Implementazione delle directory

- La scelta di come implementare le directory è un punto critico nella progettazione di un File System
- Fra le varie tecniche:
  - **lista lineare (alla Unix)**: una directory è una sequenza di coppie <nome_file, inode number>
  - **B-tree**
  - **tabella hash**

Sequenza lineare

Di facile realizzazione

Limiti

- per verificare se un file è contenuto nella directory occorre scorrere il contenuto della directory
- ricerca di tipo lineare (lenta)
- è difficile mantenere ordinata la lista senza appesantire la gestione delle directory
- occorre avvalersi di strutture d'appoggio e implementare particolari algoritmi per gestire i buchi che si creano nelle directory con la
- cancellazione di file

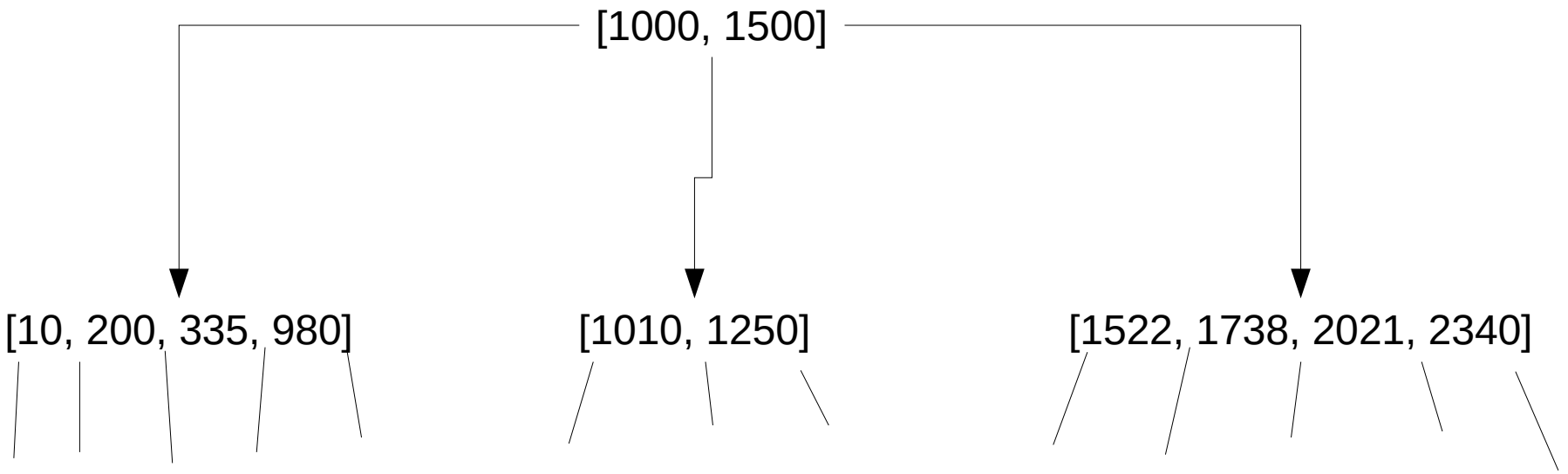
Possibili migliorie?

- appoggiarsi a strutture non lineari, in particolare alberi

B-Tree

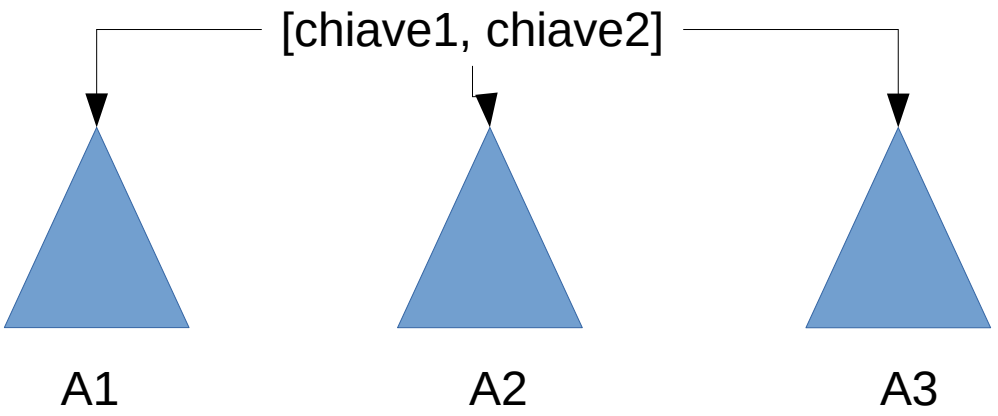
- un B-tree è una struttura ad albero ordinato.
- Ogni nodo contiene da **N** a **2N** elementi detti **chiavi**.
- Se un nodo contiene **K** elementi, allora avrà **K+1 puntatori** a nodi del livello successivo.
- Il numero N è detto **ordine dell'albero.** Per esempio il successivo è un albero di ordine 2 ...

