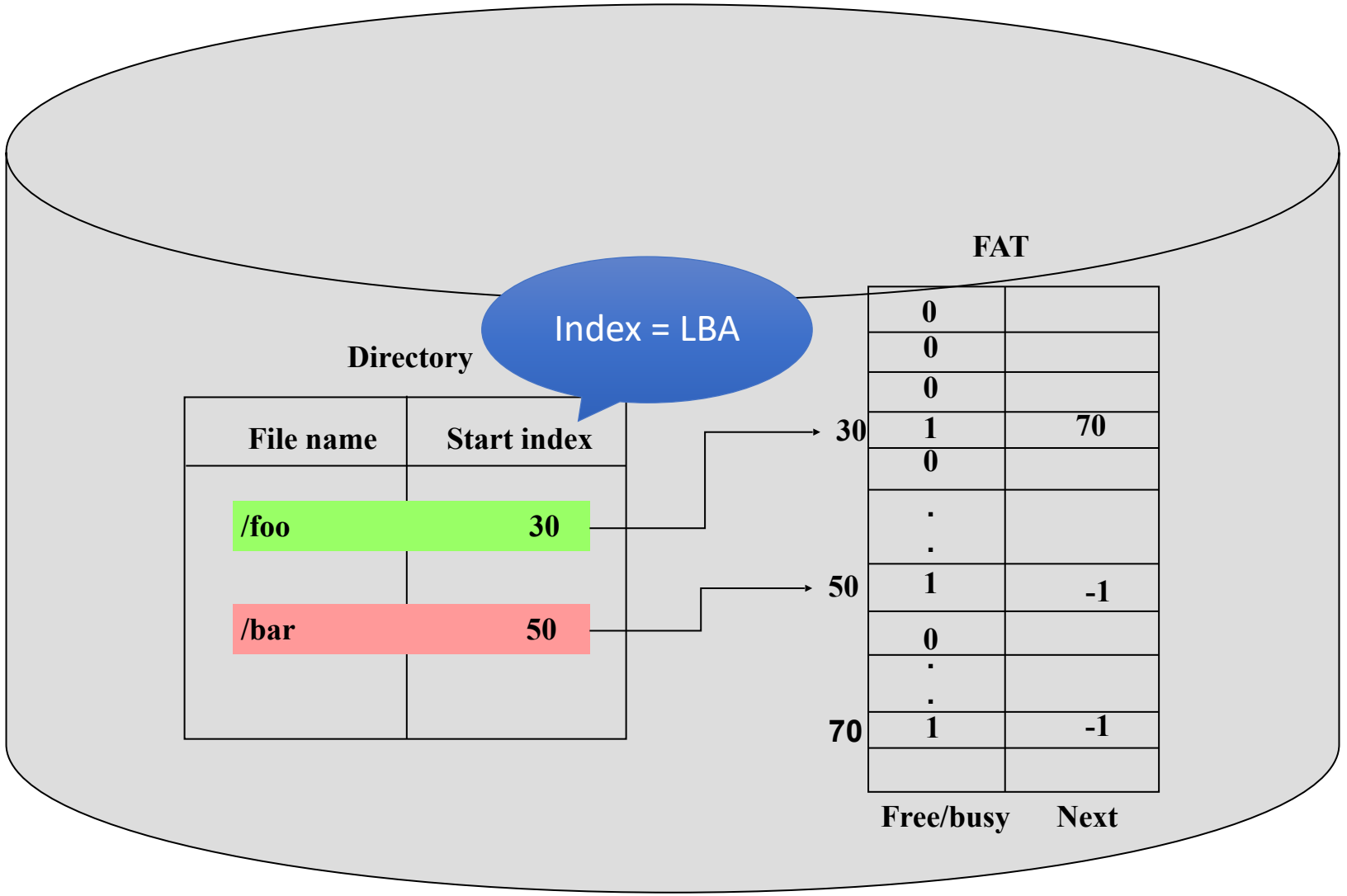
File allocation schemes on the disk

- ✓ Contiguous allocation
- ✓ Contiguous allocation with overflow
- ✓ Linked allocation
- → FAT
- Indexed allocation
- Multilevel indexed
- Hybrid indexed

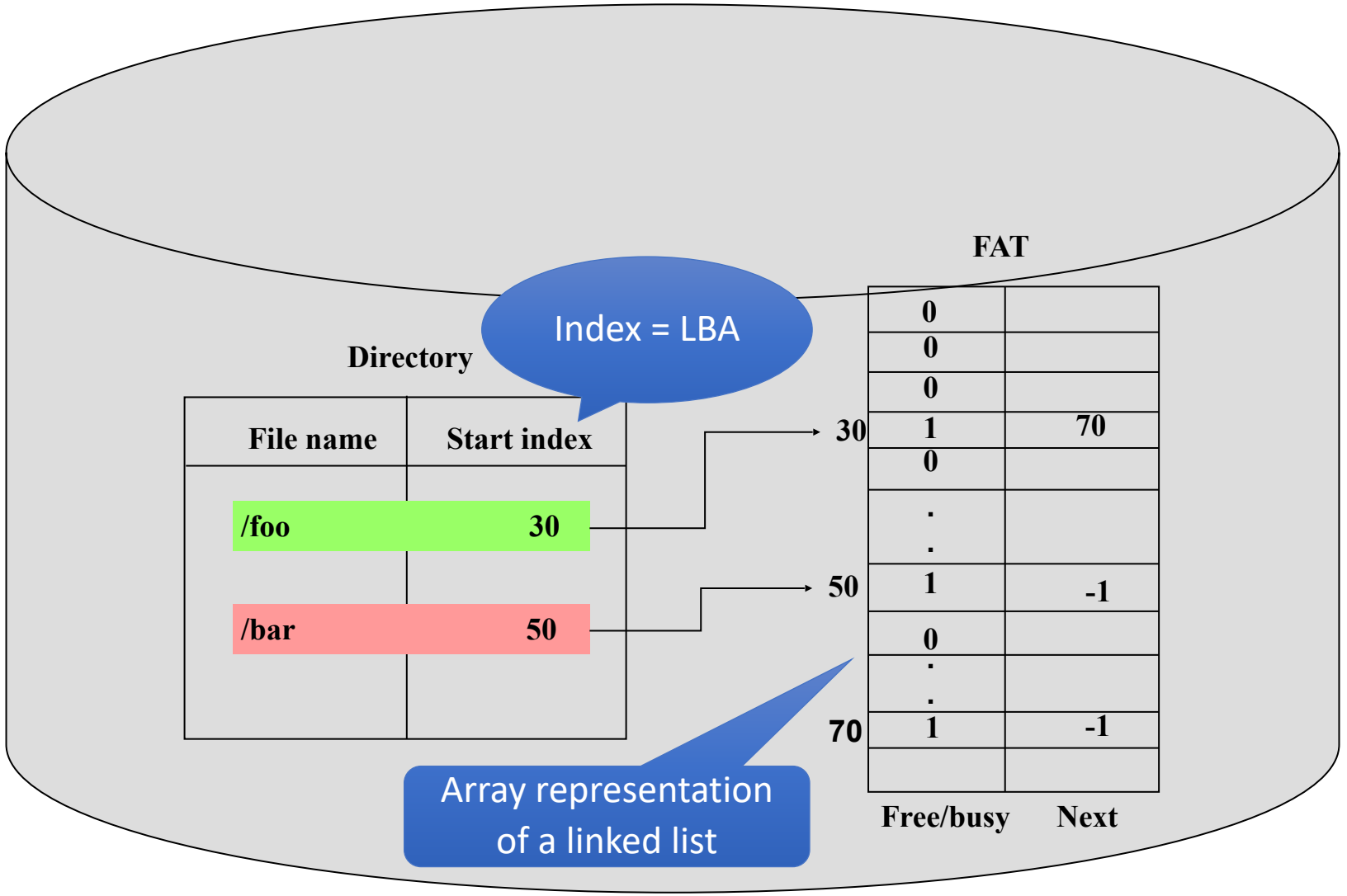
File allocation table



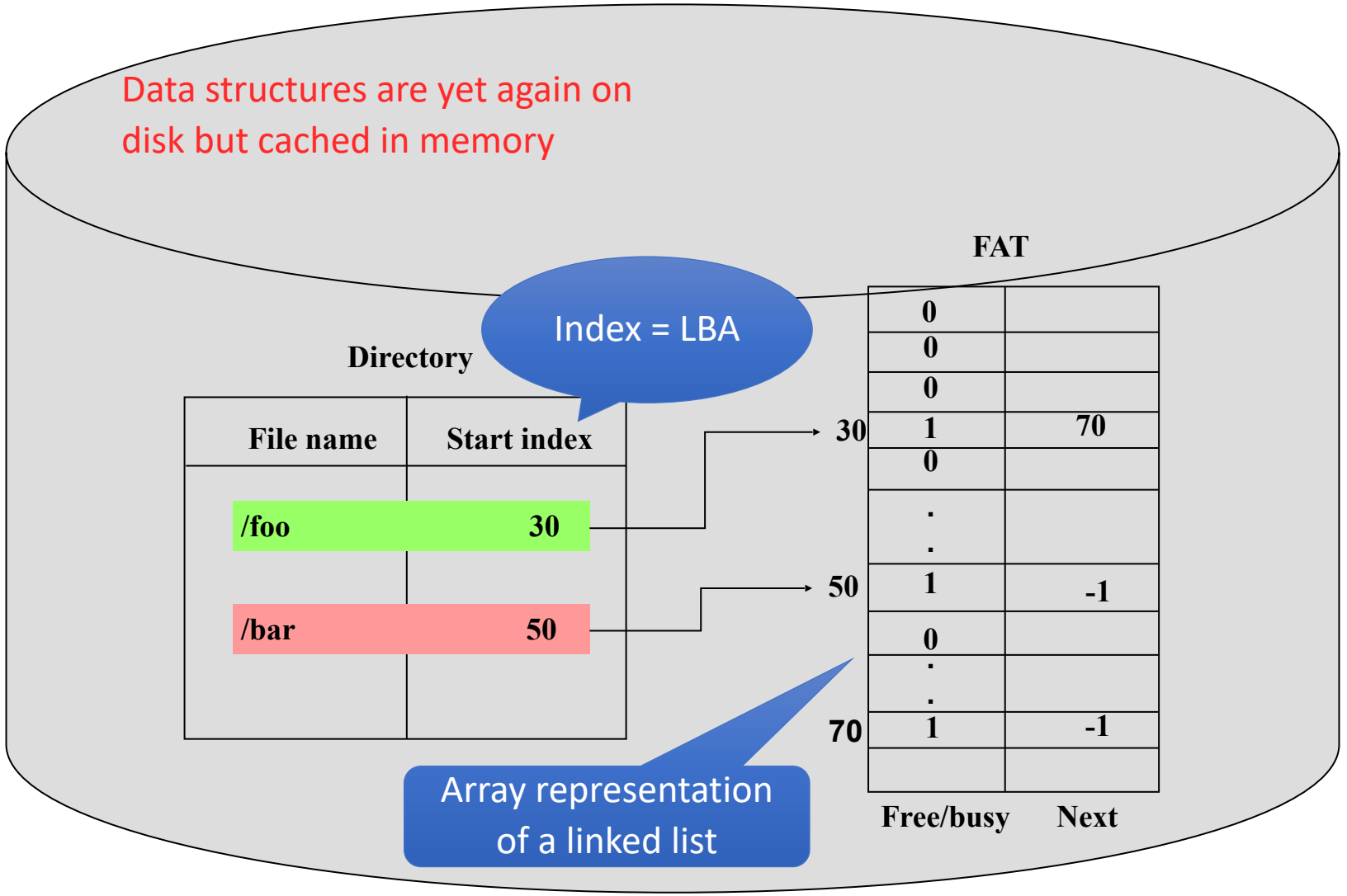
File allocation table



File allocation table



File allocation table



File allocation table

- Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

File allocation table

- ✓ Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

File allocation table

- ✓ Growth?
- ✓ Quickness of allocation (time)? Better than linked; FAT cached in memory
- Fragmentation (space)?
- Sequential access?
- Random access?

File allocation table

- ✓ Growth?
- ✓ Quickness of allocation (time)? Better than linked; FAT cached in memory
- ✓ Fragmentation (space)? Same as linked
- Sequential access?
- Random access?

File allocation table

- ✓ Growth?
- ✓ Quickness of allocation (time)? Better than linked; FAT cached in memory
- ✓ Fragmentation (space)? Same as linked
- ✓ Sequential access? Similar to linked
- Random access?

File allocation table

- ✓ Growth?
- ✓ Quickness of allocation (time)? Better than linked; FAT cached in memory
- ✓ Fragmentation (space)? Same as linked
- ✓ Sequential access? Similar to linked
- ✓ Random access? Still depends on seek time, but next-block pointers stored in FAT, so less serialization and block transfers from disk

File allocation table

- ✓ Growth?
- ✓ Quickness of allocation (time)? Better than linked; FAT cached in memory
- ✓ Fragmentation (space)? Same as linked
- ✓ Sequential access? Similar to linked
- ✓ Random access? Still depends on seek time, but next-block pointers stored in FAT, so less serialization and block transfers from disk

Less error prone than linked allocation thanks to data/metadata separation

File allocation table

- ✓ Growth?
- ✓ Quickness of allocation (time)? Better than linked; FAT cached in memory
- ✓ Fragmentation (space)? Same as linked
- ✓ Sequential access? Similar to linked
- ✓ Random access? Still depends on seek time, but next-block pointers stored in FAT, so less serialization and block transfers from disk

Less error prone than linked allocation thanks to data/metadata separation

FAT table size limits max supported disk capacity. Must resort to disk partitioning , which becomes a burden to user

File allocation table

- ✓ Growth?
- ✓ Quickness of allocation (time)? Better than linked; FAT cached in memory
- ✓ Fragmentation (space)? Same as linked
- ✓ Sequential access? Similar to linked
- ✓ Random access? Still depends on seek time, but next-block pointers stored in FAT, so less serialization and block transfers from disk

Less error prone than linked allocation thanks to data/metadata separation

FAT table size limits max supported disk capacity. Must resort to disk partitioning , which becomes a burden to user

- E.g., FAT-16 limits pointer size to 16 bits; can only address 2^16 blocks

File allocation schemes on the disk

- ✓ Contiguous allocation
- ✓ Contiguous allocation with overflow
- ✓ Linked allocation
- ✓ FAT
- → Indexed allocation
- Multilevel indexed
- Hybrid indexed

Some Unix terminology

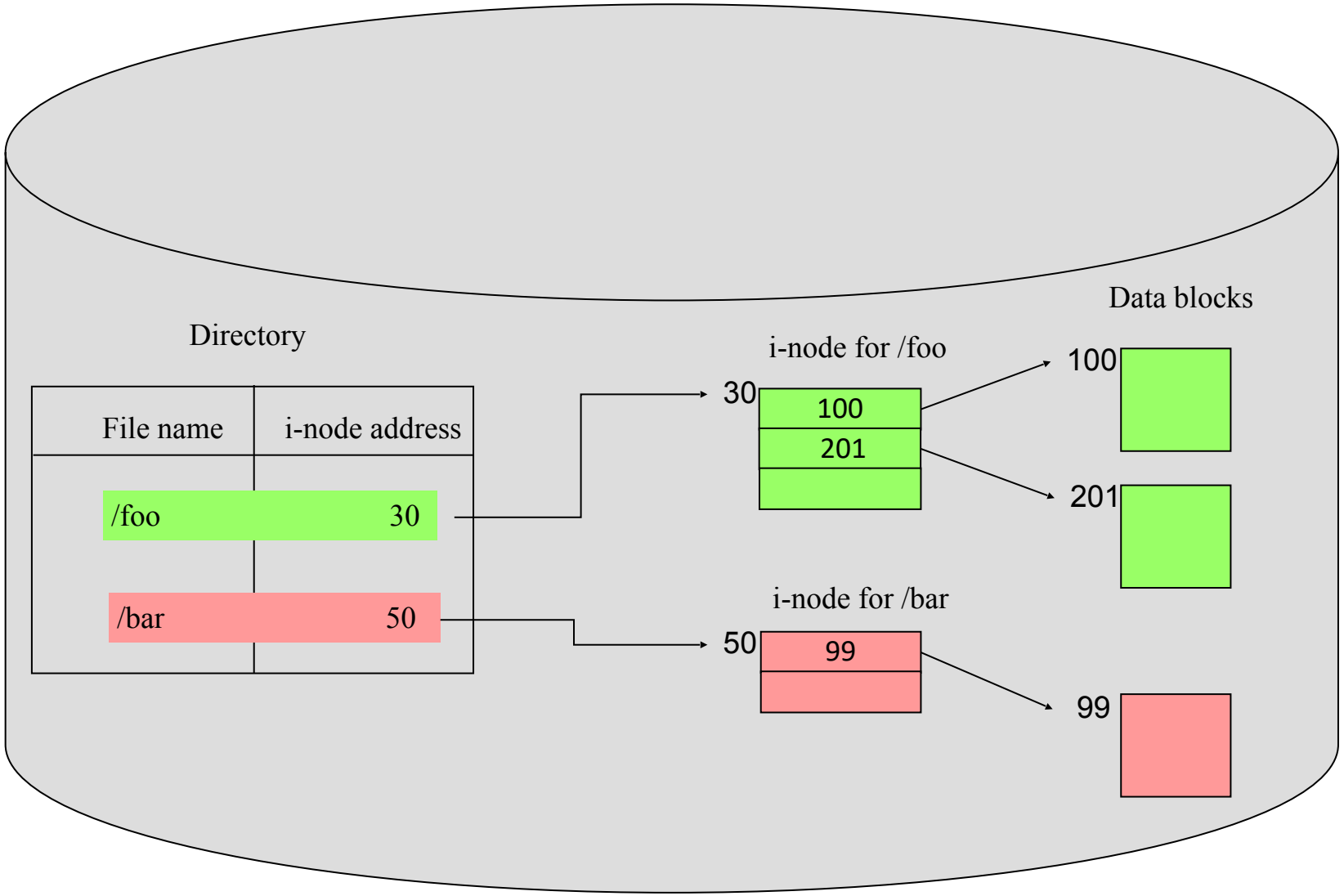
- The Unix file system stored the metadata for a file separately from the directory entry; this was a novel idea at the time
- This technique has been used in a number of subsequent file systems but the Unix terminology is often used to describe it

Some Unix terminology

- The Unix file system stored the metadata for a file separately from the directory entry; this was a novel idea at the time
- This technique has been used in a number of subsequent file systems but the Unix terminology is often used to describe it
- i-node (modern usage *inode*)
  - All of the information about a file except name was stored in a fixed-length entry on disk called an i-node (similar to what a PCB is for a process)
  - The i-nodes were kept in an array on the file system so that the disk location of an i-node could be calculated from its index, often called the i-number
  - i-nodes can be cached by the OS in memory when a file is in use
- Directory
  - Directories contained only file names and the i-number (i-node index number)
  - It is possible for two directory entries to contain the same i-number (more on that later)