B-Tree di ordine 2: esempio



Se i numeri fossero identificatori di inode con associati i relativi metadati questo albero potrebbe rappresentare i contenuti di una directory

In generale



Tutte le chiavi del sottoalbero A1 sono minori di chiave1

Tutte le chiavi del sottoalbero A2 sono comprese fra chiave1 e chiave2

Tutte le chiavi di A3 sono maggiori di chiave2

In un FS le chiavi possono essere identificatori di inode; ad ogni identificatore è associato anche l'inode relativo

B-tree

- La ricerca comincia sempre dalla radice del B-tree
- l'ordinamento dell'albero consente di trovare qualsiasi elemento in tempi rapidissimi per es.:

  in un B-tree di ordine 25 con 10.000.000 di nodi possiamo individuare qualsiasi chiave attraversando al più 4 nodi, se l'albero è bilanciato

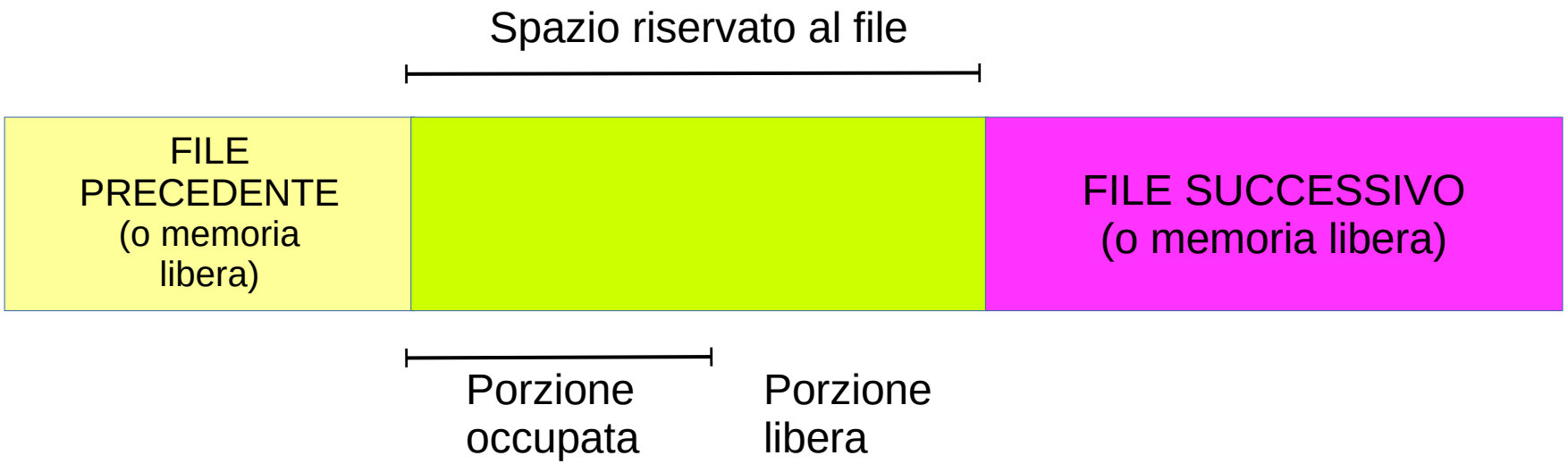
- Bilanciamento (#nodi sottoalberi sinistri = #nodi sottoalberi destri) e fan-out (apertura dell'albero) sono due caratteristiche fondamentali delle strutture ad albero usate per la ricerca