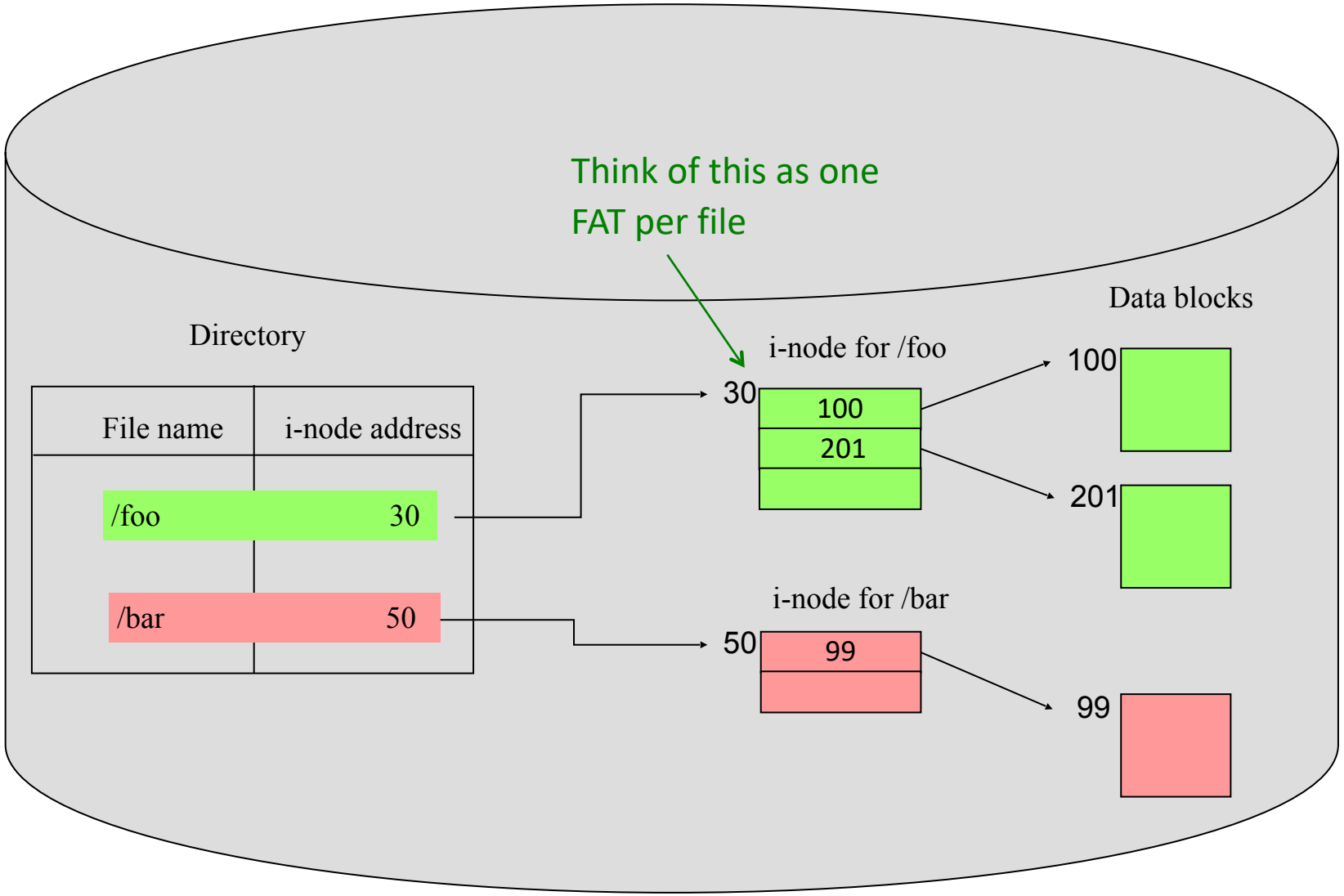
Upcoming:
dag2200

Indexed allocation



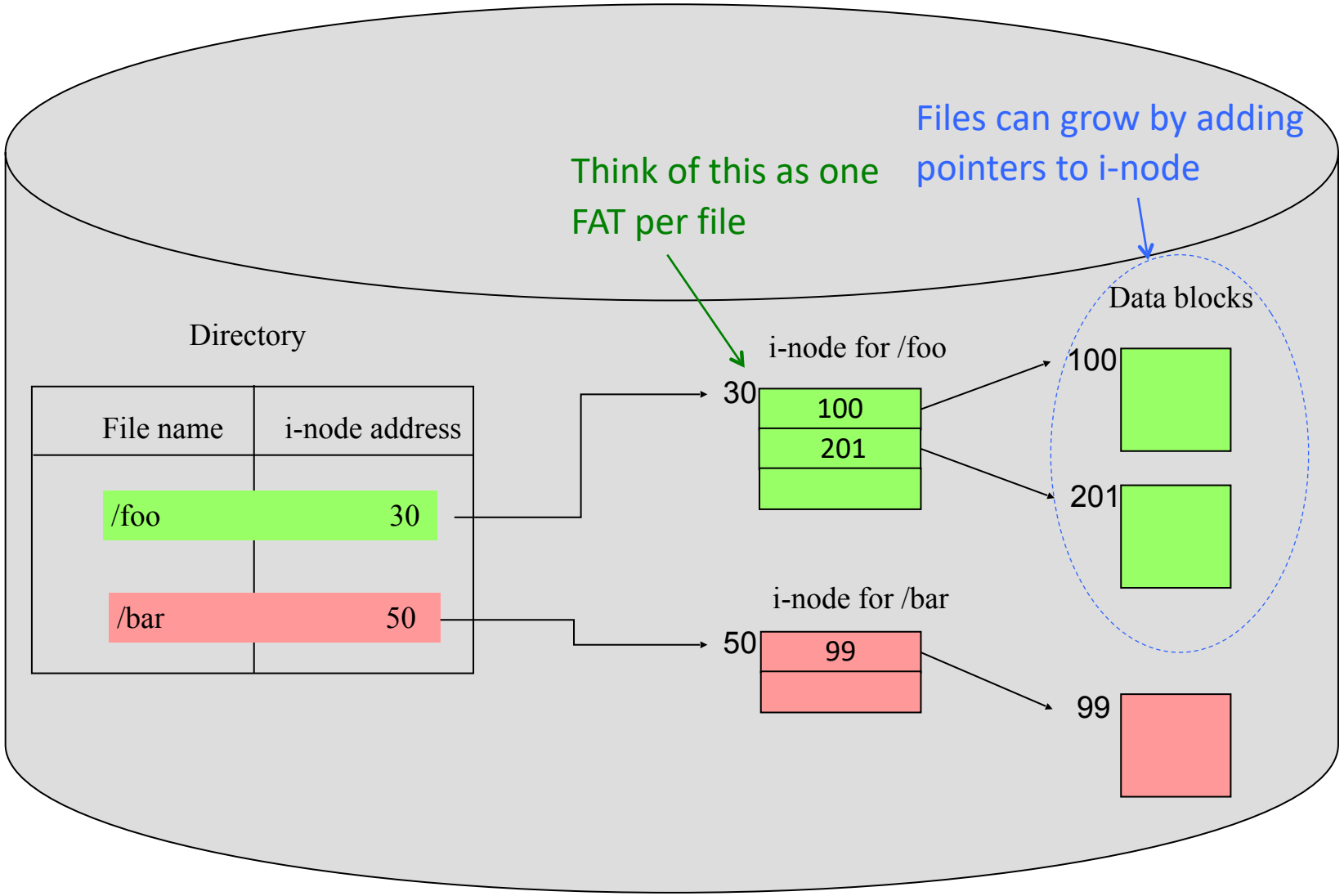
Upcoming:
dag2200

Indexed allocation



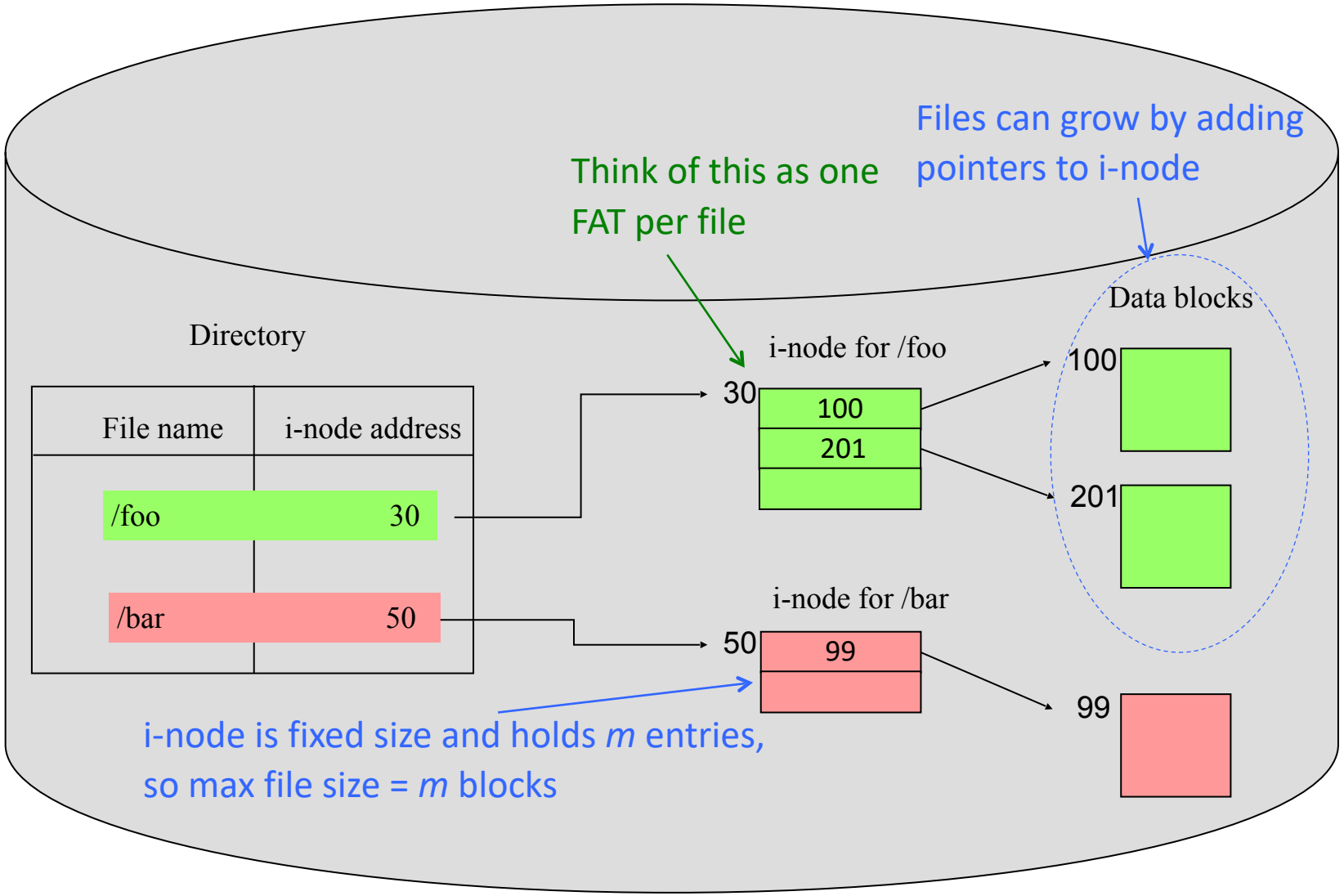
Upcoming:
dag2200

Indexed allocation



Upcoming:
dag2200

Indexed allocation



Indexed allocation…

A. Suffers from external fragmentation
B. Offers sequential access as fast as FAT
C. Supports arbitrarily large files

Indexed allocation

- Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Indexed allocation

X

- Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Big problems

Indexed allocation

X
- Growth?

✓
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Big problems

Indexed allocation

X
- Growth?

✓
- Quickness of allocation (time)?

✓
- Fragmentation (space)?
- Sequential access?
- Random access?

Big problems

Indexed allocation

X
- Growth?

Big problems

✓
- Quickness of allocation (time)?

✓
- Fragmentation (space)?

✓
- Sequential access?

Same as FAT

- Random access?

Indexed allocation

- X Growth?
- ✓ Quickness of allocation (time)?
- ✓ Fragmentation (space)?
- ✓ Sequential access?
- ✓ Random access?

Big problems

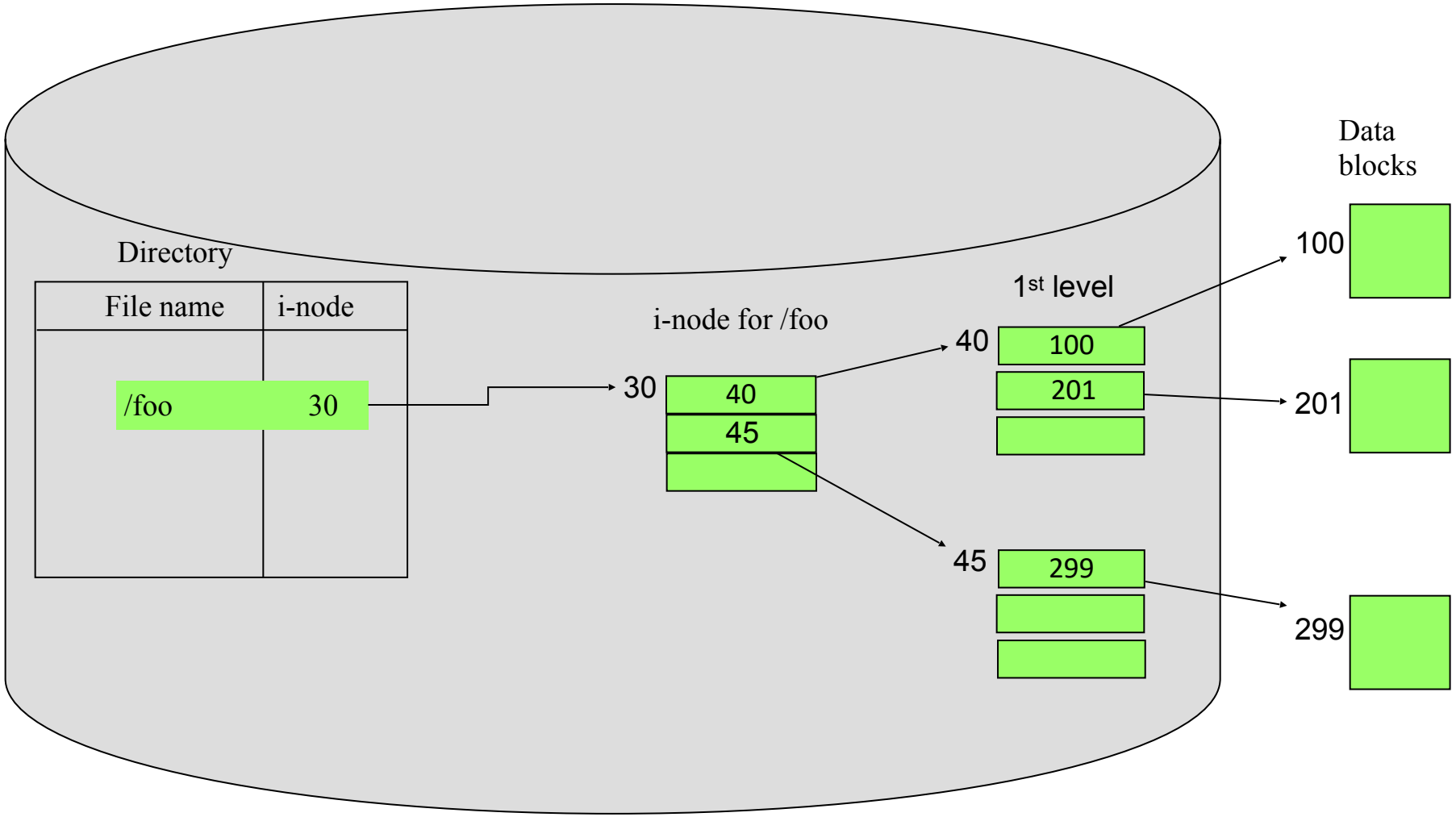
Same as FAT

Potentially better than FAT -- Still depends on seek time, but can cache i-node in memory

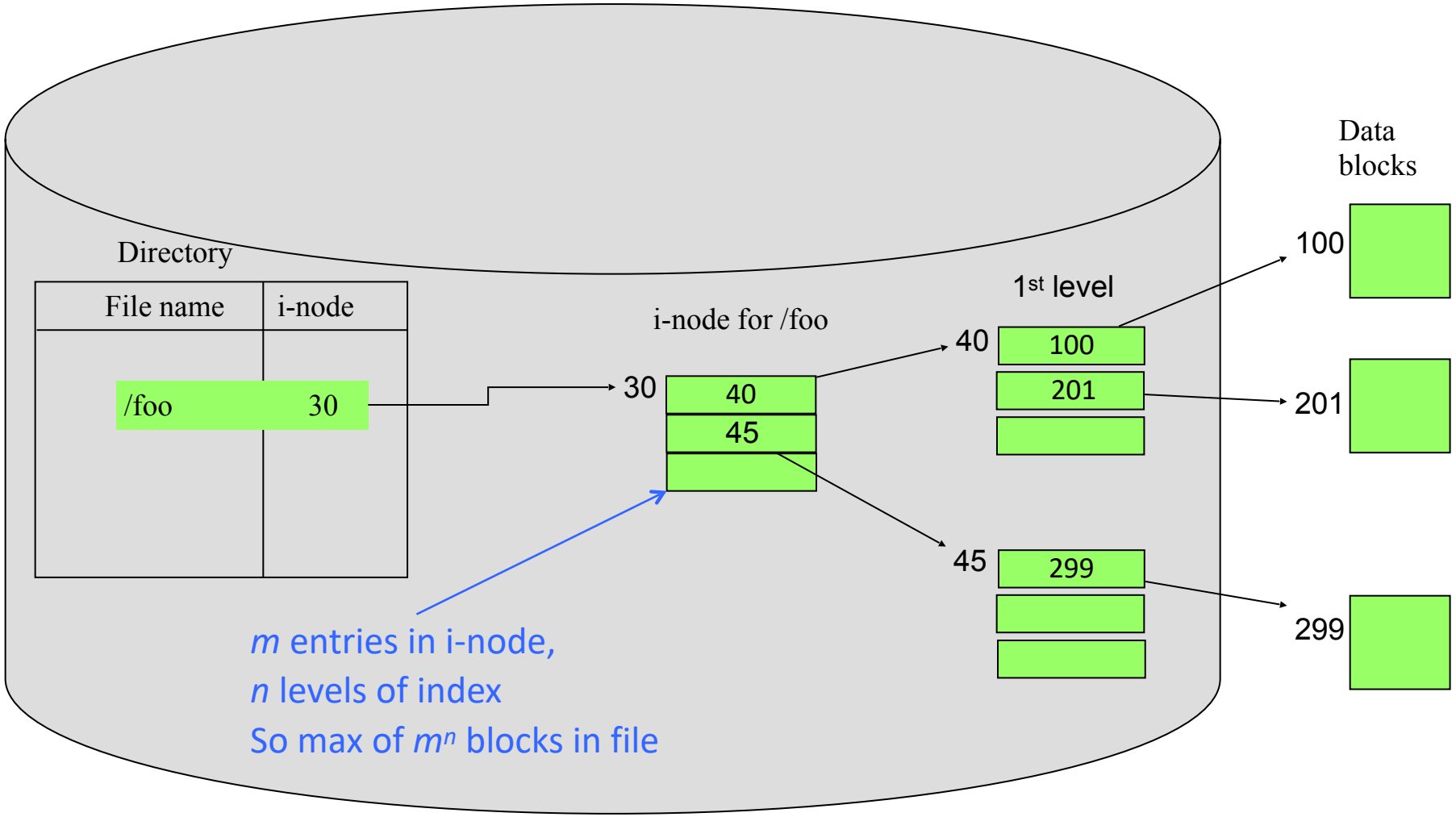
File allocation schemes on the disk

- ✓ Contiguous allocation
- ✓ Contiguous allocation with overflow
- ✓ Linked allocation
- ✓ FAT
- ✓ Indexed allocation
- ➔ Multilevel indexed
- ➔ Hybrid indexed

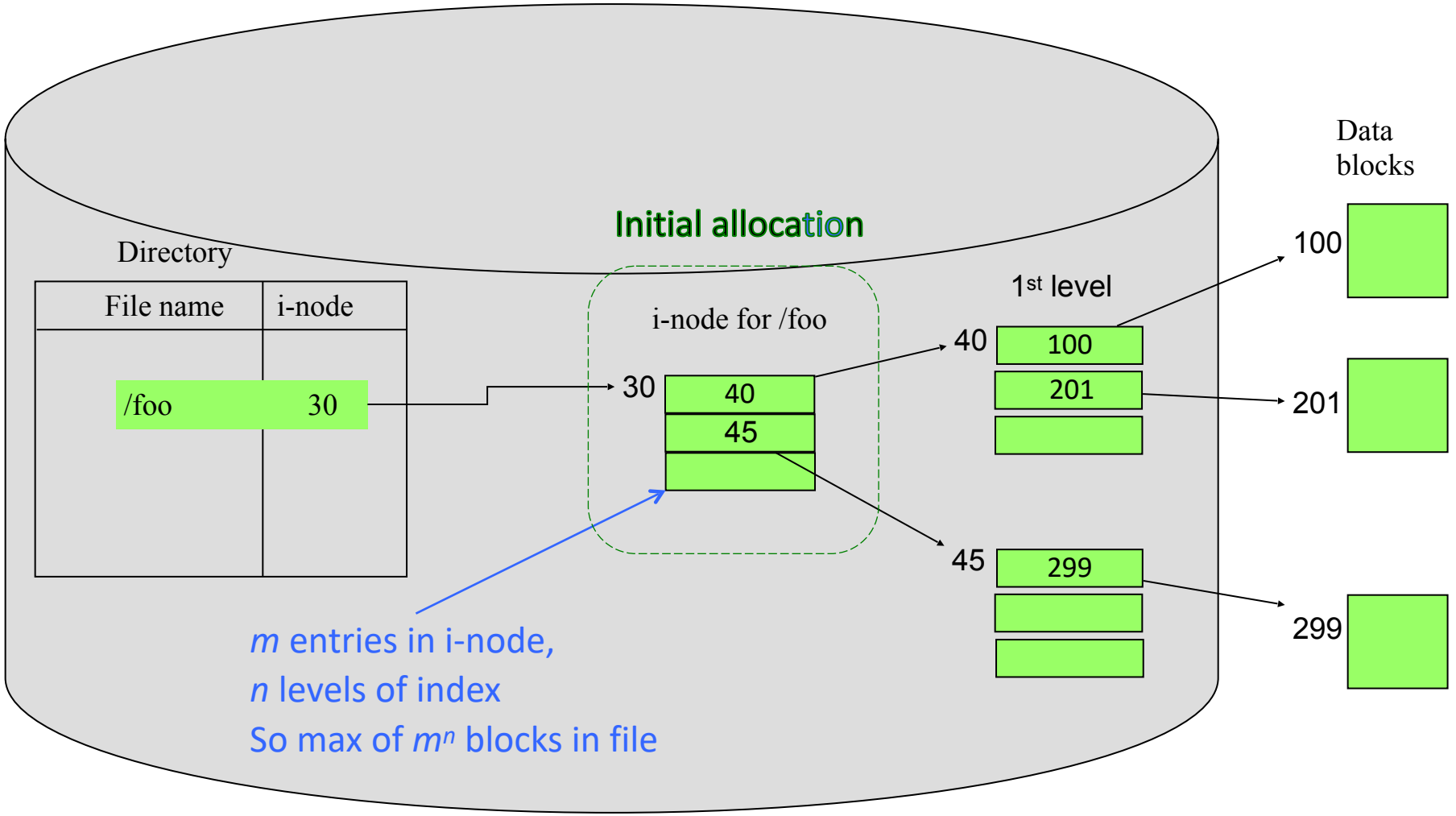
Multilevel indexed allocation



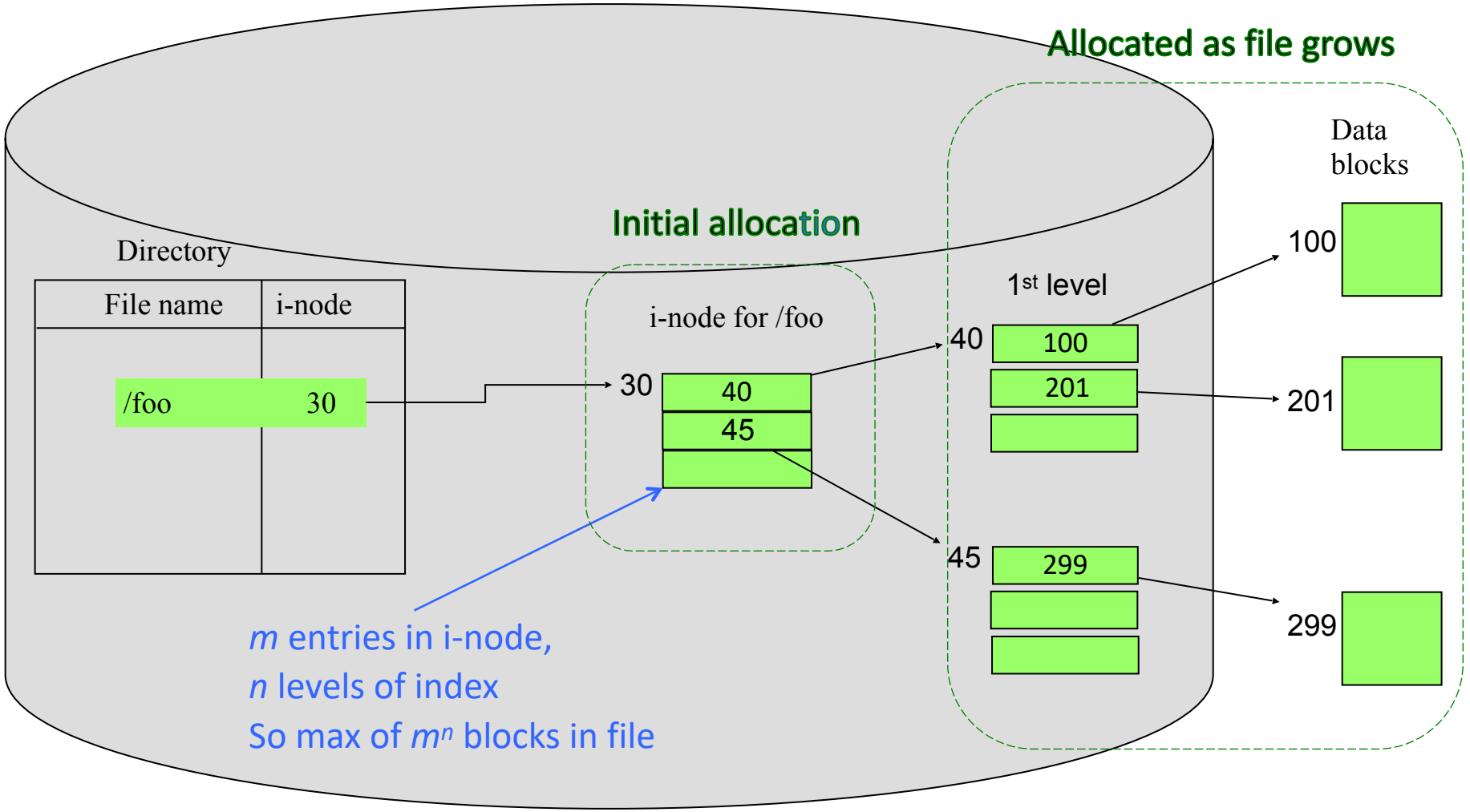
Multilevel indexed allocation



Multilevel indexed allocation



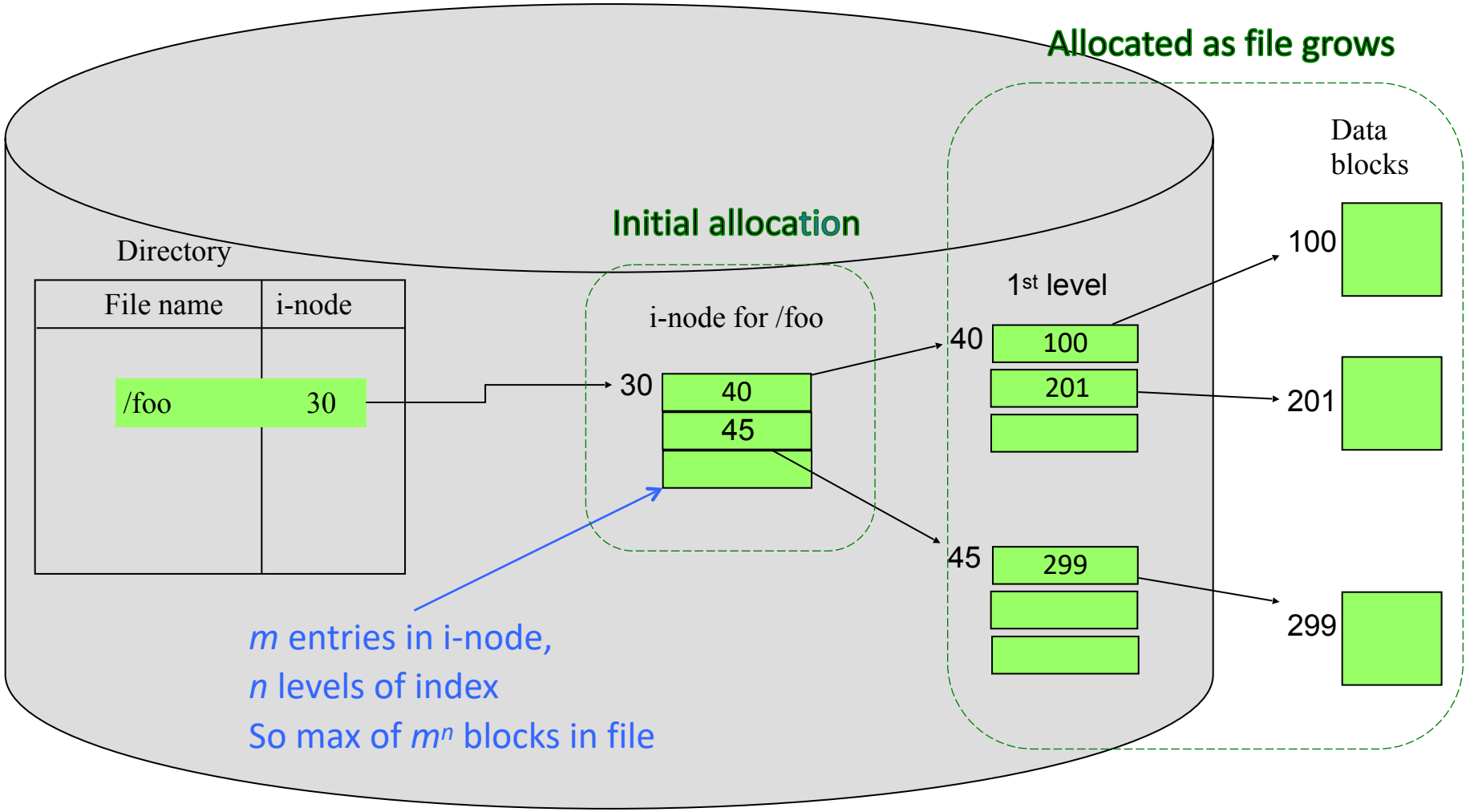
Multilevel indexed allocation



Multilevel indexed allocation

"Every problem in CS can be solved with a layer of indirection…"

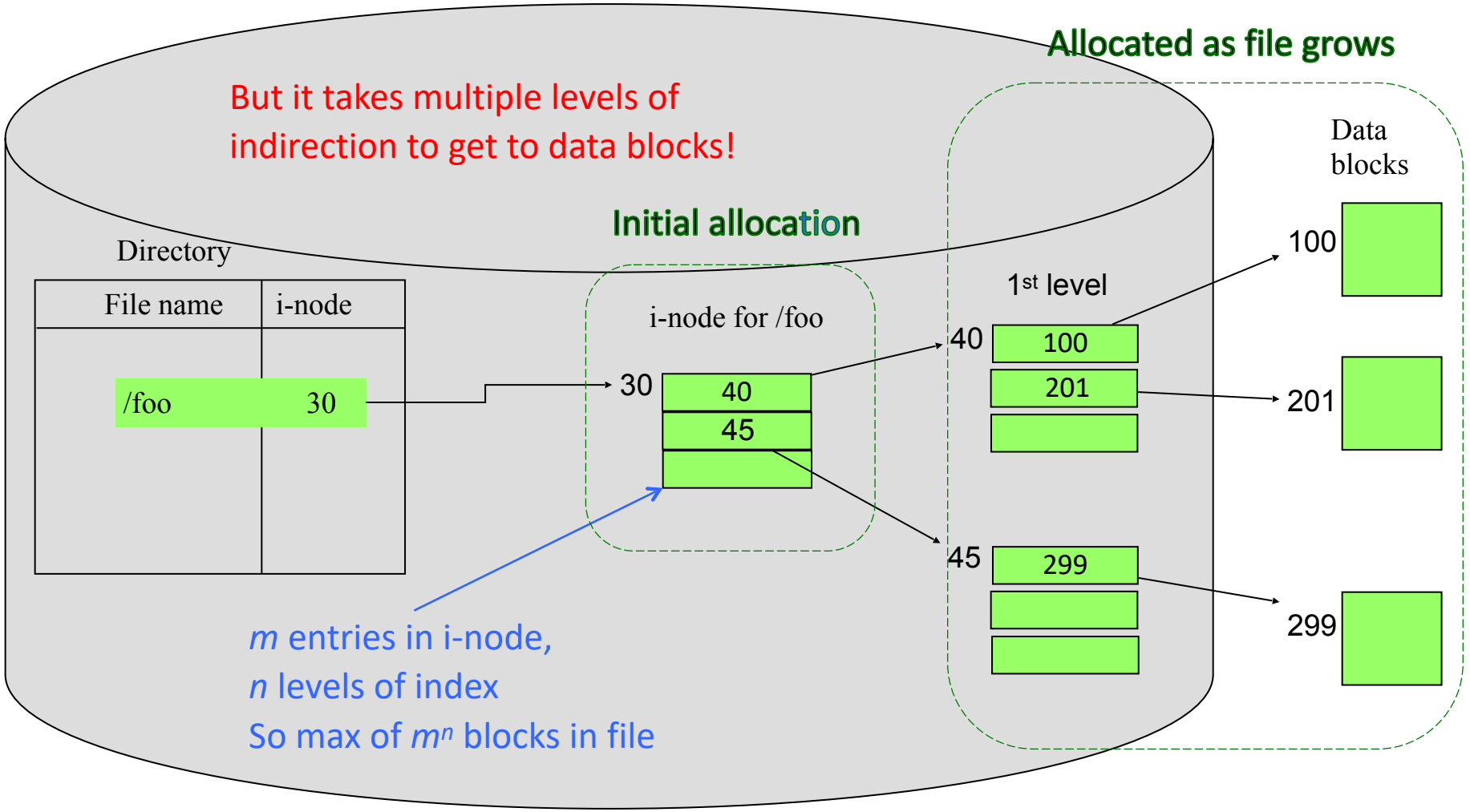
- David Wheeler



Multilevel indexed allocation

"Every problem in CS can be solved with a layer of indirection…"