Allocazione dello spazio disco ai file

- Com’è organizzata la memorizzazione dei dati su disco?
  - allocazione contigua
  - allocazione concatenata (variante: FAT)
  - allocazione indicizzata

Allocazione CONTIGUA

Ogni file è allocato in una **sequenza contigua di blocchi**



Allocazione CONTIGUA

Ogni file è allocato in una **sequenza contigua di blocchi**

Vantaggi

- **rapidità di accesso al file**: il tempo di seek (posizionamenti della testina sulla traccia giusta) è trascurabile.
- Quando si accede a un file si tiene traccia dell'ultimo blocco letto, quindi se abbiamo appena letto il blocco B, sia l'**accesso sequenziale** (al blocco B+1) sia l'**accesso diretto** (al blocco B+k) sono immediati

Allocazione CONTIGUA

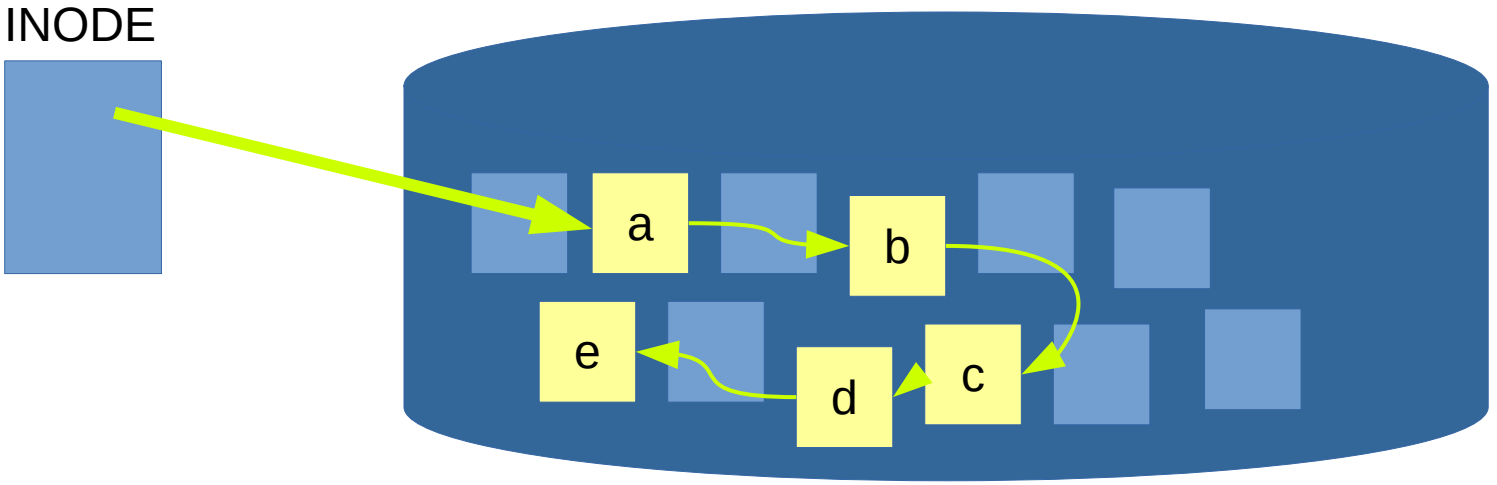
Ogni file è allocato in una **sequenza contigua di blocchi**

Svantaggi

- quanta memoria riservo per un file?
- se ne riservo troppa e il file cresce lentamente spreco memoria
- se ne riservo troppo poca? Necessità di estensioni!
- gestione dei buchi di memoria libera (best-, first-, worst-fit)
- frammentazione esterna
- necessità di deframmentare il disco di tanto in tanto

Allocazione concatenata

- Alternativa: spezzare il file in parti che possono essere allocate in modo non contiguo (blocchi dati). Occorre della sovrastruttura per mantenere l'informazione che certi blocchi dati sparsi sul disco costituiscono le parti di uno stesso file
- L'allocazione concatenata consente di fare proprio questo un file è costituito da una sequenza di blocchi sparsi per il disco ma mantenuti in una lista concatenata