- David Wheeler

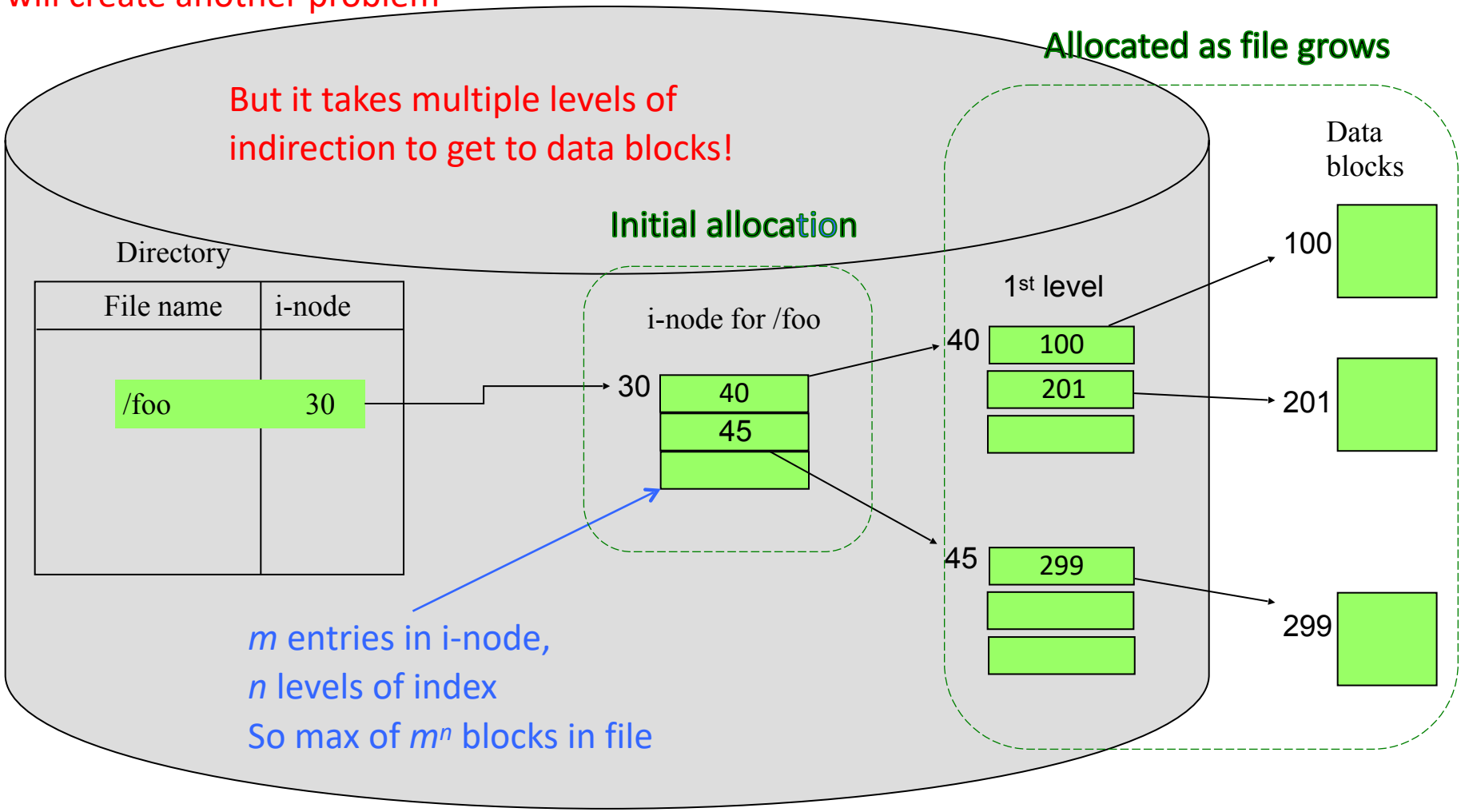


Multilevel indexed allocation

"Every problem in CS can be solved with a layer of indirection…"

"…but that usually will create another problem"

- David Wheeler



Different file sizes

- Small files are a big problem with multilevel indexed allocation
- Why?

Different file sizes

- Small files are a big problem with multilevel indexed allocation
- Why?
- The index block(s) take up as much or more space than the file!

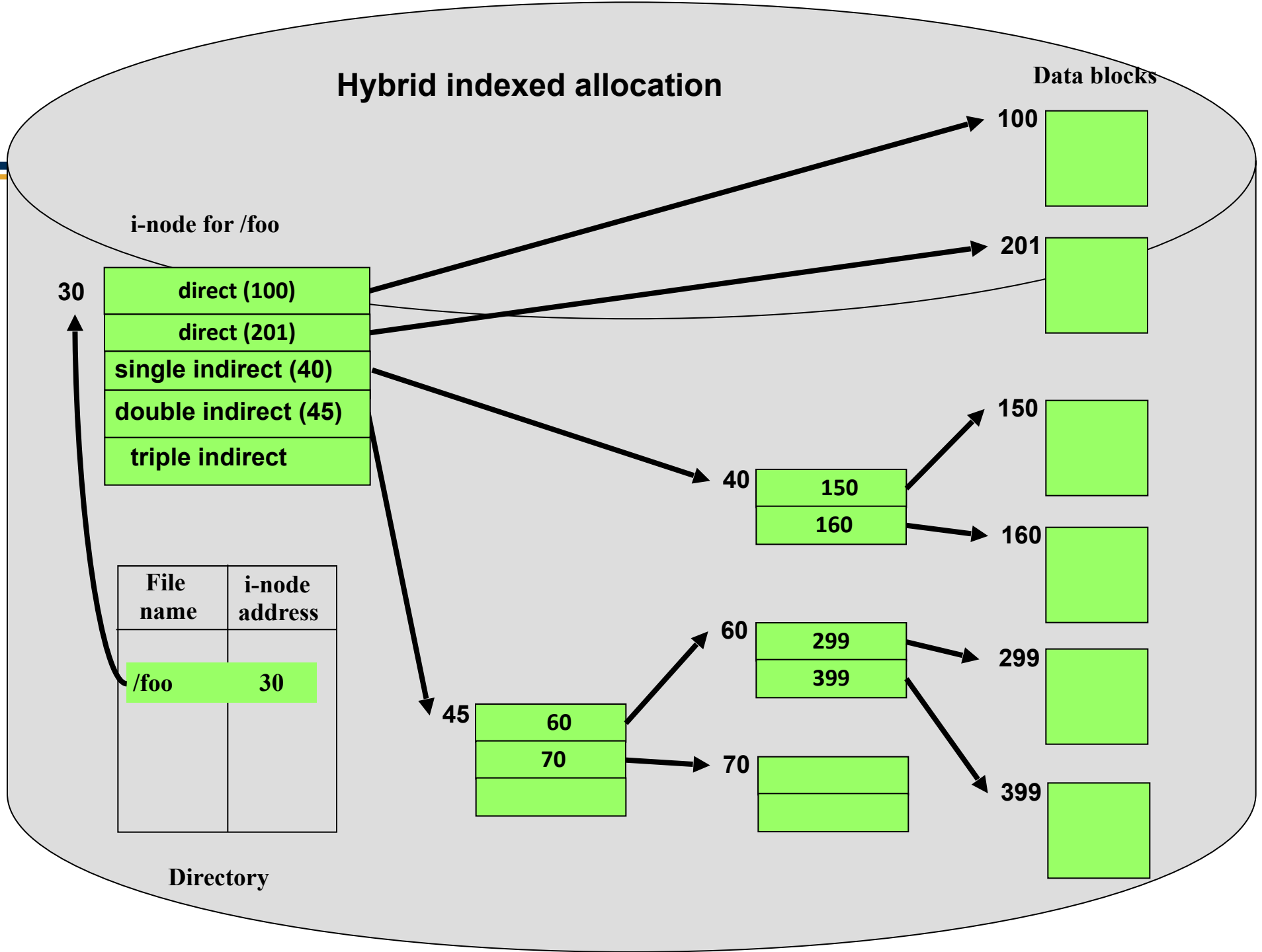
Different file sizes

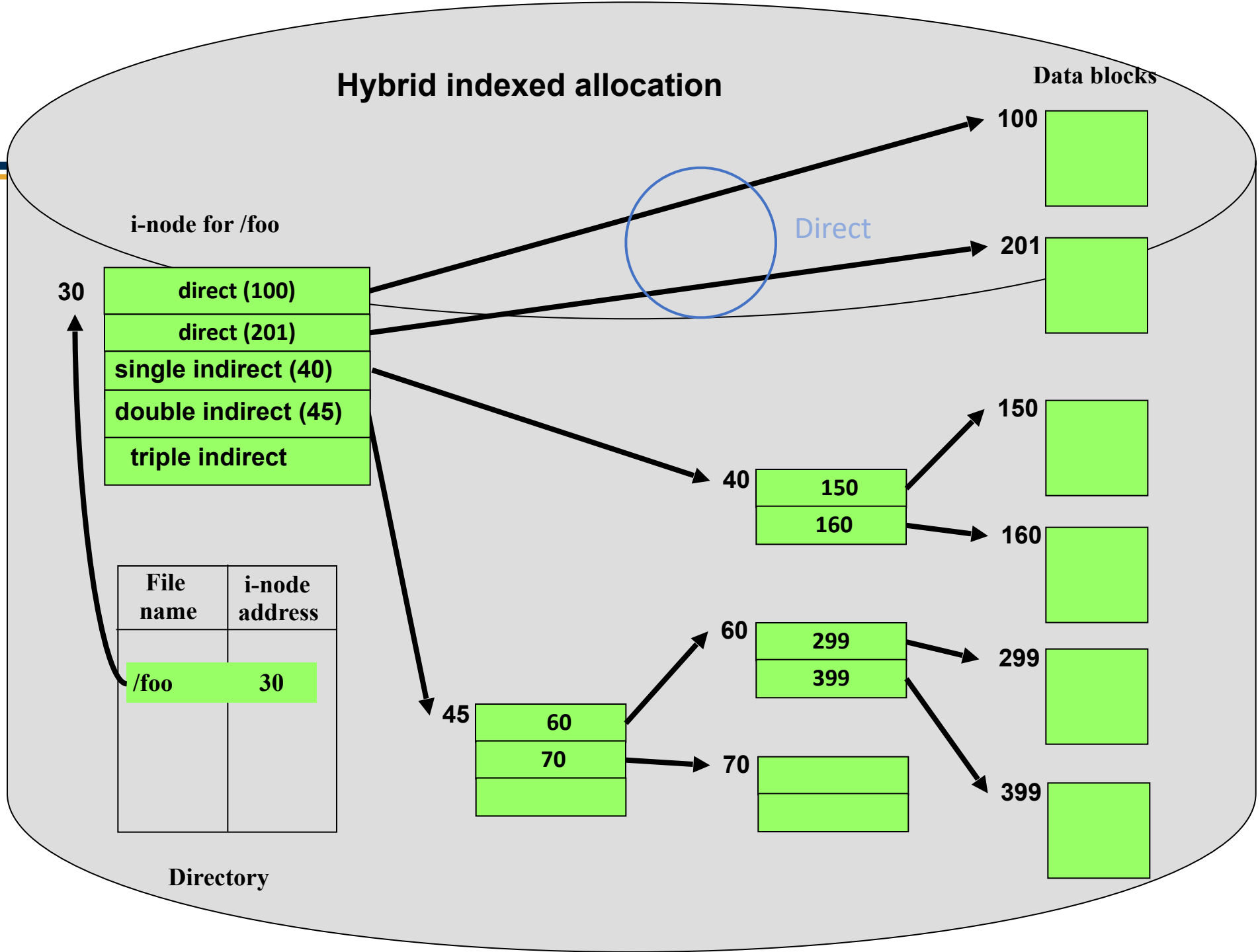
- Small files are a big problem with multilevel indexed allocation
- Why?
- The index block(s) take up as much or more space than the file!
- Too many sequential accesses to get to data

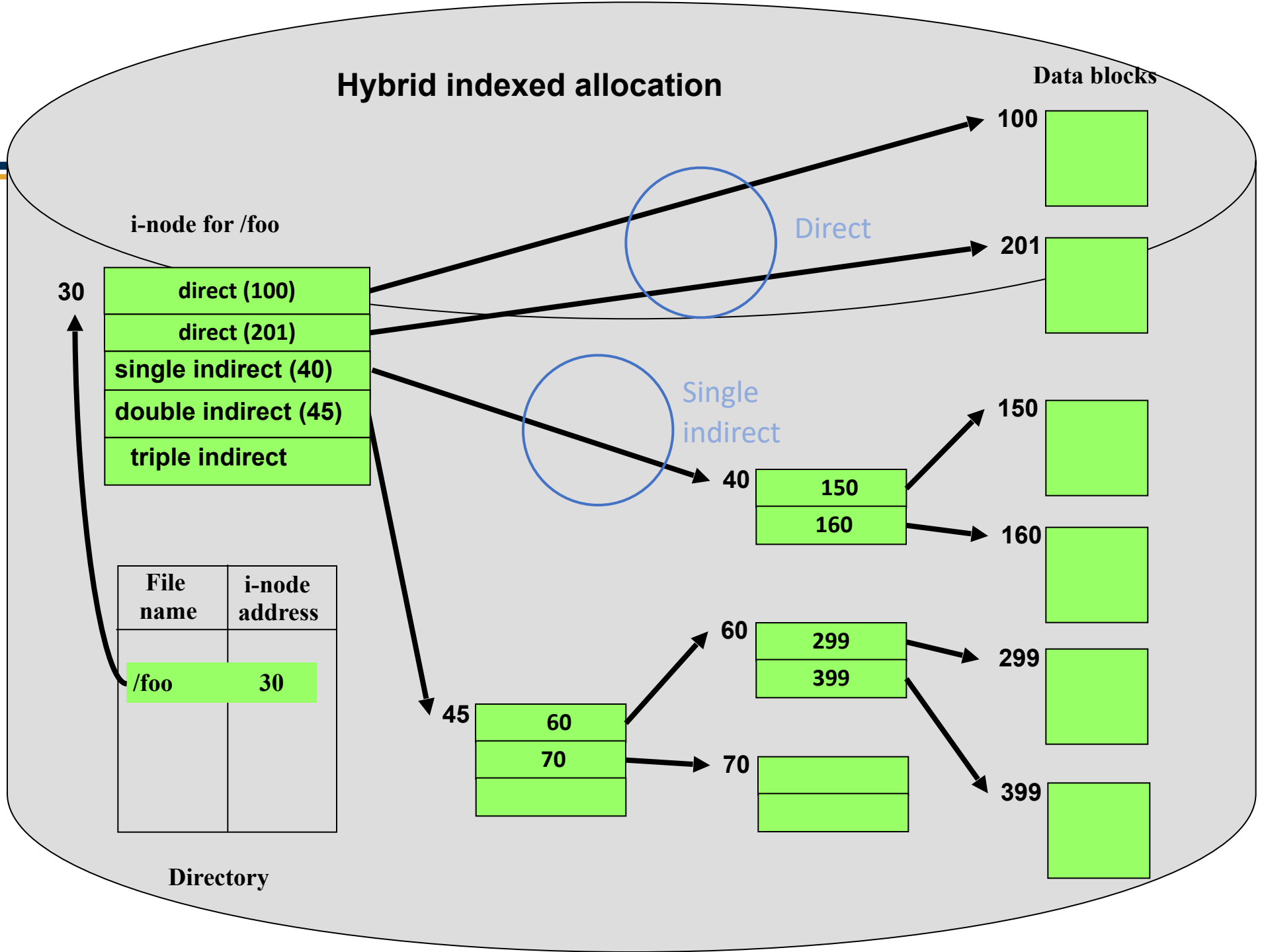
Different file sizes

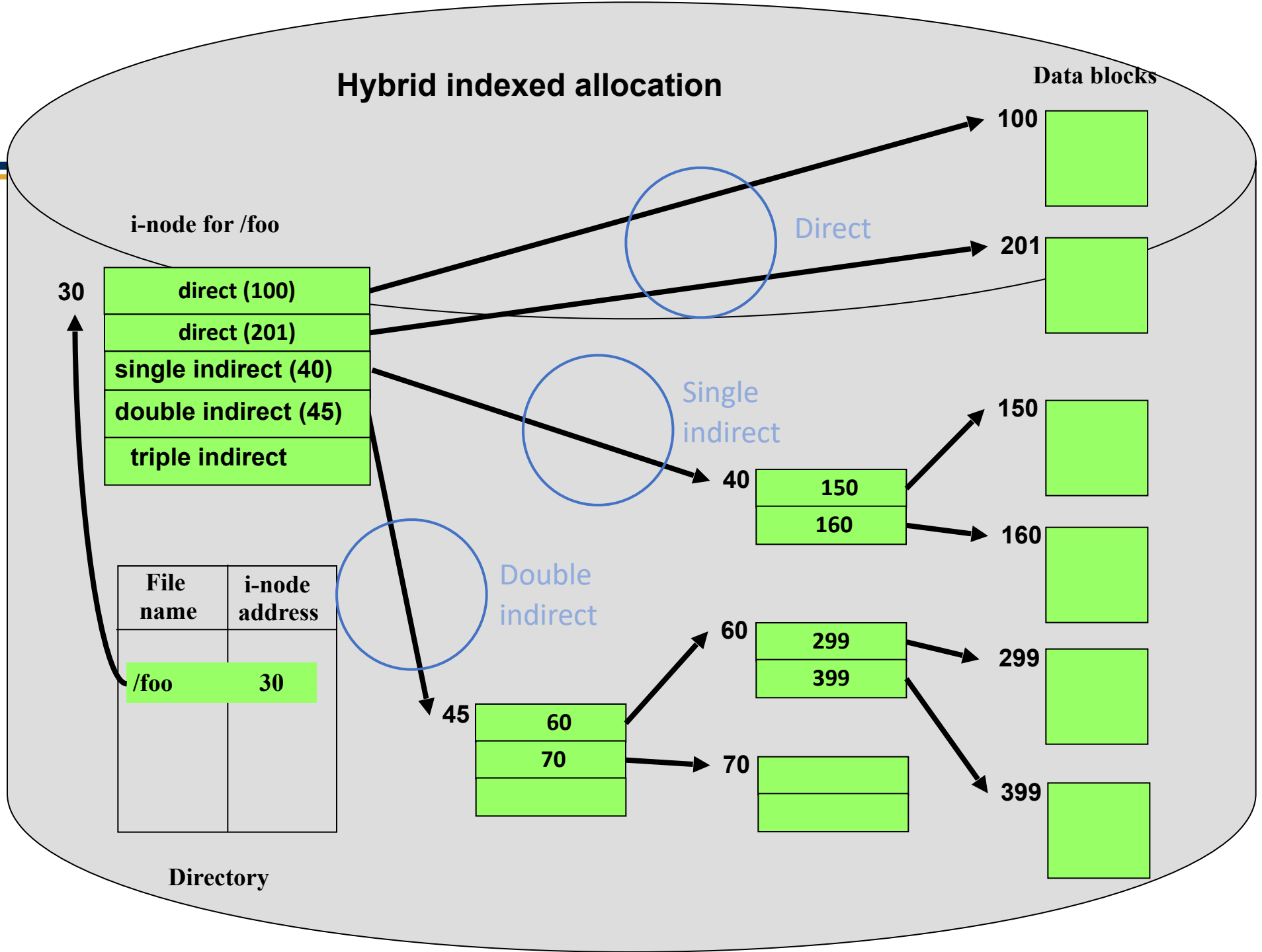
- Small files are a big problem with multilevel indexed allocation
- Why?
- The index block(s) take up as much or more space than the file!
- Too many sequential accesses to get to data

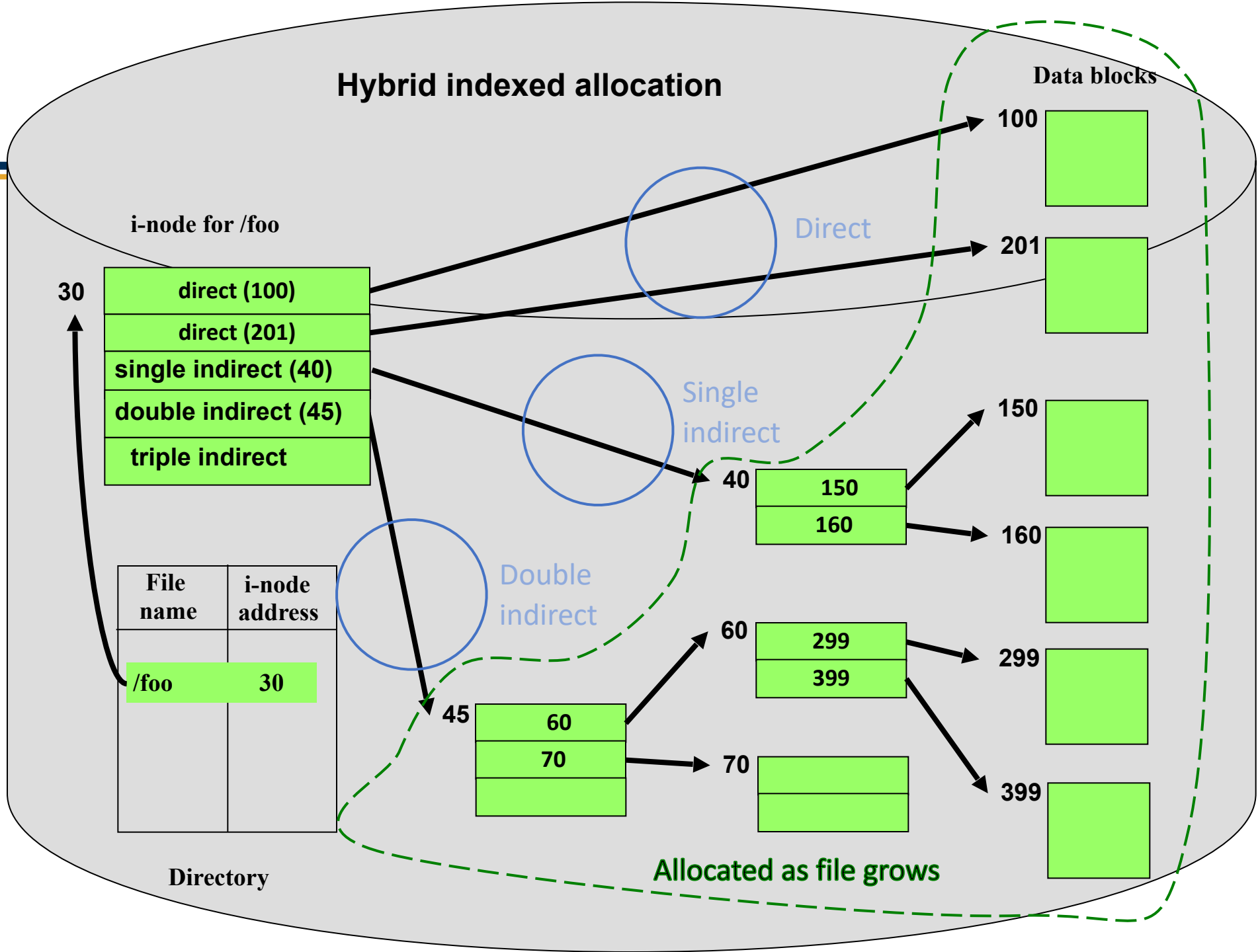
- Enter Hybrid Indexed Allocation


Hybrid indexed allocation
Data blocks
i-node for /foo
30
direct (100)
direct (201)
single indirect (40)
double indirect (45)
triple indirect
100
201
40
150
160
150
160
45
60
70
60
70
299
399
299
399
File name
i-node address
/foo
30
Directory


Hybrid indexed allocation
Data blocks
i-node for /foo
Direct
30
direct (100)
direct (201)
single indirect (40)
double indirect (45)
triple indirect
100
201
40
150
160
150
160
45
60
70
60
70
299
399
299
399
File name
i-node address
/foo
30
Directory


Hybrid indexed allocation
Data blocks
i-node for /foo
30
direct (100)
direct (201)
single indirect (40)
double indirect (45)
triple indirect
Direct
Single indirect
100
201
40
150
160
150
160
45
60
70
60
70
299
399
299
399
File name
i-node address
/foo
30
Directory


Hybrid indexed allocation
Data blocks
i-node for /foo
30
direct (100)
direct (201)
single indirect (40)
double indirect (45)
triple indirect
Direct
Single indirect
Double indirect
100
201
40
150
160
150
160
45
60
70
60
70
299
399
299
399
File name
i-node address
/foo
30
Directory


Hybrid indexed allocation
Data blocks
i-node for /foo
30
direct (100)
direct (201)
single indirect (40)
double indirect (45)
triple indirect
Direct
Single indirect
Double indirect
100
201
40
150
160
150
160
45
60
70
60
70
299
399
299
399
File name
i-node address
/foo
30
Directory
Allocated as file grows

Hybrid indexed allocation

- Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Hybrid indexed allocation

- ✓ Growth?
- Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Hybrid indexed allocation

- ✓ Growth?
- ✓ Quickness of allocation (time)?
- Fragmentation (space)?
- Sequential access?
- Random access?

Hybrid indexed allocation

- ✓ Growth?
- ✓ Quickness of allocation (time)?
- ✓ Fragmentation (space)?
- Sequential access?
- Random access?

Hybrid indexed allocation

- ✓ Growth?
- ✓ Quickness of allocation (time)?
- ✓ Fragmentation (space)?
- ✓ Sequential access?
- Random access?

Similar to FAT

Hybrid indexed allocation

- ✓ Growth?
- ✓ Quickness of allocation (time)?
- ✓ Fragmentation (space)?
- ✓ Sequential access? Similar to FAT
- ✓ Random access? Similar to indexed allocation

Best of both worlds: indexed allocation and multilevel indexed allocation

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex	Very good	Very good	messy	Medium to high	Internal and external fragmentation
Contiguous With Overflow	Contiguous blocks for small files	complex	Very good for small files	Very good for small files	OK	Medium to high	"
Linked List	Non-contiguous blocks	Bit vector	Good but dependent on seek time	Not good	Very good	Small to medium	Excellent
FAT	"	FAT	"	Good but dependent on seek time	"	Small	"
Indexed	"	Bit vector	"	"	limited	"	"
Multilevel Indexed	"	Bit vector	"	"	good	"	"
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex	Very good for small files	Very good for small files	OK	Medium to high	"
Linked List	Non-contiguous blocks	Bit vector	Good but dependent on seek time	Not good	Very good	Small to medium	Excellent
FAT	"	FAT	"	Good but dependent on seek time	"	Small	"
Indexed	"	Bit vector	"	"	limited	"	"
Multilevel Indexed	"	Bit vector	"	"	good	"	"
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex X	Very good for small files ✓	Very good for small files ✓	OK ✓	Medium to high X	" X
Linked List	Non-contiguous blocks	Bit vector	Good but dependent on seek time	Not good	Very good	Small to medium	Excellent
FAT	"	FAT	"	Good but dependent on seek time	"	Small	"
Indexed	"	Bit vector	"	"	limited	"	"
Multilevel Indexed	"	Bit vector	"	"	good	"	"
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex X	Very good for small files ✓	Very good for small files ✓	OK ✓	Medium to high X	" X
Linked List	Non-contiguous blocks	Bit vector ✓	Good but dependent on seek time ✓	Not good X	Very good ✓	Small to medium	Excellent ✓
FAT	"	FAT	"	Good but dependent on seek time	"	Small	"
Indexed	"	Bit vector	"	"	limited	"	"
Multilevel Indexed	"	Bit vector	"	"	good	"	"
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex X	Very good for small files ✓	Very good for small files ✓	OK ✓	Medium to high X	" X
Linked List	Non-contiguous blocks	Bit vector ✓	Good but dependent on seek time ✓	Not good X	Very good ✓	Small to medium	Excellent ✓
FAT	"	FAT ✓	" ✓	Good but dependent on seek time ✓	" ✓	Small ✓	" ✓
Indexed	"	Bit vector	"	"	limited	"	"
Multilevel Indexed	"	Bit vector	"	"	good	"	"
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex X	Very good for small files ✓	Very good for small files ✓	OK ✓	Medium to high X	" X
Linked List	Non-contiguous blocks	Bit vector ✓	Good but dependent on seek time ✓	Not good X	Very good ✓	Small to medium	Excellent ✓
FAT Partitioning not good for user	"	FAT ✓	" ✓	Good but dependent on seek time ✓	" ✓	Small ✓	" ✓
Indexed	"	Bit vector	"	"	limited	"	"
Multilevel Indexed	"	Bit vector	"	"	good	"	"
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex X	Very good for small files ✓	Very good for small files ✓	OK ✓	Medium to high X	" X
Linked List	Non-contiguous blocks	Bit vector ✓	Good but dependent on seek time ✓	Not good X	Very good ✓	Small to medium	Excellent ✓
FAT Partitioning not good for user	"	FAT ✓	" ✓	Good but dependent on seek time ✓	" ✓	Small ✓	" ✓
Indexed	"	Bit vector ✓	" ✓	" ✓	limited X	" ✓	" ✓
Multilevel Indexed	"	Bit vector	"	"	good	"	"
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex X	Very good for small files ✓	Very good for small files ✓	OK ✓	Medium to high X	" X
Linked List	Non-contiguous blocks	Bit vector ✓	Good but dependent on seek time ✓	Not good X	Very good ✓	Small to medium	Excellent ✓
FAT Partitioning not good for user	"	FAT ✓	" ✓	Good but dependent on seek time ✓	" ✓	Small ✓	" ✓
Indexed	"	Bit vector ✓	" ✓	" ✓	limited X	" ✓	" ✓
Multilevel Indexed	"	Bit vector ✓	" ✓	" ✓	good ✓	" ✓	" ✓
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex X	Very good for small files ✓	Very good for small files ✓	OK ✓	Medium to high X	" X
Linked List	Non-contiguous blocks	Bit vector ✓	Good but dependent on seek time ✓	Not good X	Very good ✓	Small to medium	Excellent ✓
FAT Partitioning not good for user	"	FAT ✓	" ✓	Good but dependent on seek time ✓	" ✓	Small ✓	" ✓
Indexed	"	Bit vector ✓	" ✓	" ✓	limited X	" ✓	" ✓
Multilevel Indexed Small Files ☹	"	Bit vector ✓	" ✓	" ✓	good ✓	" ✓	" ✓
Hybrid	"	Bit vector	"	"	"	"	"

Allocation Strategy	File represen-tation	Free list mainte-nance	Sequential Access	Random Access	File growth	Allocation Overhead	Space Efficiency
Contiguous	Contiguous blocks	complex X	Very good ✓	Very good ✓	messy X	Medium to high X	Internal and external fragmentation X
Contiguous With Overflow	Contiguous blocks for small files	complex X	Very good for small files ✓	Very good for small files ✓	OK ✓	Medium to high X	" X
Linked List	Non-contiguous blocks	Bit vector ✓	Good but dependent on seek time ✓	Not good X	Very good ✓	Small to medium	Excellent ✓
FAT Partitioning not good for user	"	FAT ✓	" ✓	Good but dependent on seek time ✓	" ✓	Small ✓	" ✓
Indexed	"	Bit vector ✓	" ✓	" ✓	limited X	" ✓	" ✓
Multilevel Indexed Small Files ☹	"	Bit vector ✓	" ✓	" ✓	good ✓	" ✓	" ✓
Hybrid	"	Bit vector ✓	" ✓	" ✓	" ✓	" ✓	" ✓

Hybrid FS Example

Given the following:

Size of index block = 512 bytes
Size of data block = 2048 bytes
Size of pointer = 8 bytes (to index or data blocks)

The i-node consists of

2 direct data block pointers,
1 single indirect pointer, and
1 double indirect pointer.

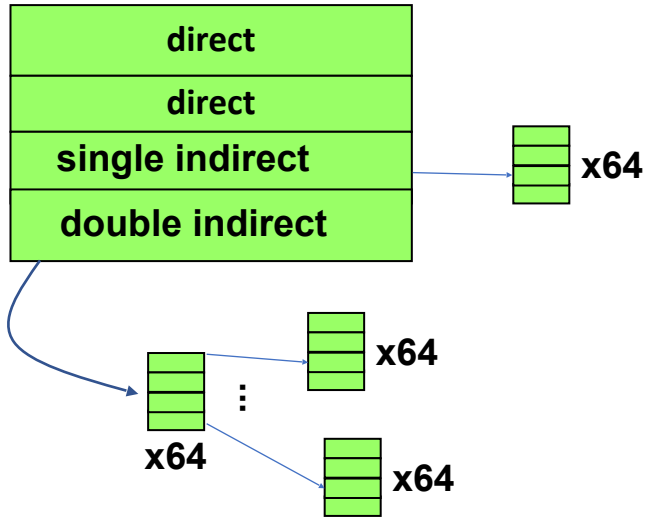
An index block is used for the i-node as well as for the index blocks that store pointers to other index blocks and data blocks. Note that the index blocks and data blocks are allocated on a need basis.

(a) What is the maximum size (in bytes) of a file that can be stored in this file system?
(b) How many data blocks are needed for storing the same data file of 266 KB?
(c) How many index blocks are needed for storing a data file of size 266 KB?

Example: solution

Size of index block = 512 bytes
Size of data block = 2048 bytes
Size of pointer = 8 bytes (to index or data blocks)

The i-node consists of
2 direct data block pointers,
1 single indirect pointer, and
1 double indirect pointer.



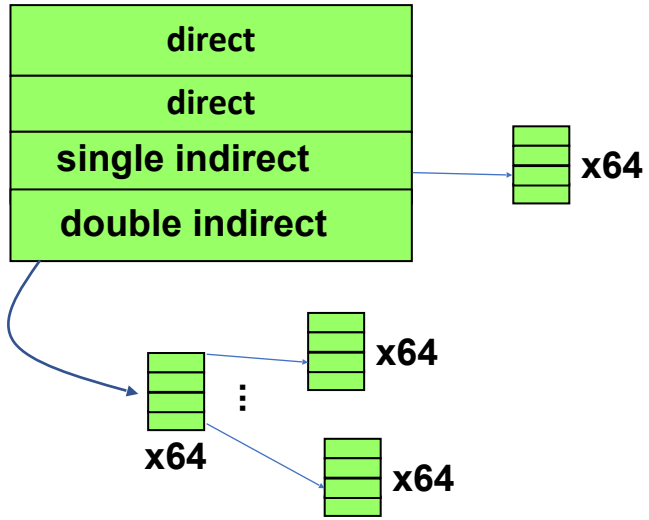
Example: solution

(a) Maximum file size:

- An index block can hold 512/8 = 64 entries
- 2 direct data block pointers in i-node
- 64 entries in first-level index
- 64 * 64 entries in second-level index
- Total: 4,162 blocks * 2,048 bytes = 8,523,776 bytes

Size of index block = 512 bytes
Size of data block = 2048 bytes
Size of pointer = 8 bytes (to index or data blocks)

The i-node consists of
2 direct data block pointers,
1 single indirect pointer, and
1 double indirect pointer.



Example: solution

(a) Maximum file size:

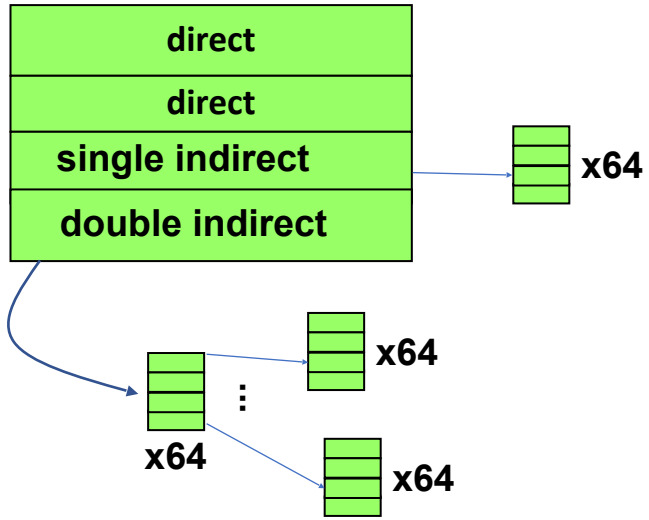
- An index block can hold 512/8 = 64 entries
- 2 direct data block pointers in i-node
- 64 entries in first-level index
- 64 * 64 entries in second-level index
- Total: 4,162 blocks * 2,048 bytes = 8,523,776 bytes

(b) Number of data blocks to hold 266KB:

- $266 * 2^{10} / 2048 = 266 / 2 = 133$ blocks

Size of index block = 512 bytes
Size of data block = 2048 bytes
Size of pointer = 8 bytes (to index or data blocks)

The i-node consists of
2 direct data block pointers,
1 single indirect pointer, and
1 double indirect pointer.



Example: solution

Size of index block = 512 bytes
Size of data block = 2048 bytes
Size of pointer = 8 bytes (to index or data blocks)

The i-node consists of
2 direct data block pointers,
1 single indirect pointer, and
1 double indirect pointer.

(a) Maximum file size:

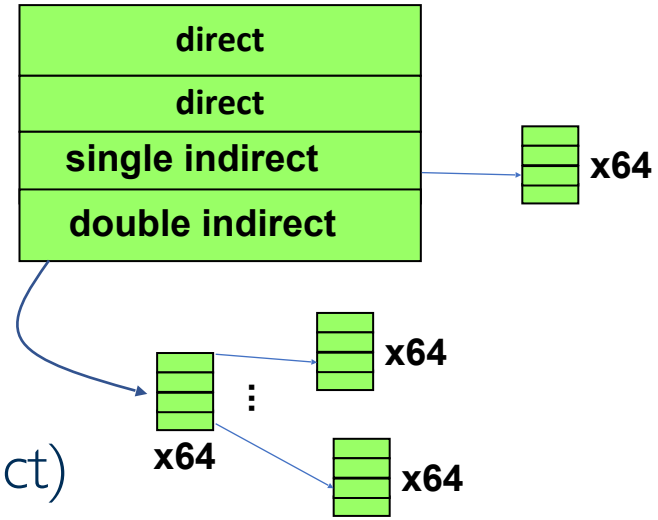
- An index block can hold 512/8 = 64 entries
- 2 direct data block pointers in i-node
- 64 entries in first-level index
- 64 * 64 entries in second-level index
- Total: 4,162 blocks * 2,048 bytes = 8,523,776 bytes

(b) Number of data blocks to hold 266KB:

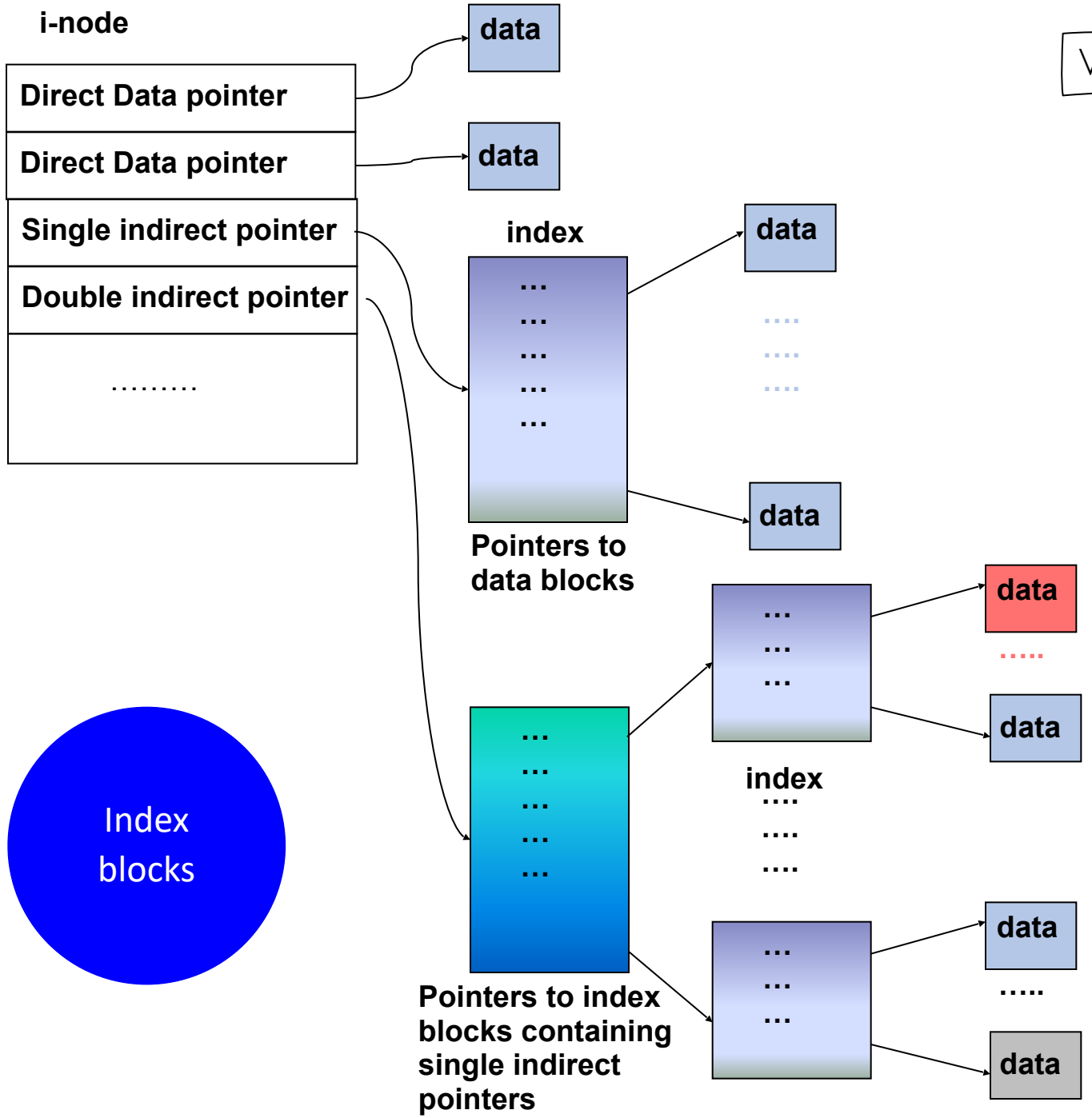
- $266 * 2^{10} / 2048 = 266 / 2 = 133$ blocks

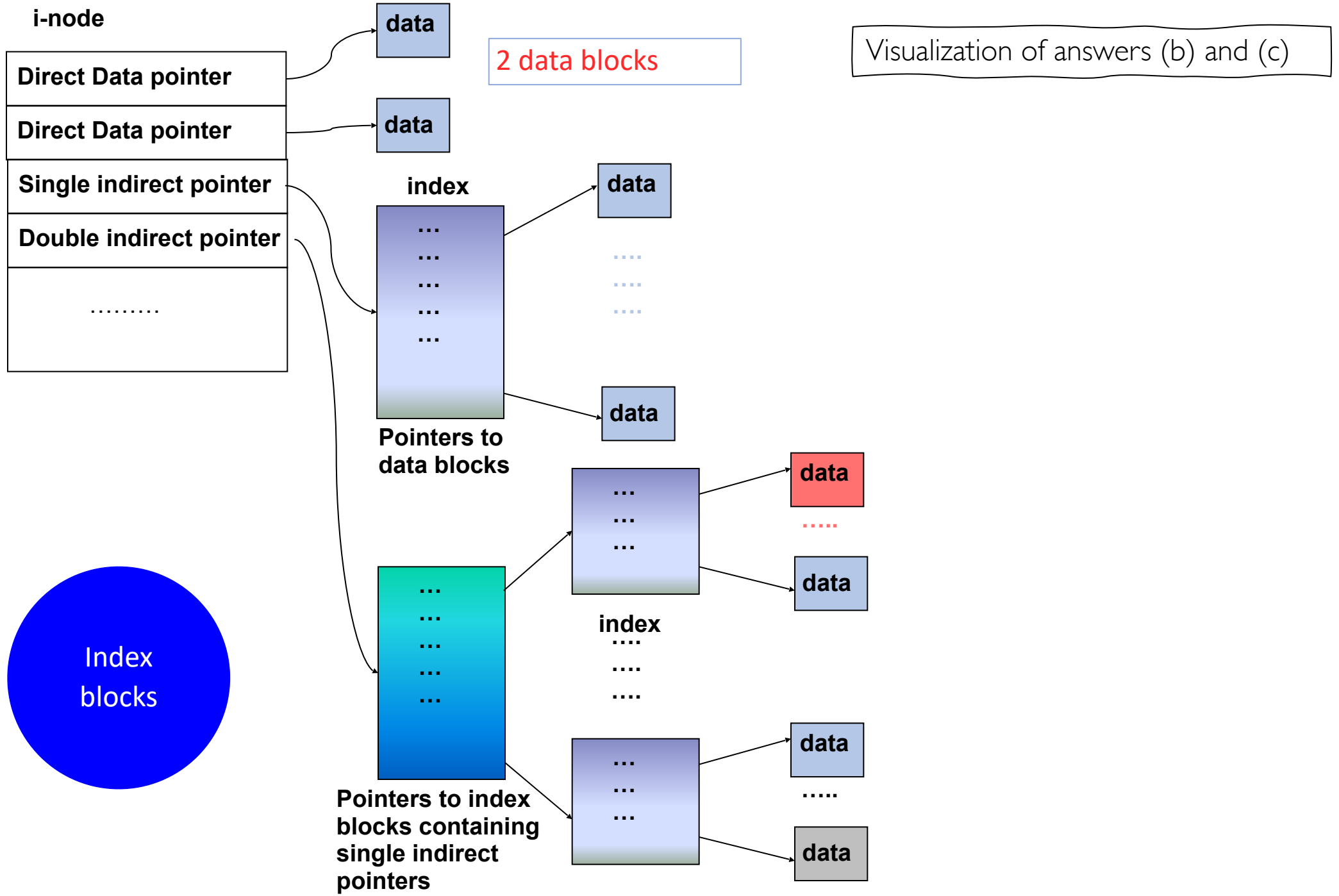
(c) How many index blocks to hold 266KB (133 blocks):

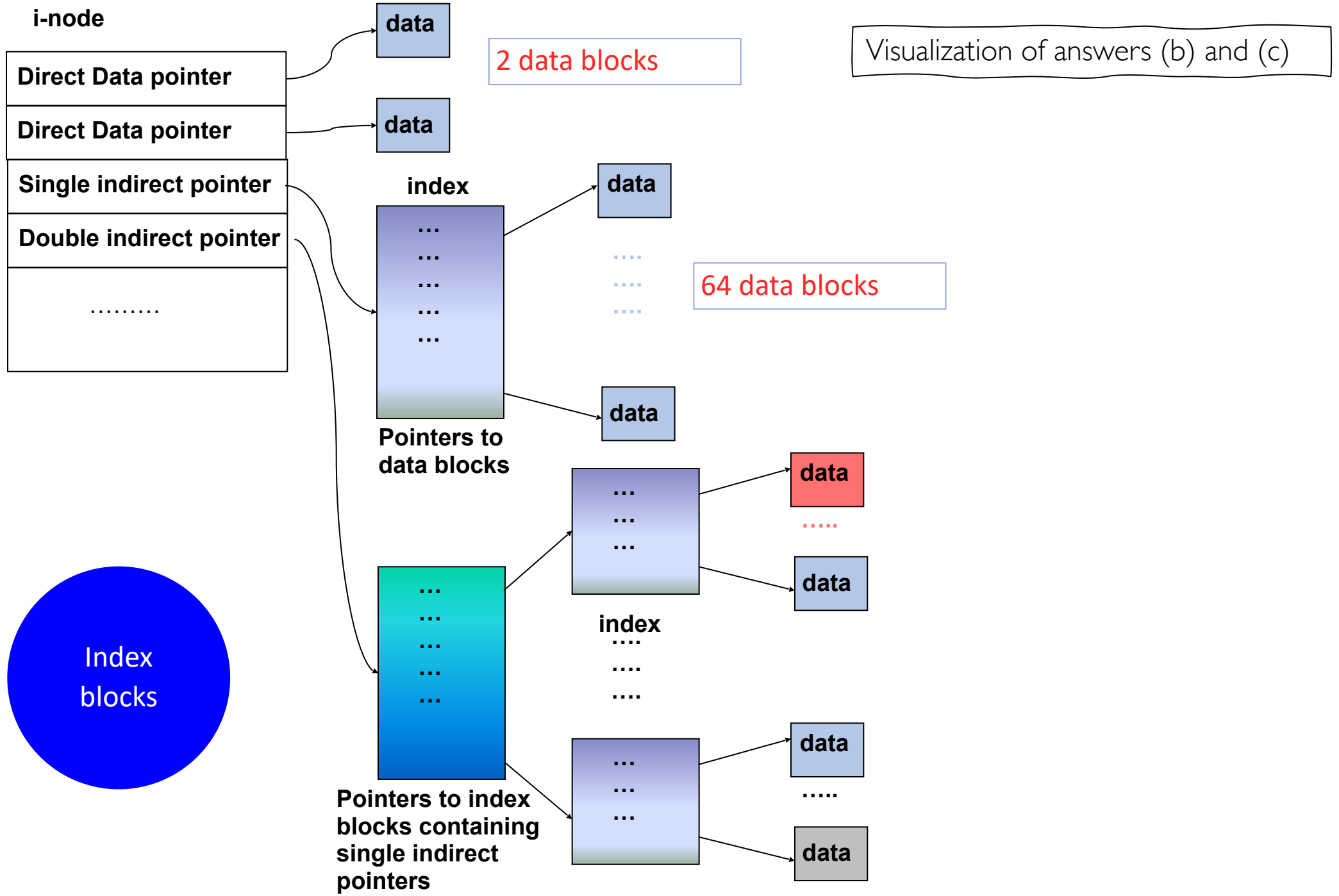
- 2 direct + 64 first-level blocks + (64 + 3) second-level blocks
- 1 i-node + 1 first-level index (under single indirect)
  + 1 first-level index + 2 second-level index (under double indirect)

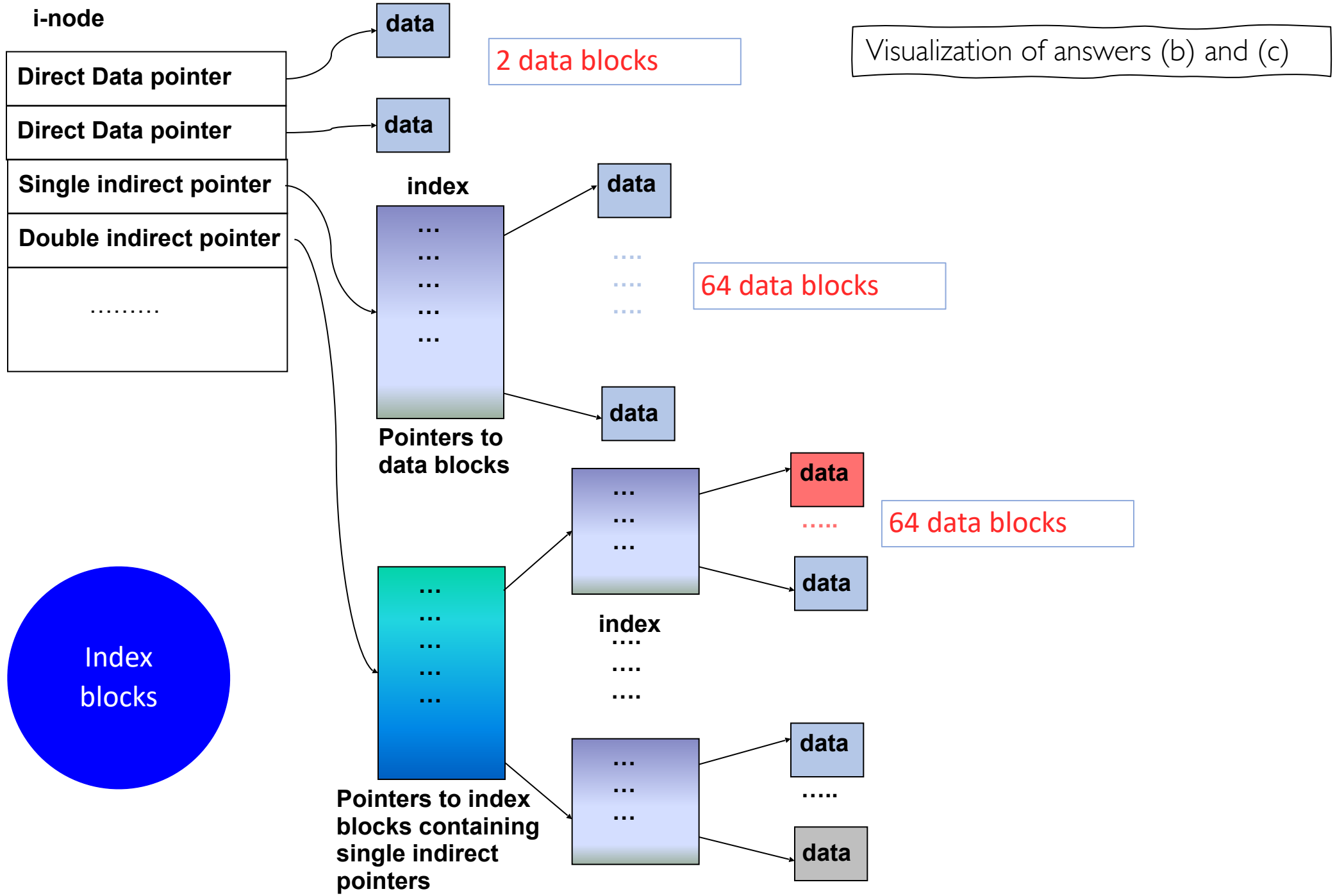


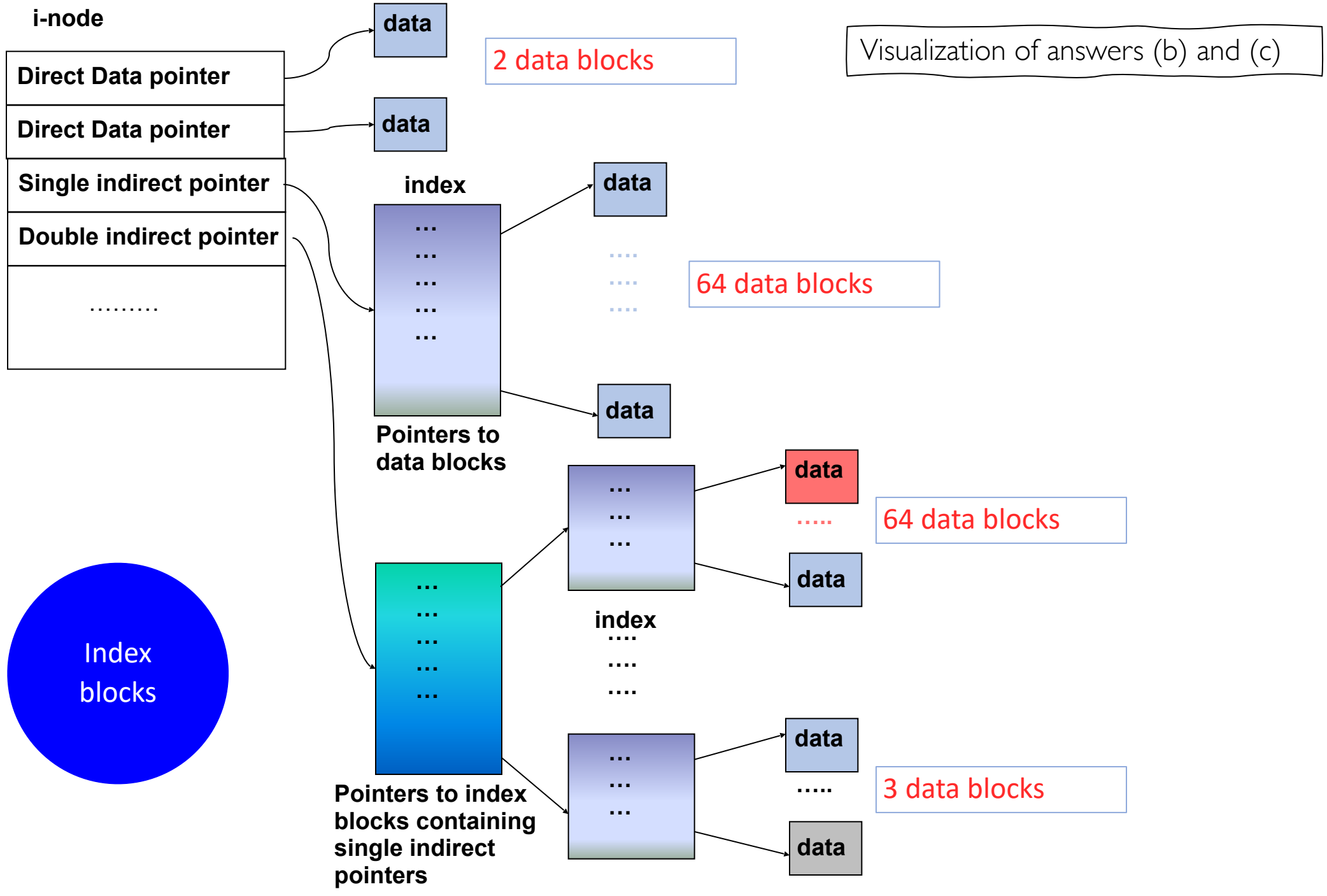
Visualization of answers (b) and (c)

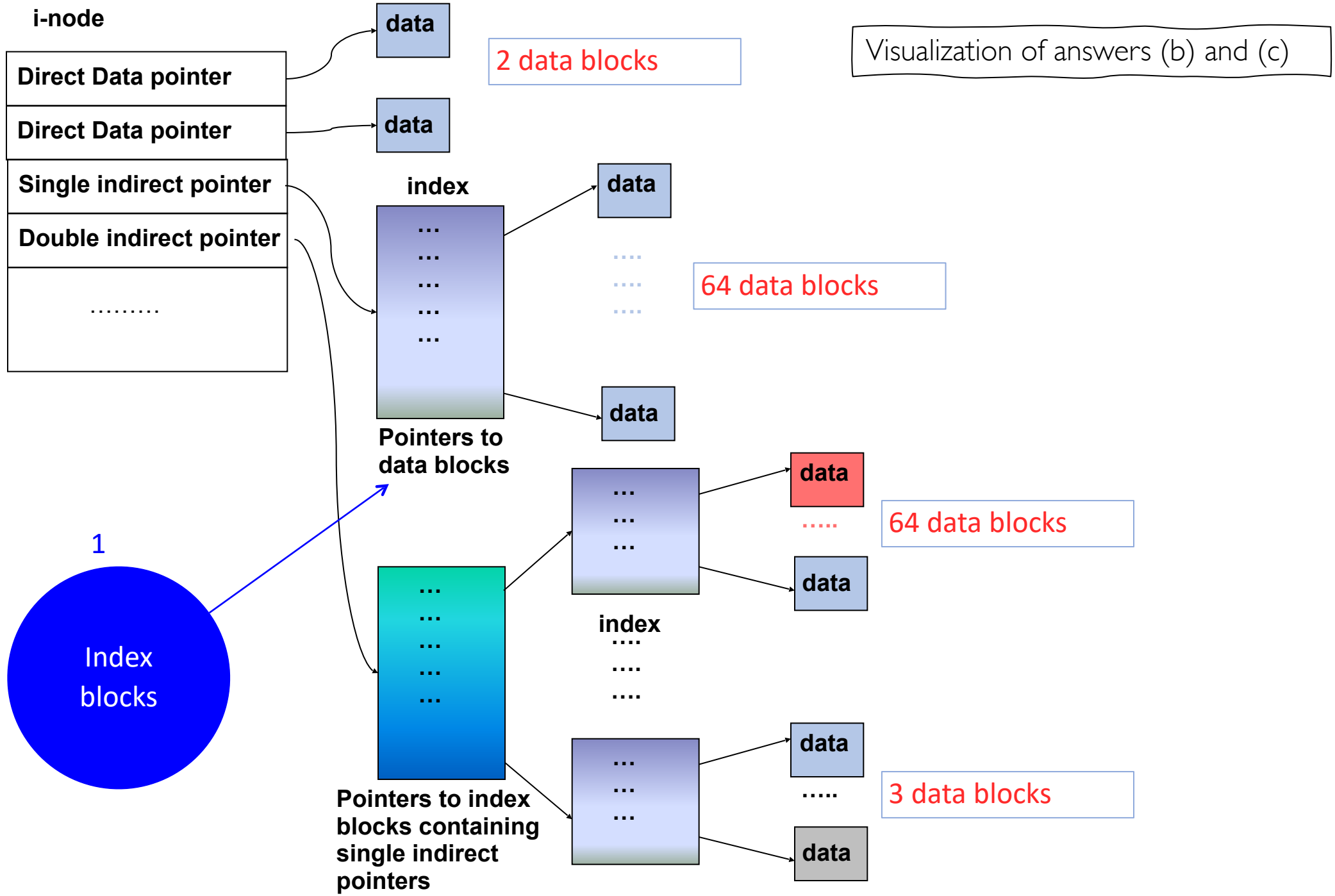


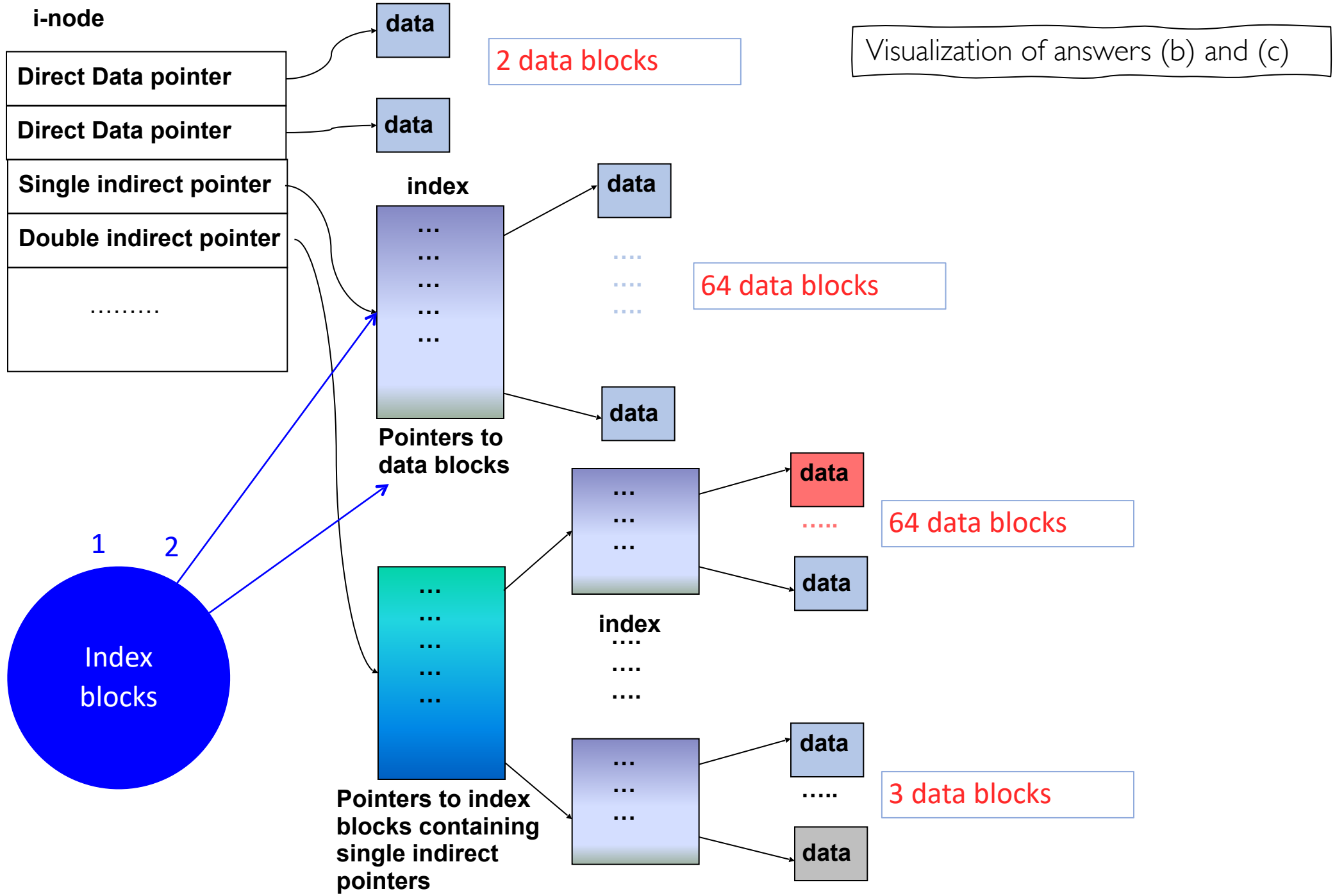

i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
Visualization of answers (b) and (c)
index
data
data
Pointers to
data blocks
data
data
index
Index
blocks
Pointers to index
blocks containing
single indirect
pointers
data
data

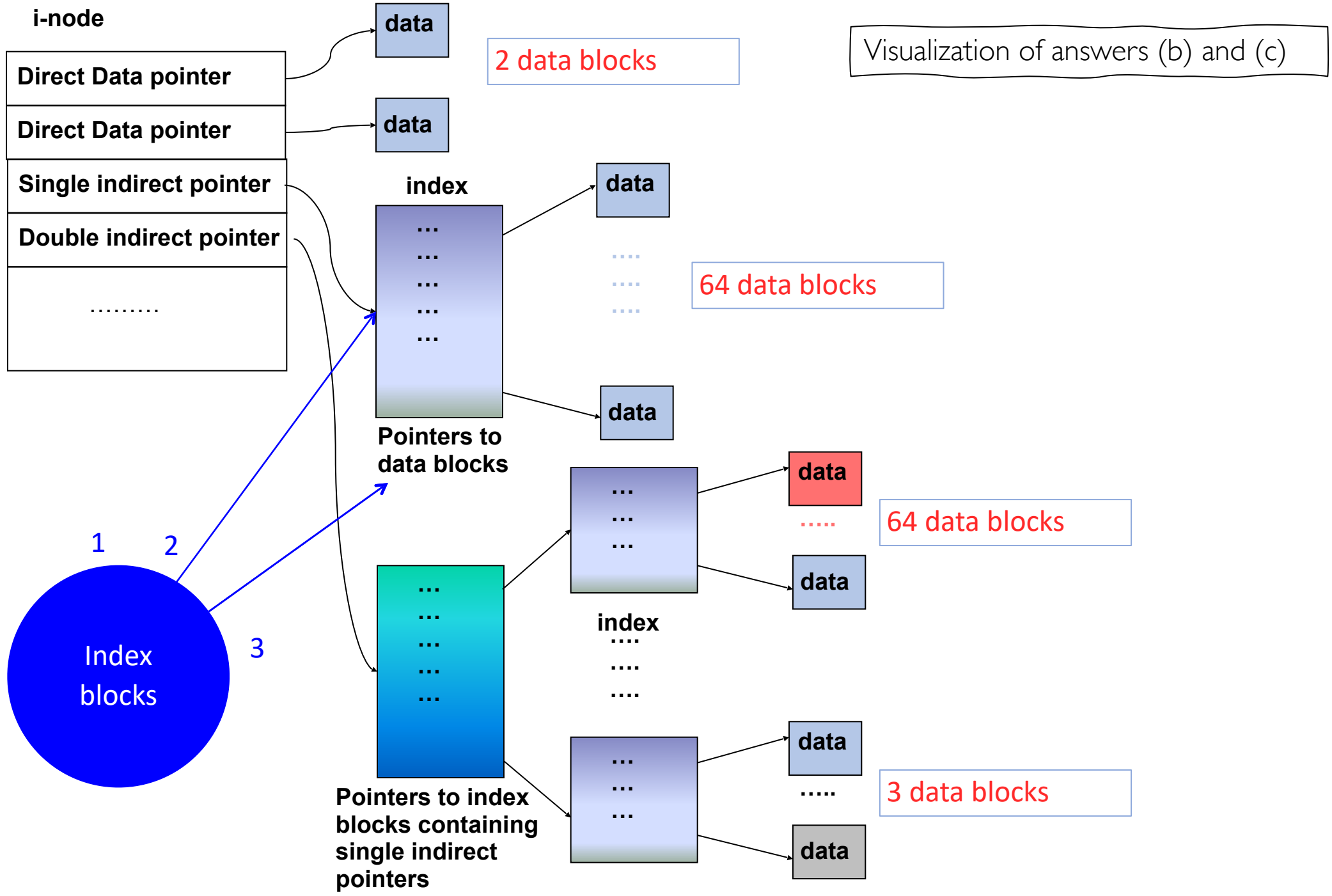

i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
Visualization of answers (b) and (c)
index
data
64 data blocks
data
Pointers to data blocks
data
index
data
Pointers to index blocks containing single indirect pointers
data
data
Index blocks

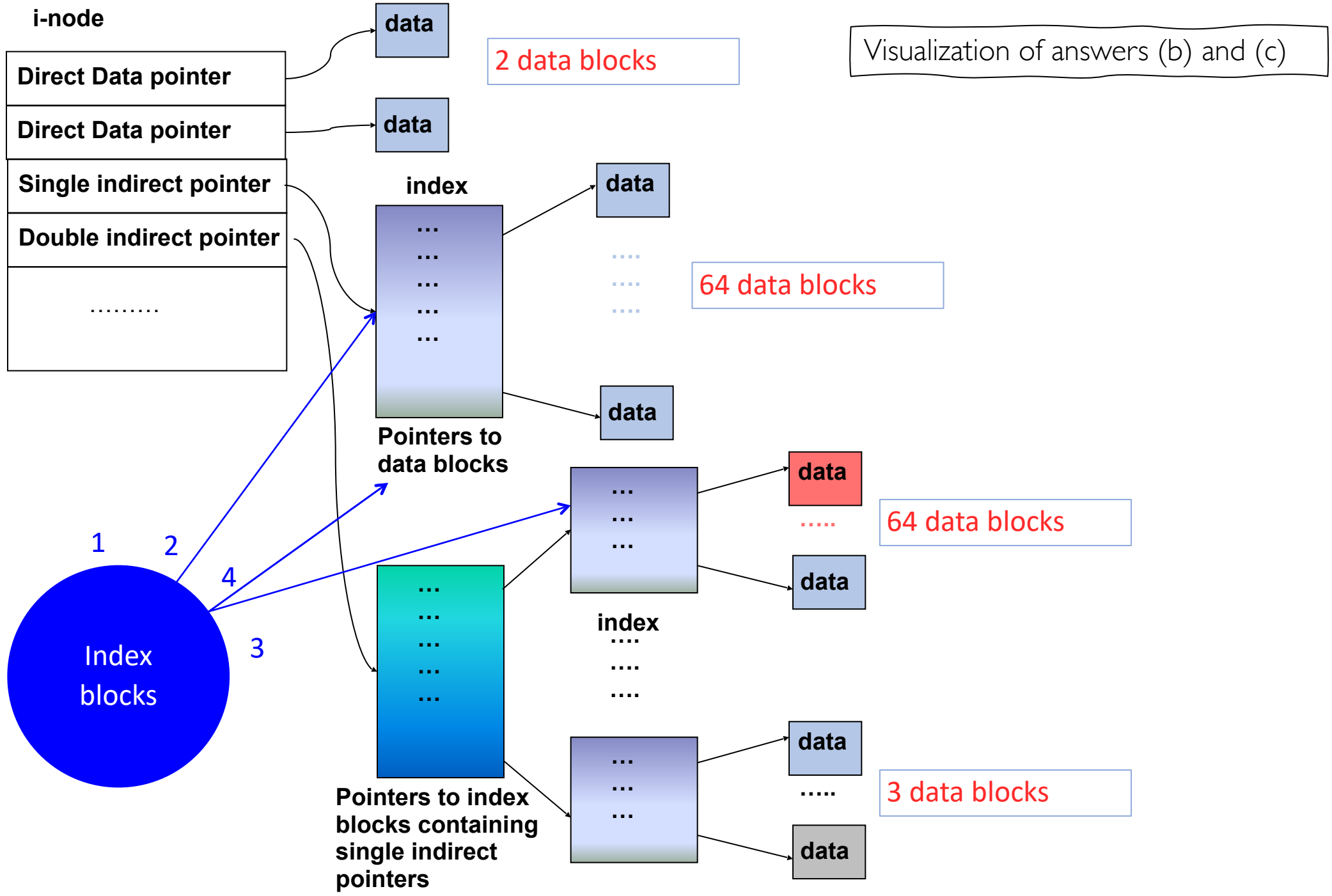

i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
Visualization of answers (b) and (c)
index
data
64 data blocks
data
Pointers to data blocks
data
64 data blocks
data
index
Index blocks
Pointers to index blocks containing single indirect pointers
data
data

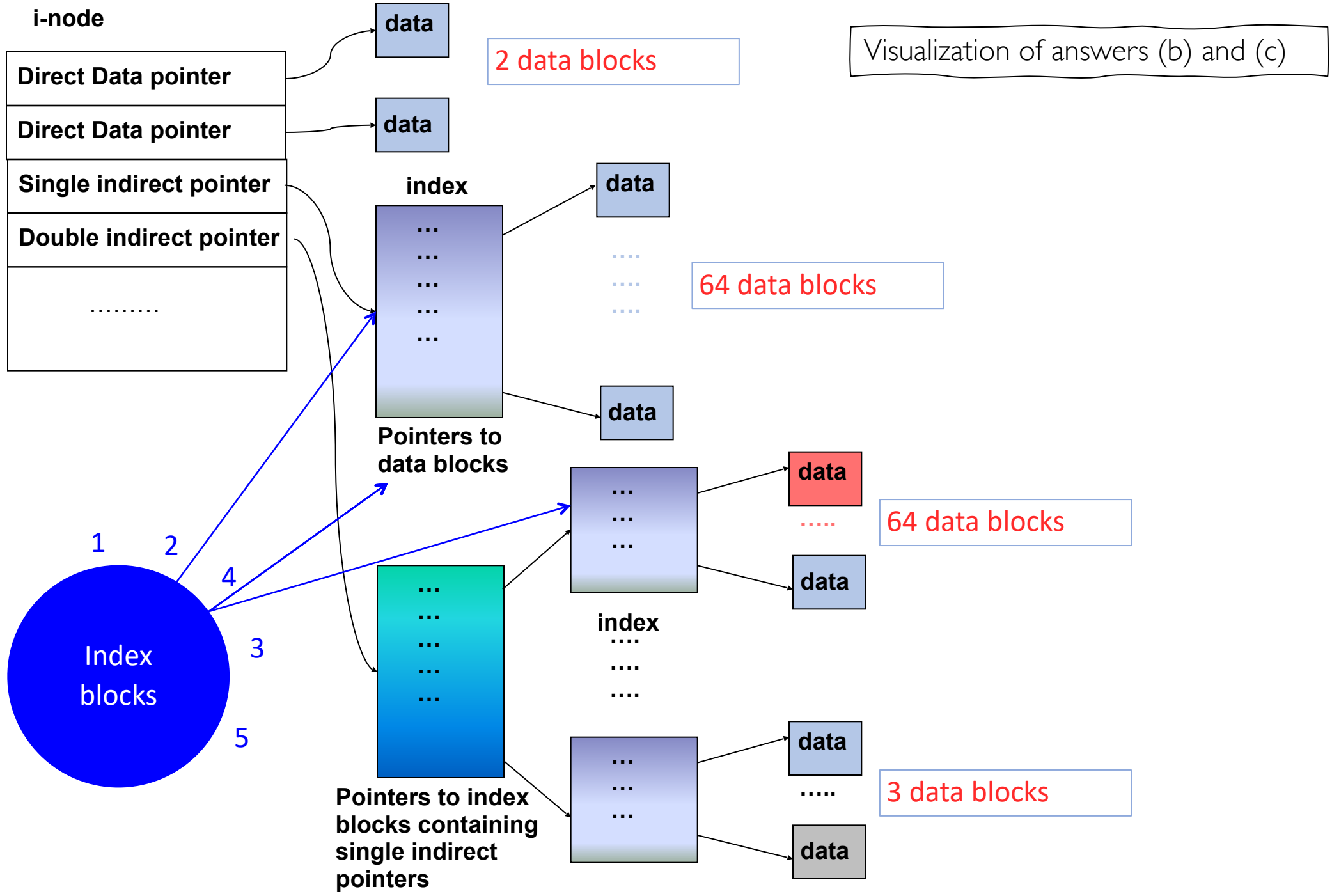

i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
Visualization of answers (b) and (c)
index
data
64 data blocks
data
Pointers to data blocks
data
64 data blocks
data
index
Index blocks
Pointers to index blocks containing single indirect pointers
data
3 data blocks
data

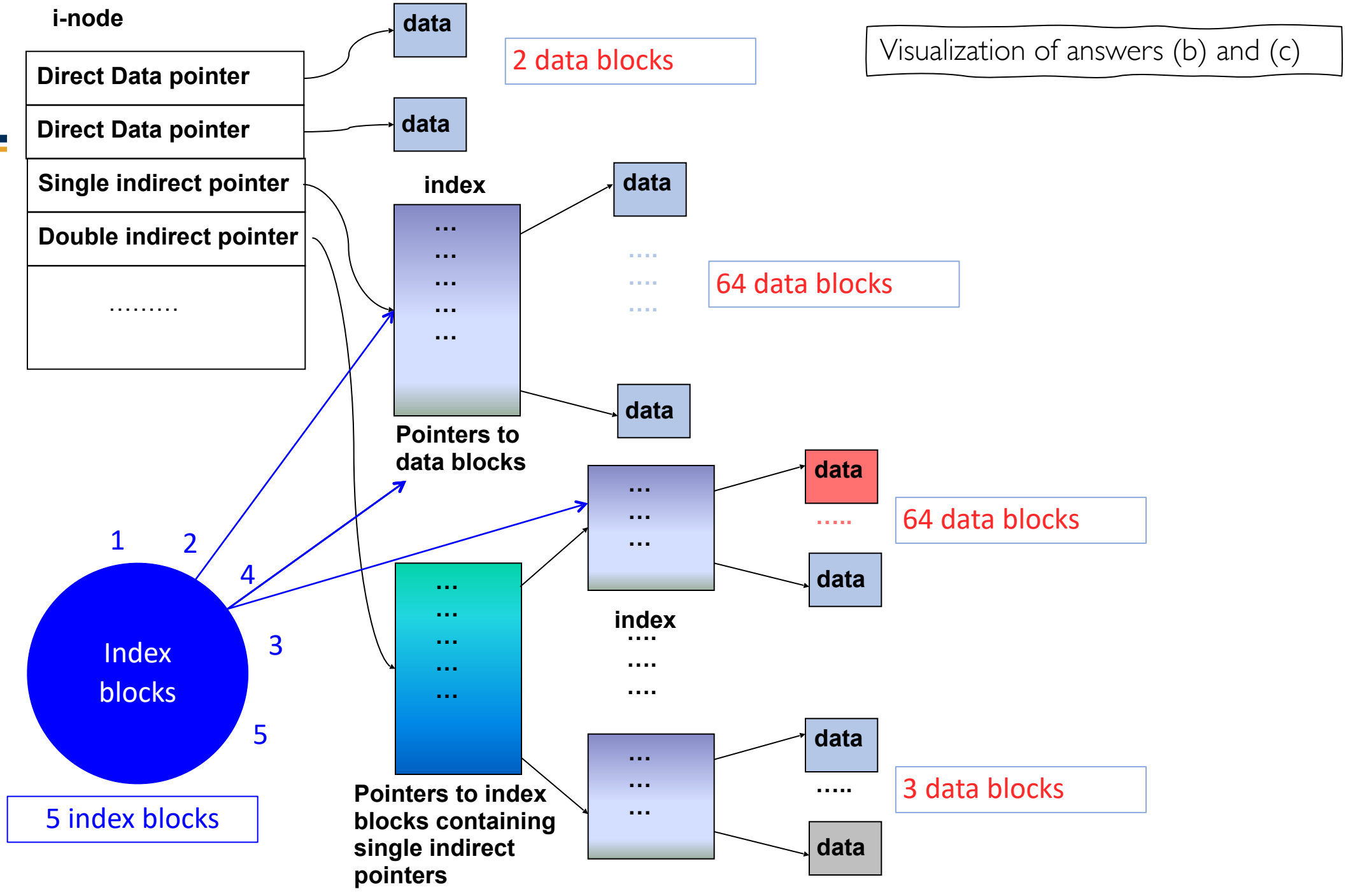

i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
Visualization of answers (b) and (c)
index
data
64 data blocks
data
Pointers to data blocks
1
Index blocks
data
64 data blocks
data
index
Pointers to index blocks containing single indirect pointers
data
3 data blocks
data


i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
Visualization of answers (b) and (c)
index
data
64 data blocks
data
Pointers to data blocks
1
2
Index blocks
data
64 data blocks
data
index
Pointers to index blocks containing single indirect pointers
data
3 data blocks
data

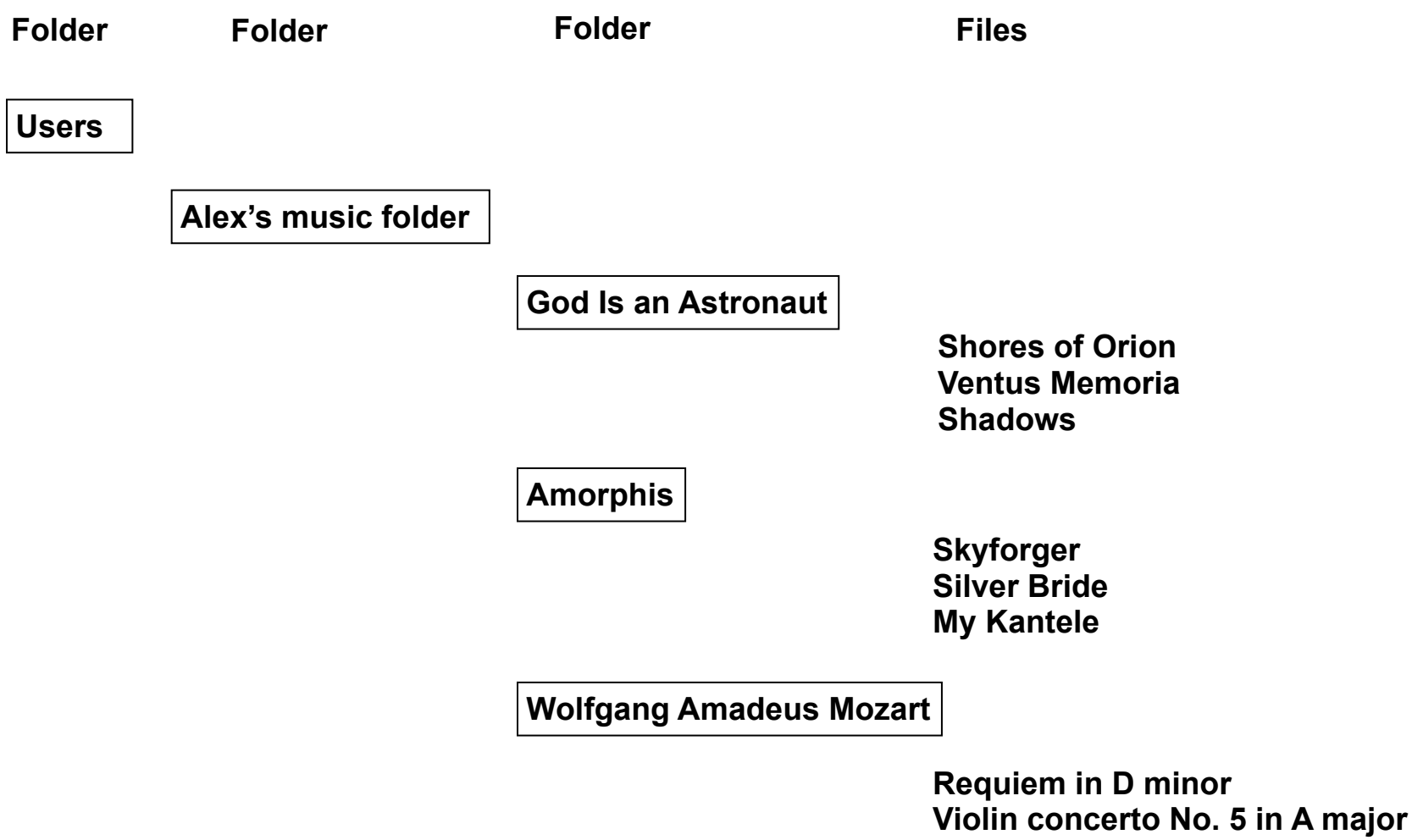

i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
index
data
64 data blocks
data
Pointers to data blocks
Visualization of answers (b) and (c)
1
2
3
Index blocks
data
64 data blocks
data
index
Pointers to index blocks containing single indirect pointers
data
3 data blocks
data


i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
Visualization of answers (b) and (c)
index
data
64 data blocks
data
Pointers to data blocks
1
2
4
3
Index blocks
data
64 data blocks
data
index
Pointers to index blocks containing single indirect pointers
data
3 data blocks
data

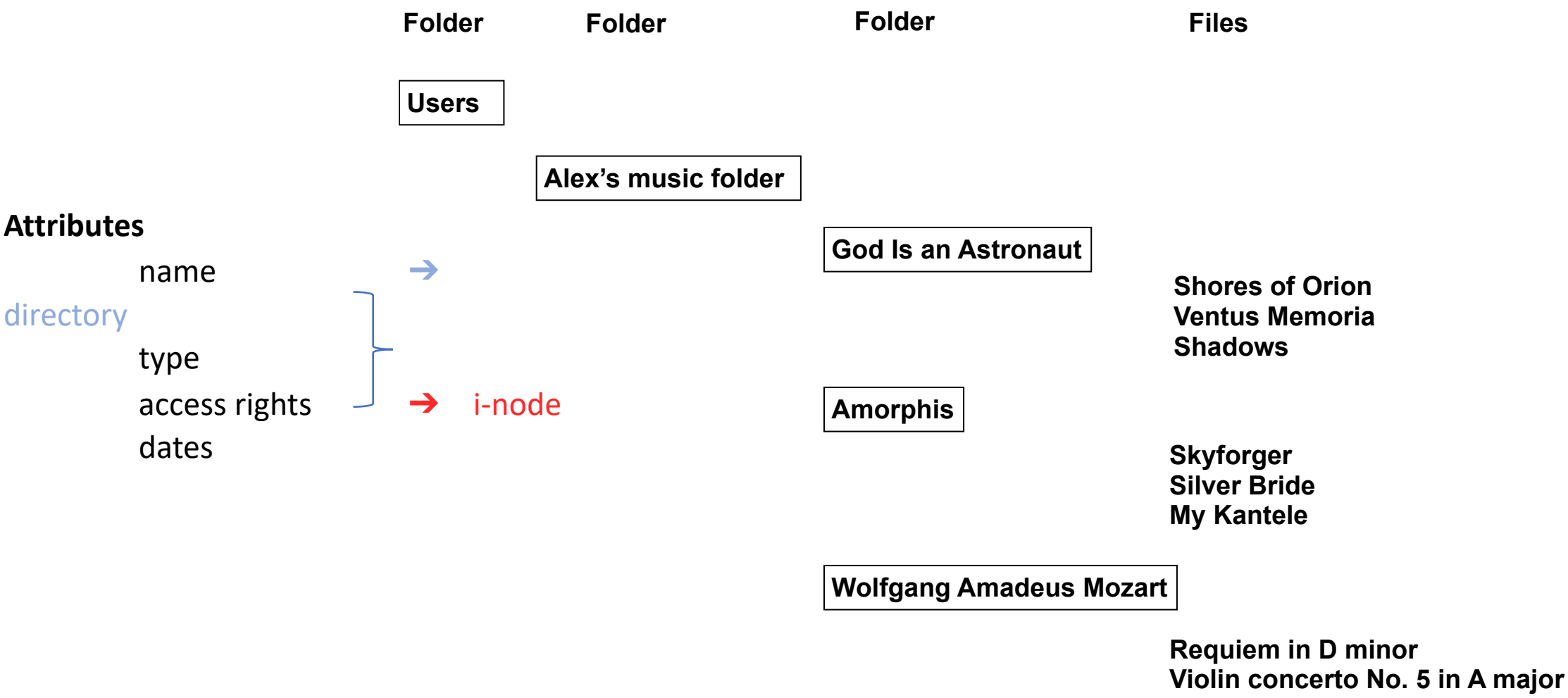

i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
index
data
64 data blocks
data
Pointers to data blocks
Visualization of answers (b) and (c)
1
2
3
4
5
Index blocks
data
64 data blocks
data
index
Pointers to index blocks containing single indirect pointers
data
3 data blocks
data


i-node
Direct Data pointer
Direct Data pointer
Single indirect pointer
Double indirect pointer
.....
data
data
2 data blocks
Visualization of answers (b) and (c)
index
data
64 data blocks
data
Pointers to data blocks
1
2
4
3
5
Index blocks
5 index blocks
data
64 data blocks
data
index
Pointers to index blocks containing single indirect pointers
data
3 data blocks
data

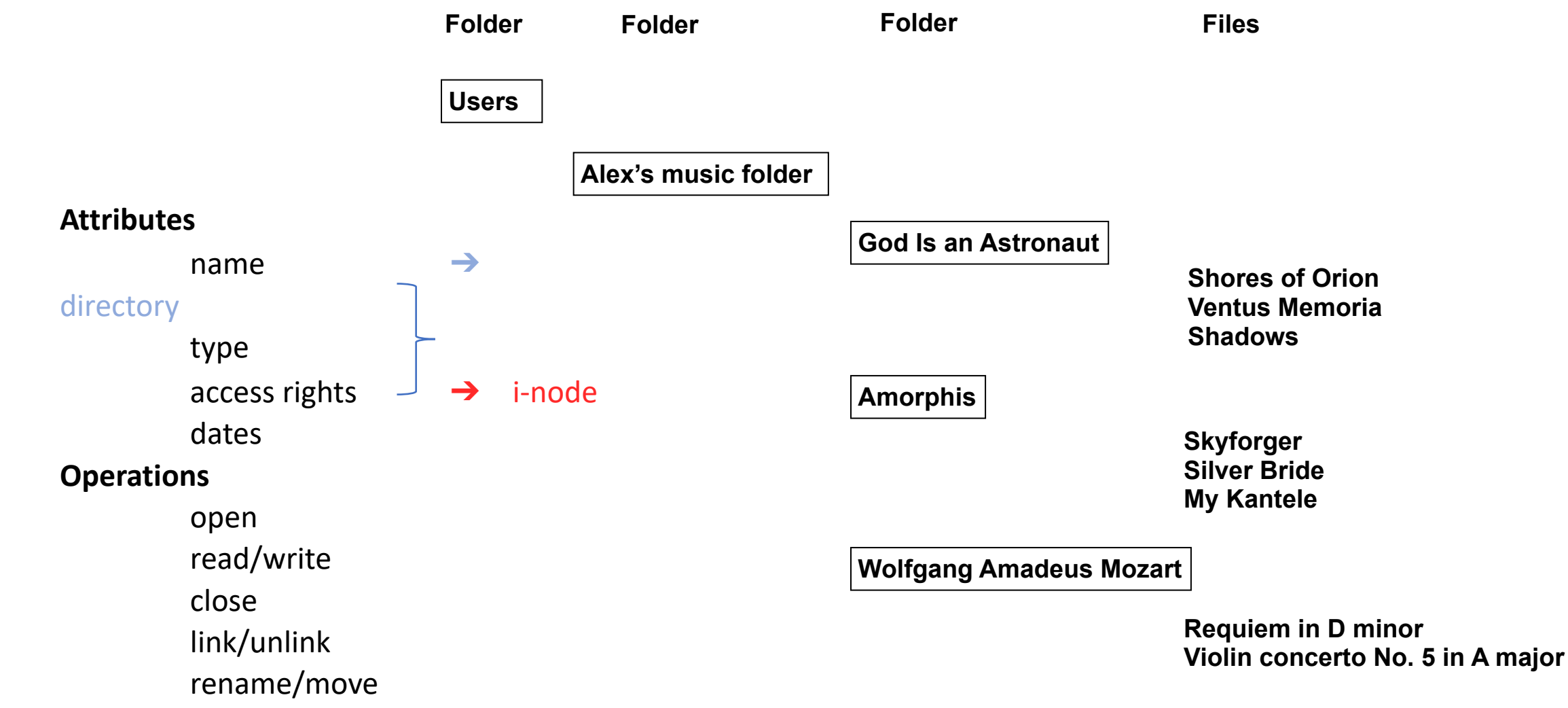
Unix file system



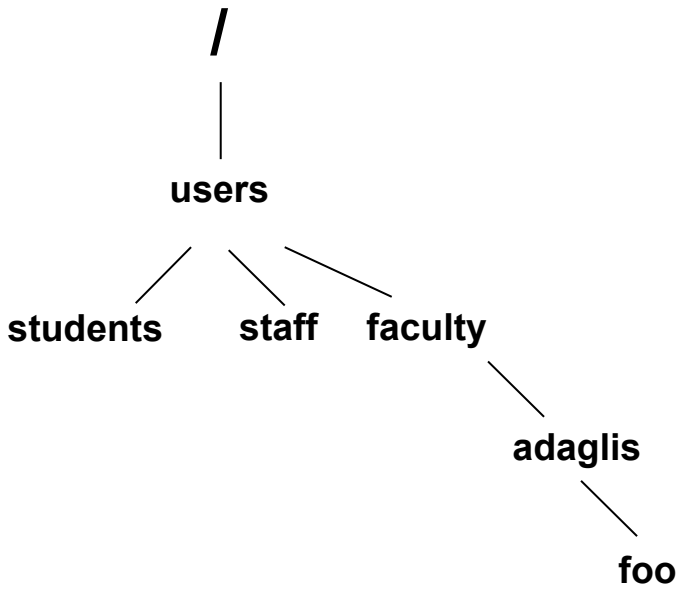
Unix file system



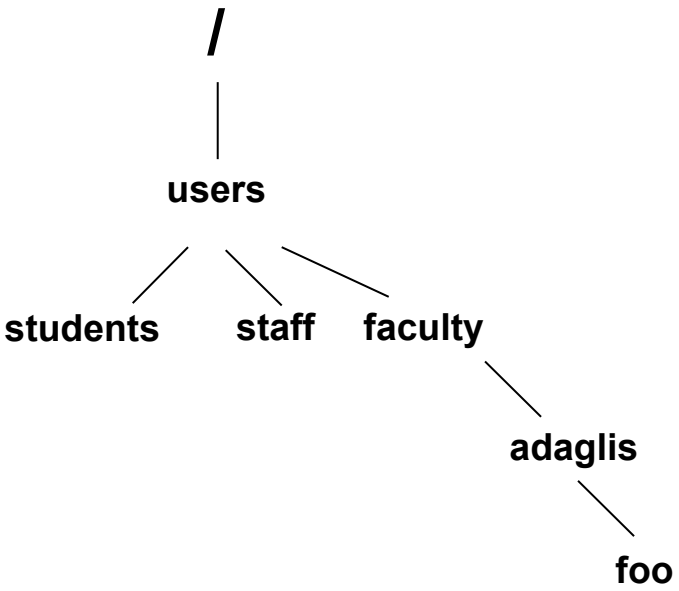
Unix file system



Unix file system



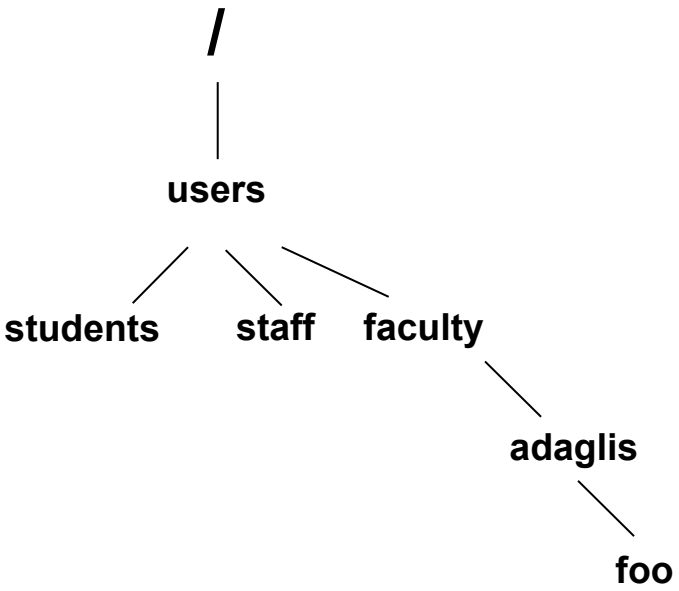
Unix file system




```
-rw-r--r--   1 adaglis  faculty 475  Apr 18  2024 foo
```

Unix file system

- Metadata in the i-node
  - Access rights: U G O
  - User name (uid)
  - Group name (gid)
  - Size
  - Modification date
  - Data/index block addrs

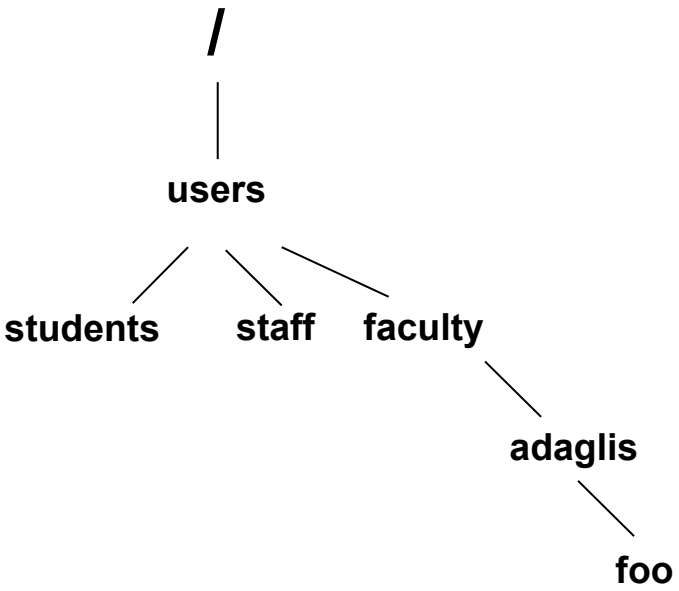


User Group Other

```
-rw-r--r--   1 adaglis  faculty 475  Apr 18  2024 foo
```

Unix file system

- Metadata in the i-node
  - Access rights: U G O
  - User name (uid)
  - Group name (gid)
  - Size
  - Modification date
  - Data/index block addrs

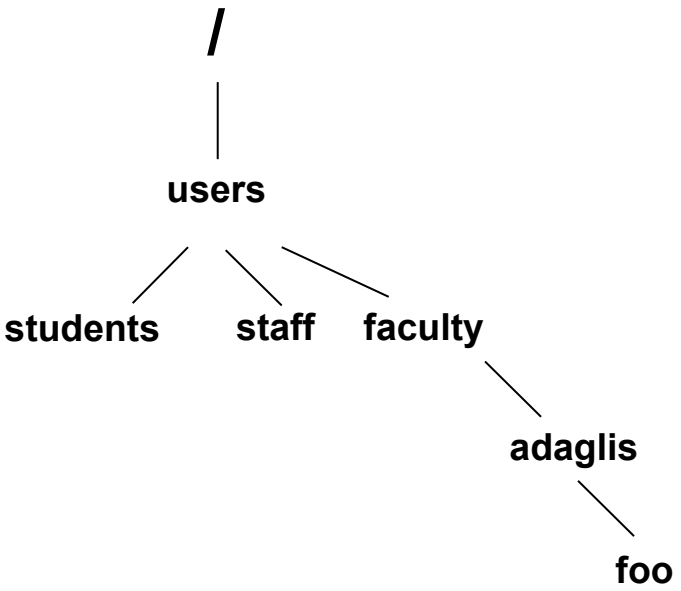


User Group Other Links

```
-rw-r--r--   1 adaglis  faculty 475  Apr 18   2024 foo
```

Unix file system

- Metadata in the i-node
  - Access rights: U G O
  - User name (uid)
  - Group name (gid)
  - Size
  - Modification date
  - Data/index block addrs



User Group Other

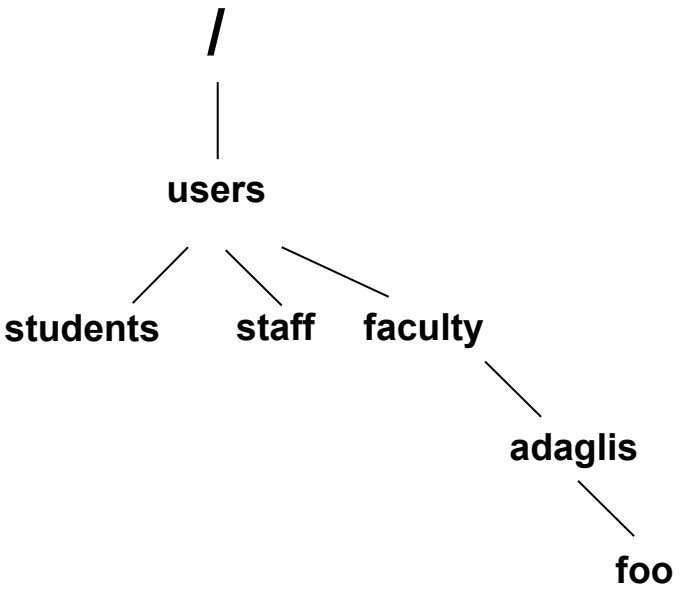
Links

Name (in directory)

```
-rw-r--r--   1 adaglis  faculty 475  Apr 18   2024 foo
```

Unix file system

- Metadata in the i-node
  - Access rights: U G O
  - User name (uid)
  - Group name (gid)
  - Size
  - Modification date
  - Data/index block addrs



"d" if directory (i.e., "folder")

User Group Other

Links

Name (in directory)

```
-rw-r--r--   1 adaglis  faculty 475  Apr 18   2024 foo
```

Links

- Remember that Unix-style file system separates the directory entry from the i-node; this means more than one directory entry can index the same i-node.
- When you create a file, you initialize an i-node and create a directory entry; this sets the link count in the i-node to 1
- You can also create additional links to the file with the link() system call or *ln* command; this increments the link count in the i-node for each link; this is called a "hard link"
- Curiously, there is no such thing as a primary or secondary link; both directory entries are names for the file and neither link has precedence over the other

Links

- Remember that Unix-style file system separates the directory entry from the i-node; this means more than one directory entry can index the same i-node.
- When you create a file, you initialize an i-node and create a directory entry; this sets the link count in the i-node to 1
- You can also create additional links to the file with the link() system call or *ln* command; this increments the link count in the i-node for each link; this is called a "hard link"
- Curiously, there is no such thing as a primary or secondary link; both directory entries are names for the file and neither link has precedence over the other
- There is no system call to "remove" a file; you can only unlink a directory entry
- The file is deleted from the file system only when its link count reaches zero; you remove a link with the unlink() system call or the *rm* command

Links

- Remember that Unix-style file system separates the directory entry from the i-node; this means more than one directory entry can index the same i-node.
- When you create a file, you initialize an i-node and create a directory entry; this sets the link count in the i-node to 1
- You can also create additional links to the file with the link() system call or *ln* command; this increments the link count in the i-node for each link; this is called a "hard link"
- Curiously, there is no such thing as a primary or secondary link; both directory entries are names for the file and neither link has precedence over the other
- There is no system call to "remove" a file; you can only unlink a directory entry
- The file is deleted from the file system only when its link count reaches zero; you remove a link with the unlink() system call or the *rm* command
- Even better, each i-node also has an in-use count which counts the number of processes that have the file open; the i-node is not deallocated until the link count AND the in-use count both reach zero, so you can have an open file that has an i-node but no directory entry and hence no name in the file system

Symbolic links

- Unix also provides a way to create file aliases; these are called symbolic links
- A symbolic link occupies an i-node (contrast with a hard link doesn't take an additional i-node)
- The i-node is marked as a symbolic link and the data blocks contain a path name instead of user data

Symbolic links

- Unix also provides a way to create file aliases; these are called symbolic links
- A symbolic link occupies an i-node (contrast with a hard link doesn't take an additional i-node)
- The i-node is marked as a symbolic link and the data blocks contain a path name instead of user data
- When a symbolic link is encountered while following a path name, the path is replaced by the contents of the symbolic link and the search continues by following the path in the symbolic link
- The file system doesn't promise that symbolic links point to anything; the presence of a symbolic link pointing to a file name doesn't prevent that file from being unlinked

Symbolic links

- Unix also provides a way to create file aliases; these are called symbolic links
- A symbolic link occupies an i-node (contrast with a hard link doesn't take an additional i-node)
- The i-node is marked as a symbolic link and the data blocks contain a path name instead of user data
- When a symbolic link is encountered while following a path name, the path is replaced by the contents of the symbolic link and the search continues by following the path in the symbolic link
- The file system doesn't promise that symbolic links point to anything; the presence of a symbolic link pointing to a file name doesn't prevent that file from being unlinked
- Symlinks work across file systems contrasted with hard links that do not

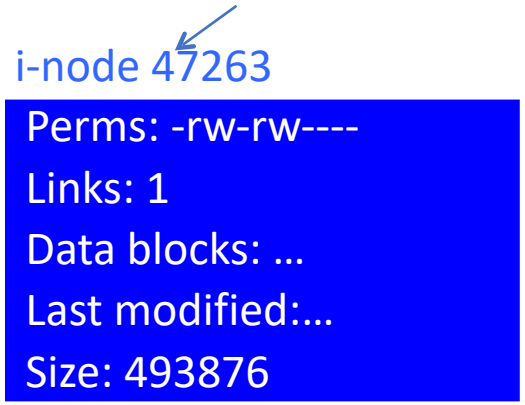
Unix file and links

/Users

/adaglis

/data

/base (i-node = 47263)



Unix file and links

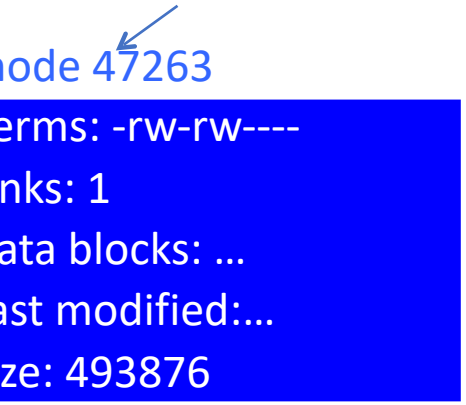
/Users

/adaglis

/data

/base (i-node = 47263)

This is a directory entry; it contains only a name and i-number



Unix file and links

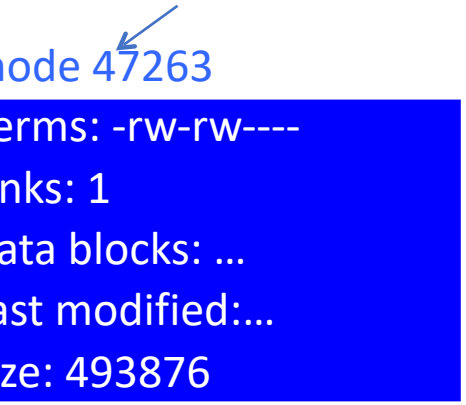
/Users

/adaglis

/data

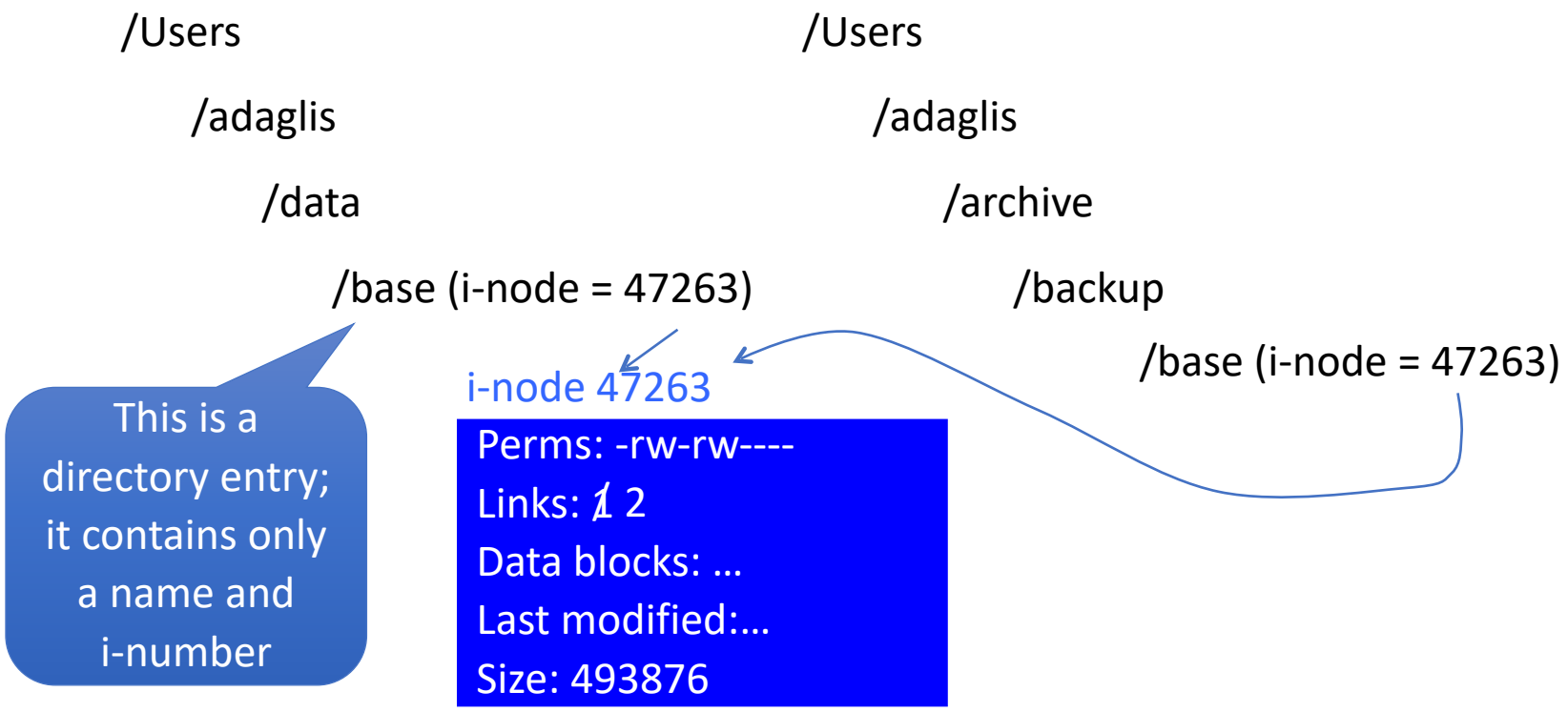
/base (i-node = 47263)

This is a directory entry; it contains only a name and i-number



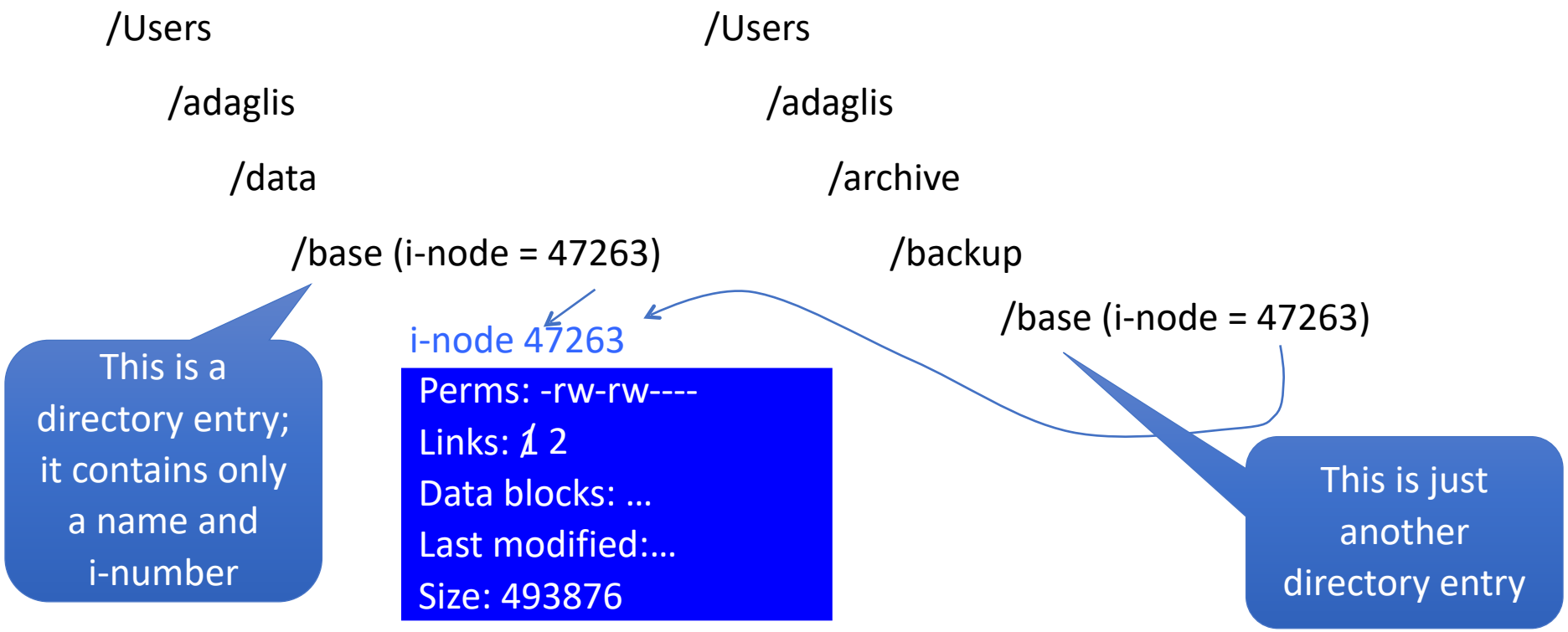

```
ln /Users/adaglis/data/base /Users/adaglis/archive/backup/base
```

Unix file and links



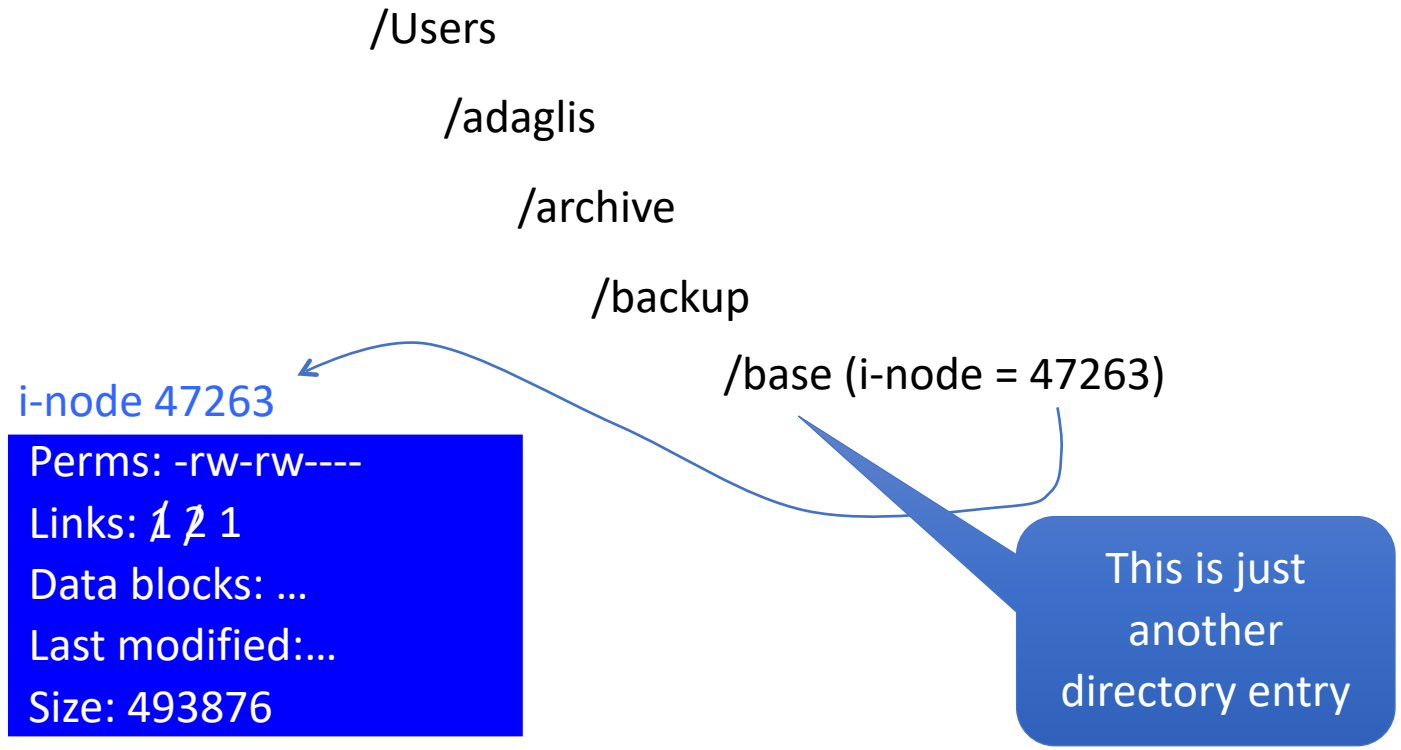

```
ln /Users/adaglis/data/base /Users/adaglis/archive/backup/base
```

Unix file and links




```
ln /Users/adaglis/data/base /Users/adaglis/archive/backup/base
```

Unix file and links




```
ln /Users/adaglis/data/base /Users/adaglis/archive/backup/base
rm /Users/adaglis/data/base # remember this calls unlink()
```

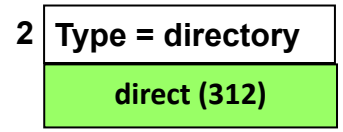
Attribute	Meaning	Elaboration
Name	Name of the file	Attribute set at the time of creation or renaming
Alias	Other names that exist for the same physical file	Attribute gets set when an alias is created; system such as Unix provide explicit commands for creating aliases for a given file; Unix supports aliasing at two different levels (physical or hard, and symbolic or soft)
Owner	Usually the user who created the file	Attribute gets set at the time of creation of a file; systems such as Unix provide mechanism for the file's ownership to be changed by the superuser
Creation time	Time when the file was created first	Attribute gets set at the time a file is created or copied from some other place
Last write time	Time when the file was last written to	Attribute gets set at the time the file is written to or copied; in most file systems the creation time attribute is the same as the last write time attribute; Note that moving a file from one location to another preserves the creation time of the file
Privileges •Read •Write •Execute	The permissions or access rights to the file specifies who can do what to the file	Attribute gets set to default values at the time of creation of the file; usually, file systems provide commands to modify the privileges by the owner of the file; modern Linux and Windows file systems such as ext4 and NTFS also provide an access control list (ACL) to give more granular access to different users
Size	Total space occupied on the file system	Attribute gets set every time the size changes due to modification to the file

Unix command	Semantics	Elaboration
touch <name>	**Create a file with the name <name>**	Creates a zero byte file with the name <name> and a creation time equal to the current wall clock time
mkdir <sub-dir>	**Create a sub-directory <sub-dir>**	The user must have write privilege to the current working directory (if <sub-dir> is a relative name) to be able to successfully execute this command
rm <name>	**Remove (or delete) the file named <name>**	Only the owner of the file (and/or superuser) can delete a file
rmdir <sub-dir>	**Remove (or delete) the sub-directory named <sub-dir>**	Only the owner of the <sub-dir> (and/or the superuser) can remove the named sub-directory
ln –s <orig> <new>	**Create a name <new> and make it symbolically equivalent to the file <orig>**	This is name equivalence only; so if the file <orig> is deleted, the storage associated with <orig> is reclaimed, and hence <new> will be a dangling reference to a non-existent file
ln <orig> <new>	**Create a name <new> and make it physically equivalent to the file <orig>**	Even if the file <orig> is deleted, the physical file remains accessible via the name <new>
chmod <rights> <name>	**Change the access rights for the file <name> as specified in the mask <rights>**	Only the owner of the file (and/or the superuser) can change the access rights
chown <user> <name>	**Change the owner of the file <name> to be <user>**	Only superuser can change the ownership of a file
chgrp <group> <name>	**Change the group associated with the file <name> to be <group>**	Only the owner of the file (and/or the superuser) can change the group associated with a file
cp <orig> <new>	**Create a new file <new> that is a copy of the file <orig>**	The copy is created in the same directory if <new> is a file name; if <new> is a directory name, then a copy with the same name <orig> is created in the directory <new>
mv <orig> <new>	**Renames the file <orig> with the name <new>**	Renaming happens in the same directory if <new> is a file name; if <new> is a directory name, then the file <orig> is moved into the directory <new> preserving its name <orig>
cat/more/less <name>	**View the file contents**	

Directory Structure

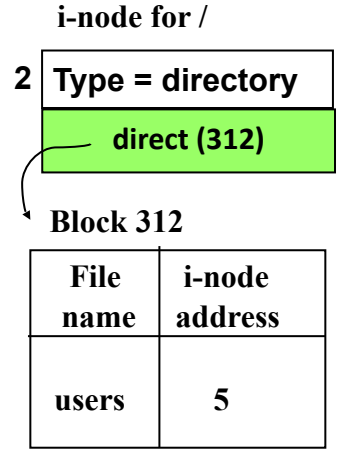
Let's study the path /users/faculty/adaglis/foo

i-node for /



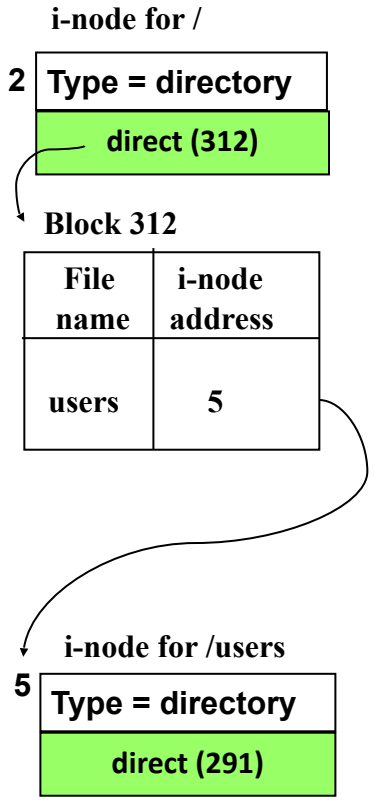
Directory Structure

Let's study the path /users/faculty/adaglis/foo



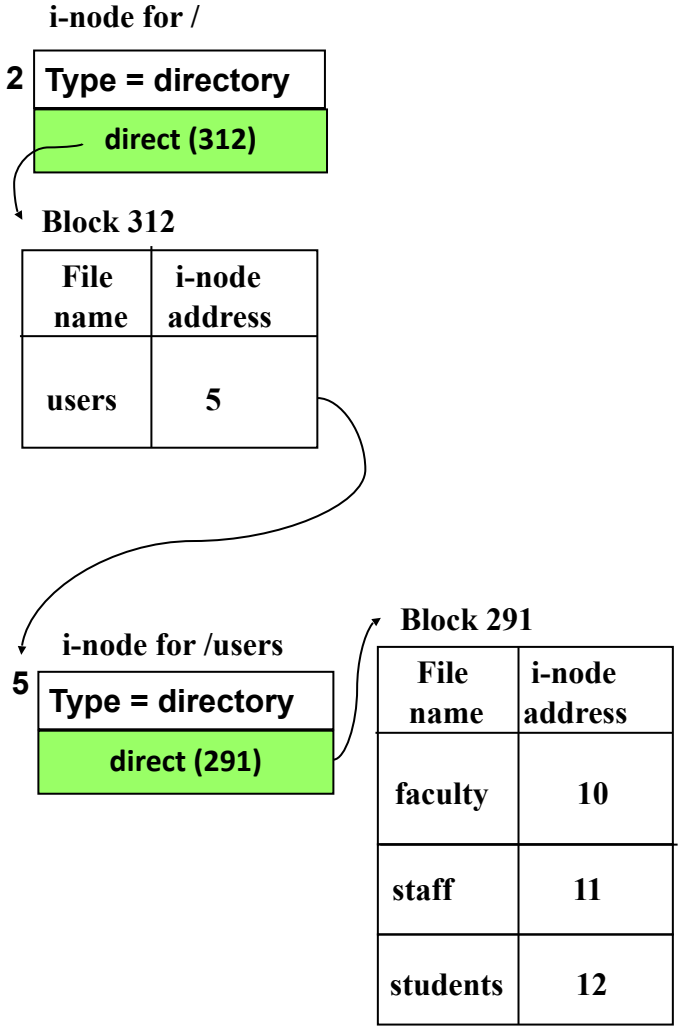
Directory Structure

Let's study the path /users/faculty/adaglis/foo



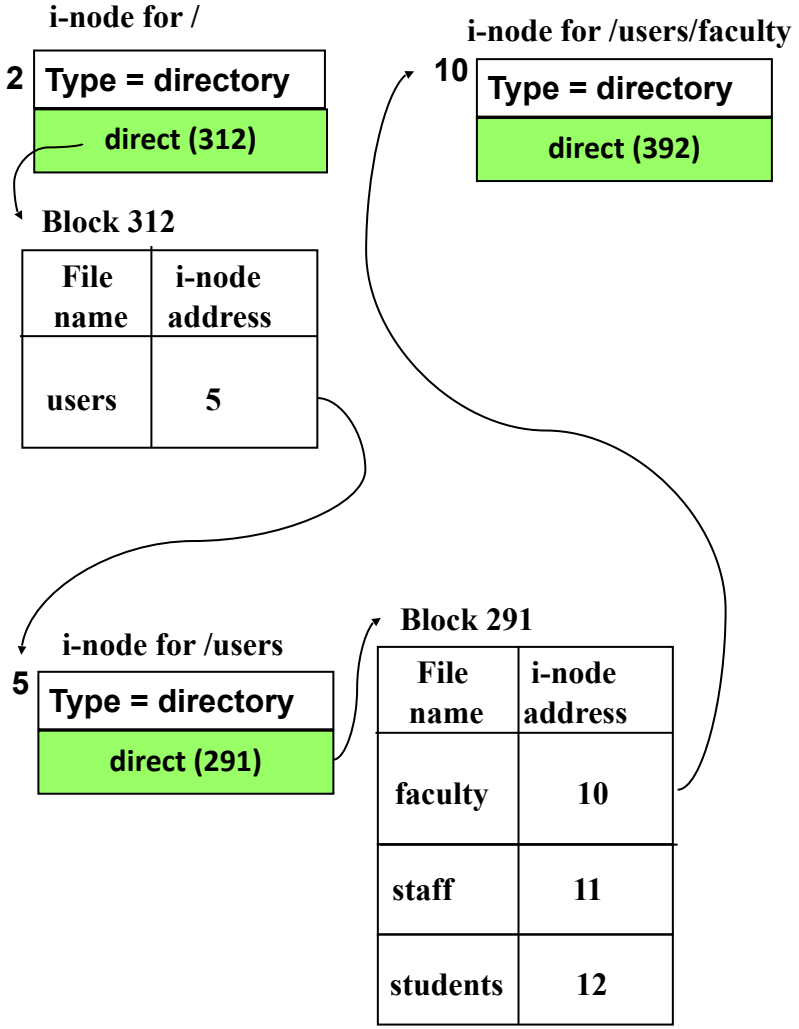
Directory Structure

Let's study the path /users/faculty/adaglis/foo



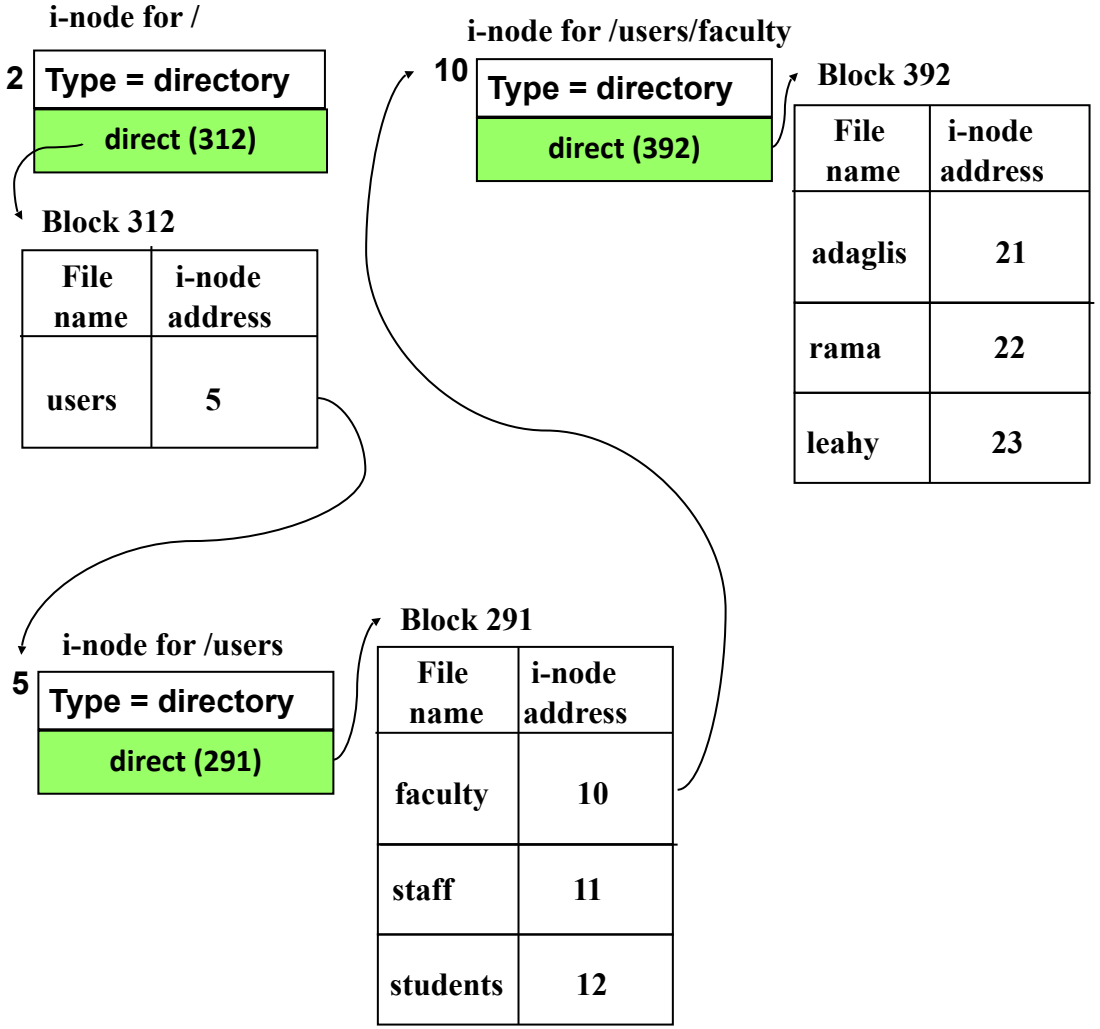
Directory Structure

Let's study the path /users/faculty/adaglis/foo



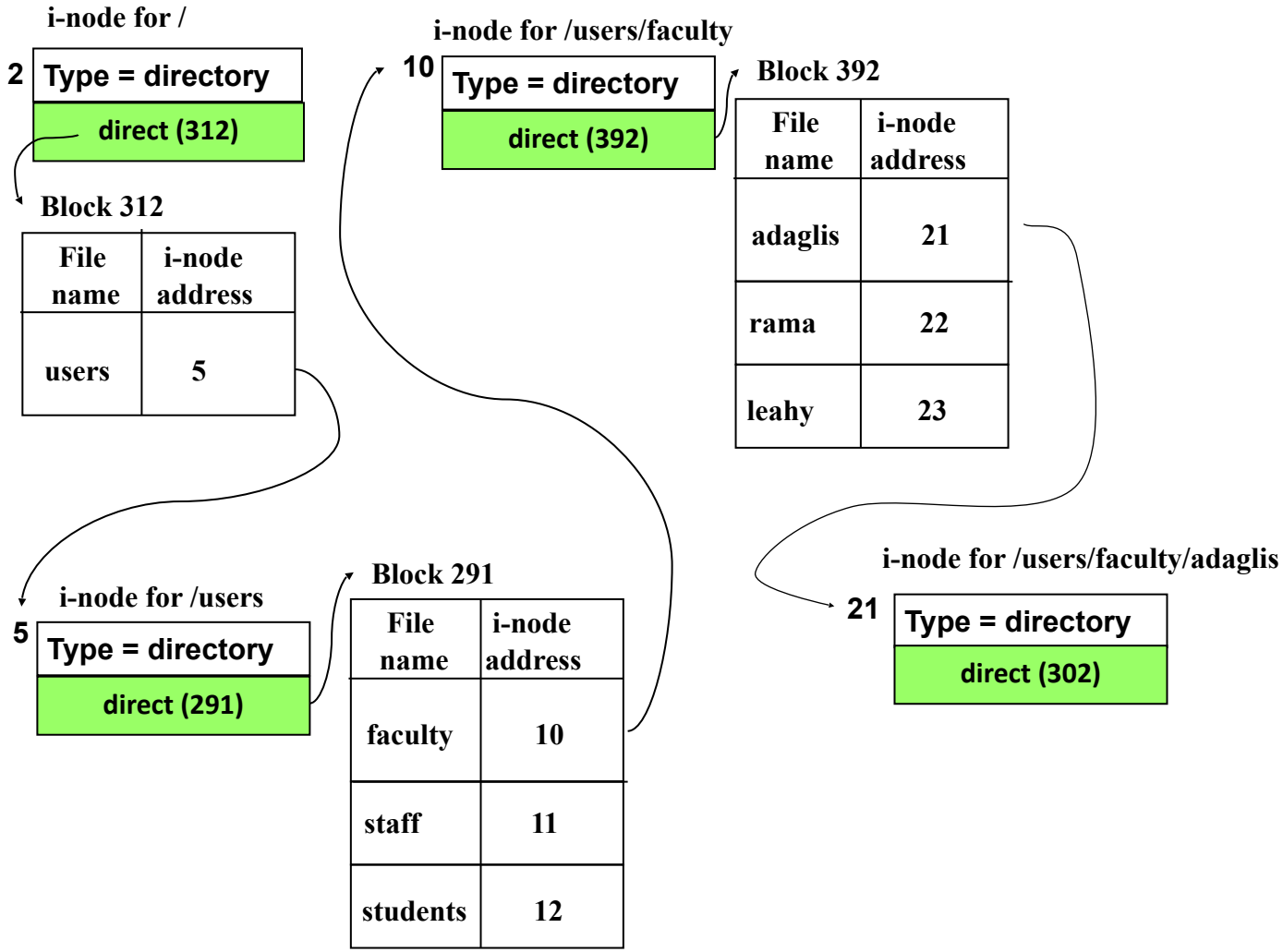
Directory Structure

Let's study the path /users/faculty/adaglis/foo



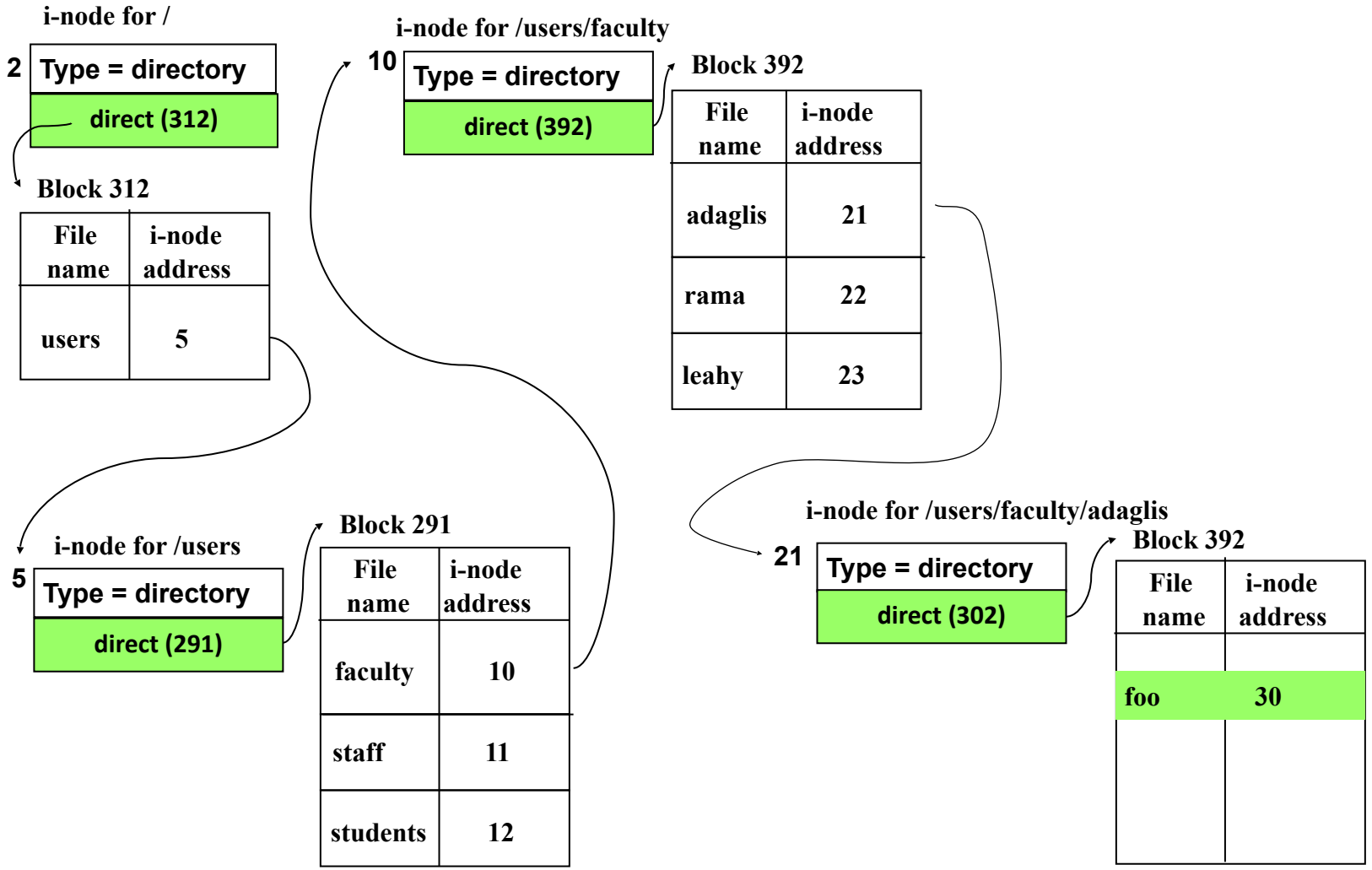
Directory Structure

Let's study the path /users/faculty/adaglis/foo



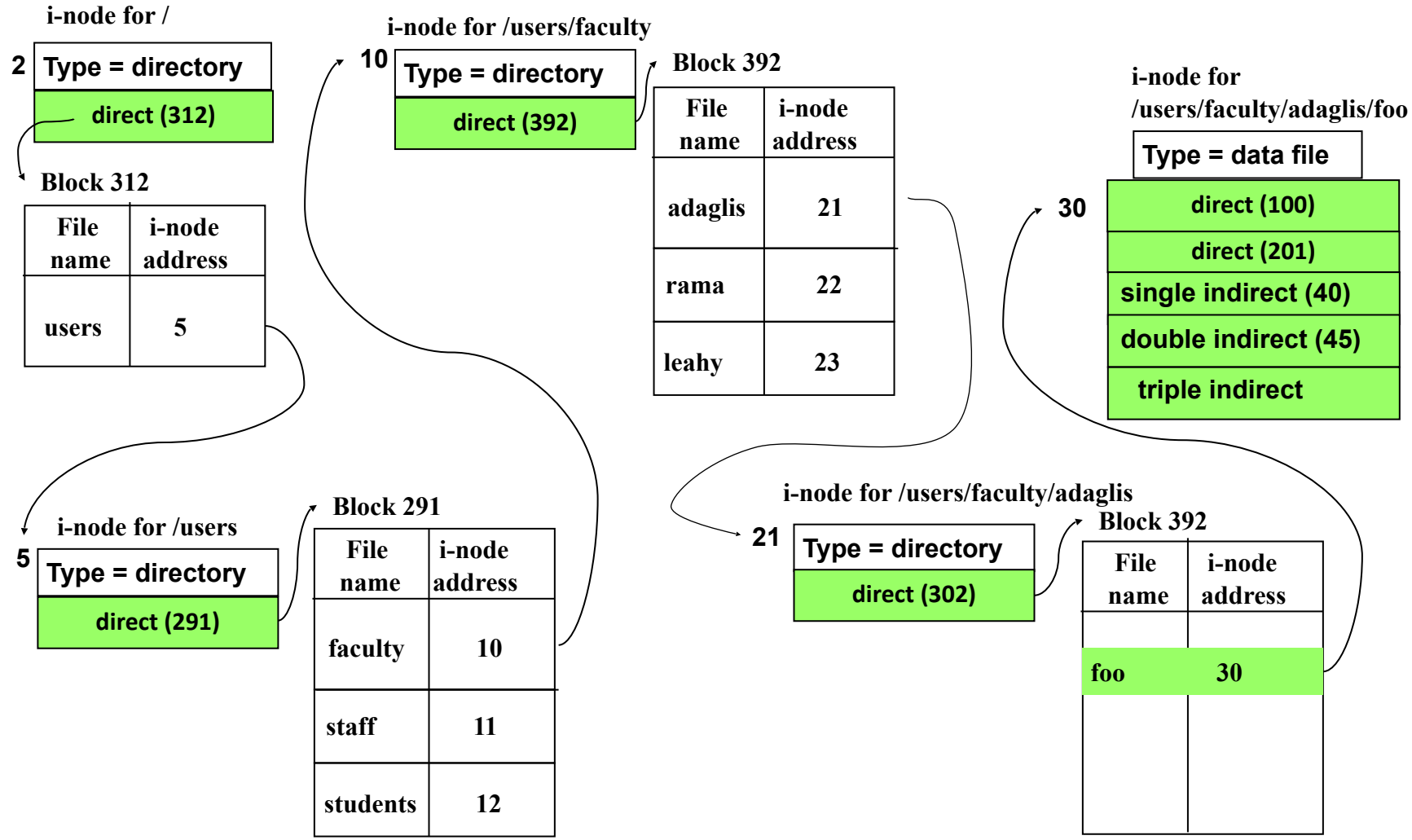
Directory Structure

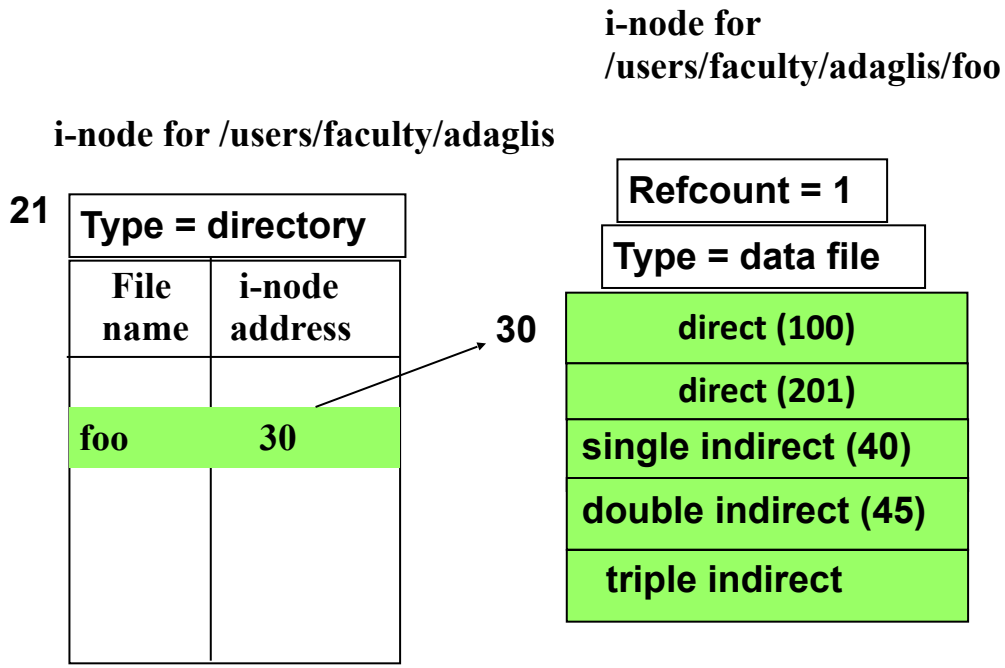
Let's study the path /users/faculty/adaglis/foo



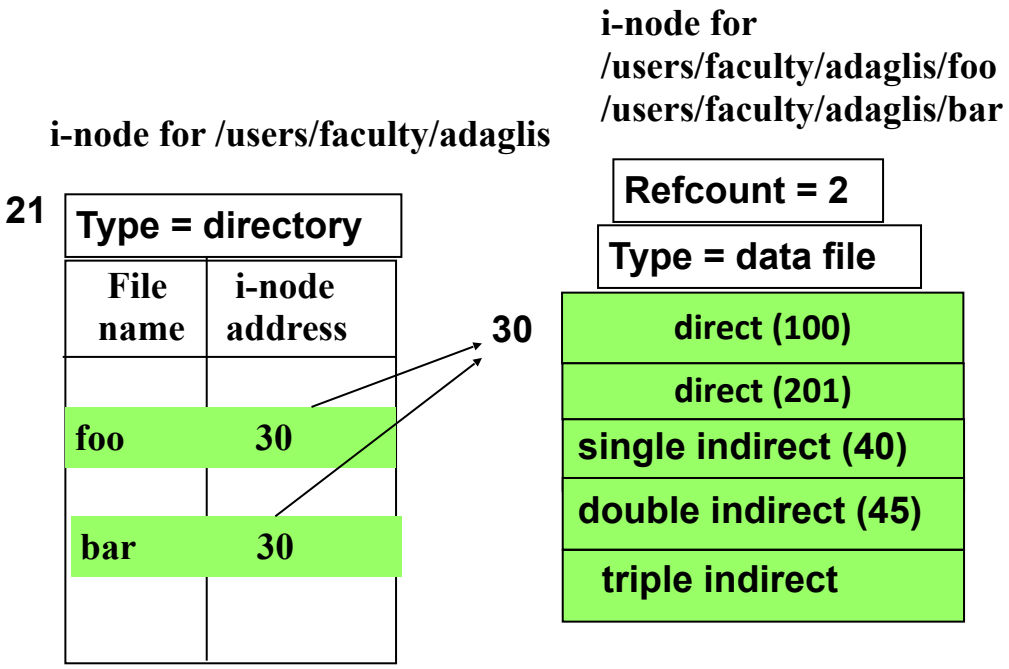
Directory Structure

Let's study the path /users/faculty/adaglis/foo



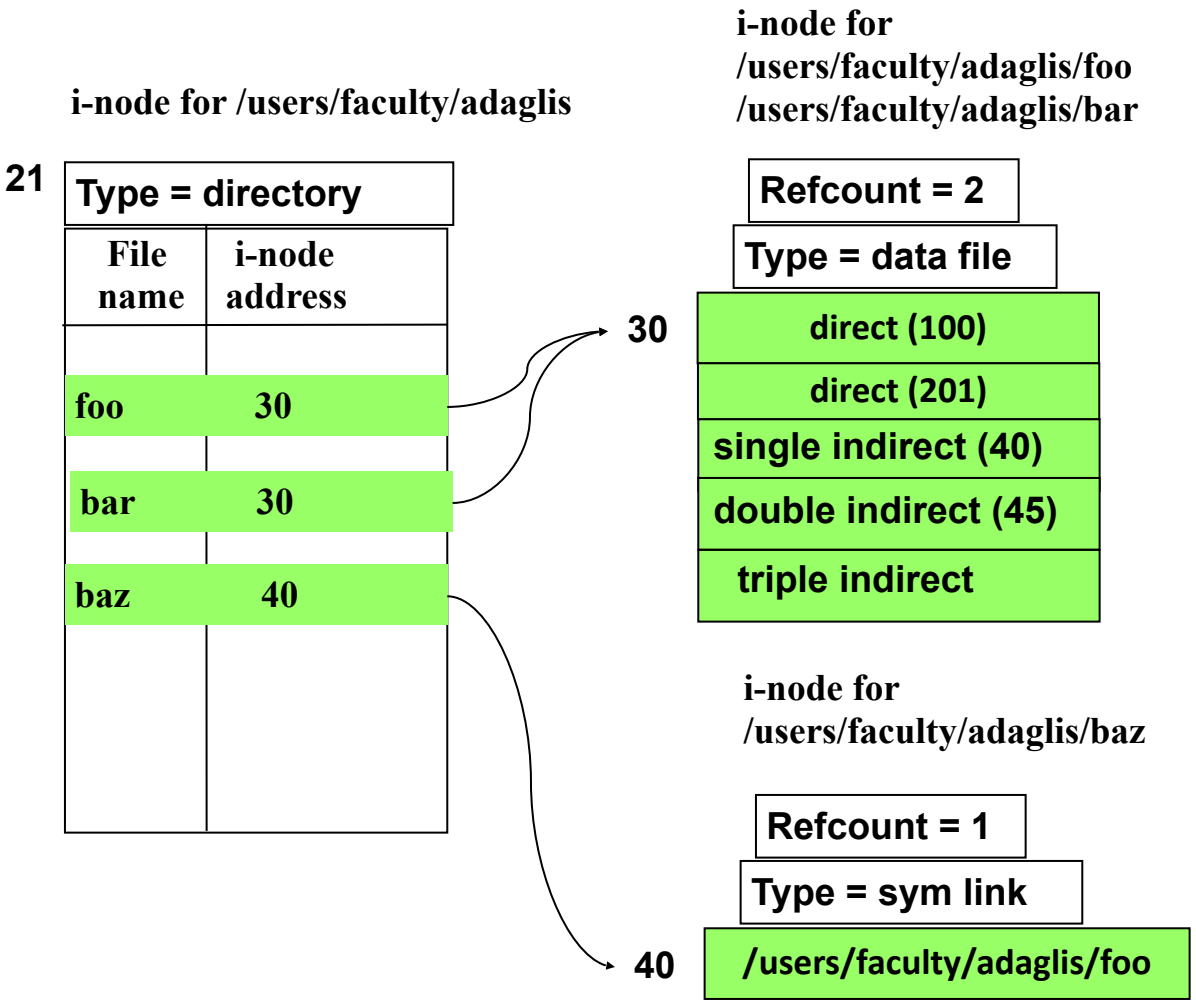


Reference count on files



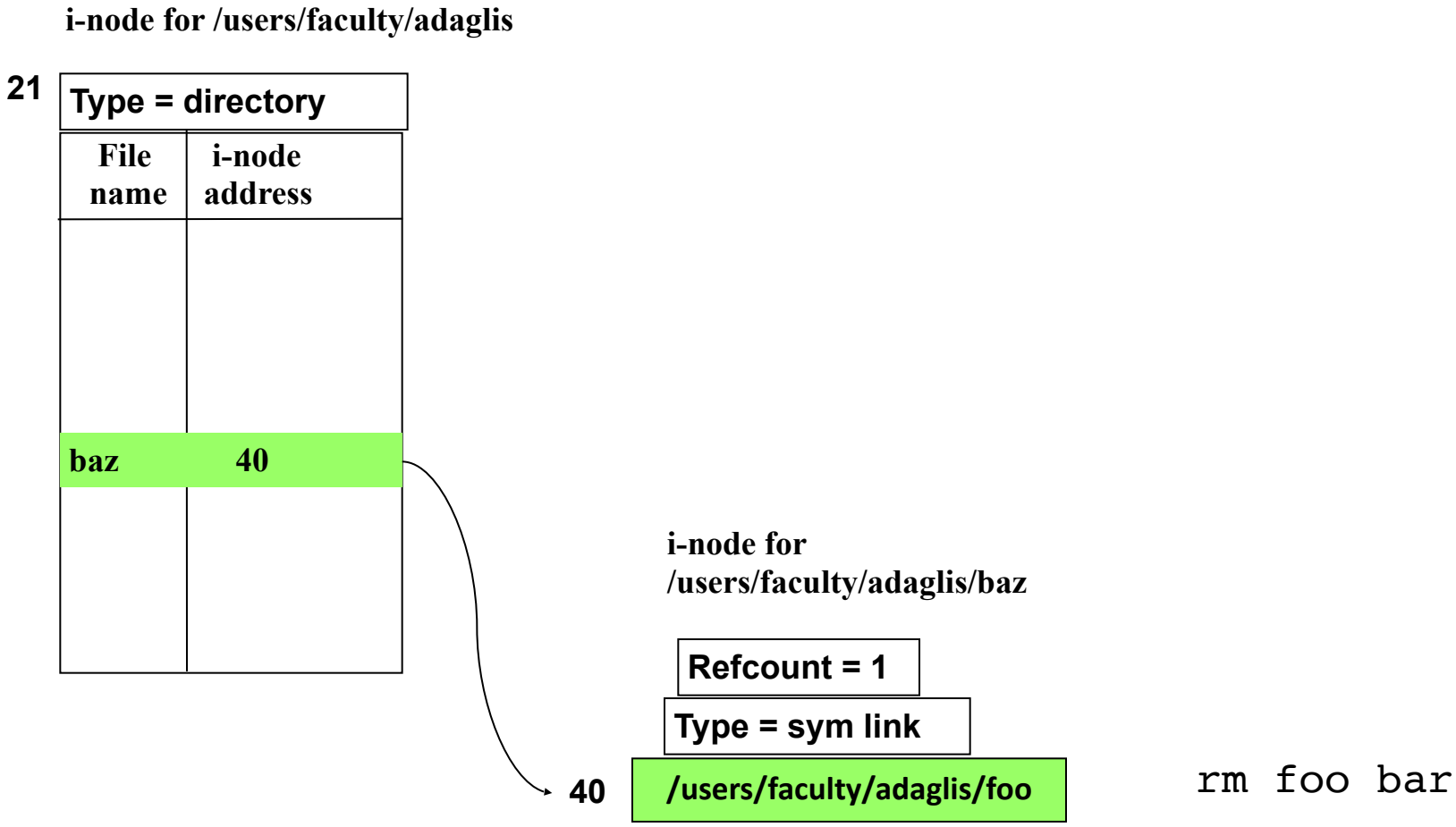
Hard links

```
ln foo bar
```



Sym links `ln -s foo baz`

After deleting foo and bar




```
rm foo bar
```

Examples in book

Work out examples for i-node creation and deletion in book on your Own:

Examples 5, 6, and 7