Allocazione concatenata

Vantaggi

- non è necessario preallocare memoria per i file
- la lista concatenata è dinamica per natura
- non è necessario effettuare alcuna deframmentazione

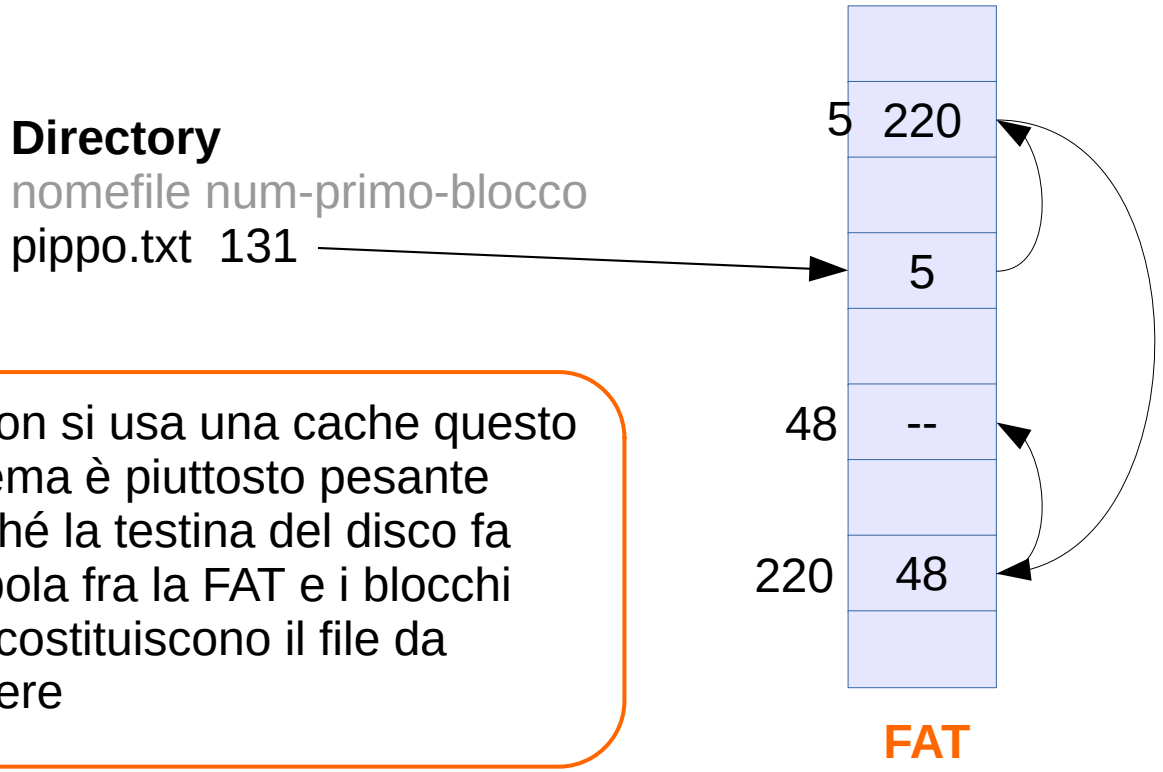
Allocazione concatenata

Svantaggi

- solo l'accesso sequenziale è efficiente, accesso diretto e indicizzato richiedono di scorrere la lista dei blocchi comunque
- i puntatori ai blocchi sono sparsi per il disco ogni salto di blocco comporta un tempo di latenza (tempo sprecato)
- occorre spazio per mantenere i puntatori ai blocchi successivi
- se un puntatore si corrompe si perde tutta la porzione successiva di file (!!!)

Allocazione concatenata: File Allocation Table (FAT)

- usata nei sistemi MS-DOS e successori
- si riserva una sezione della partizione per mantenere una tabella che ha tanti elementi quanti blocchi. Se un blocco fa parte di un file, il contenuto della entry corrispondente in tabella è un riferimento al blocco successivo:



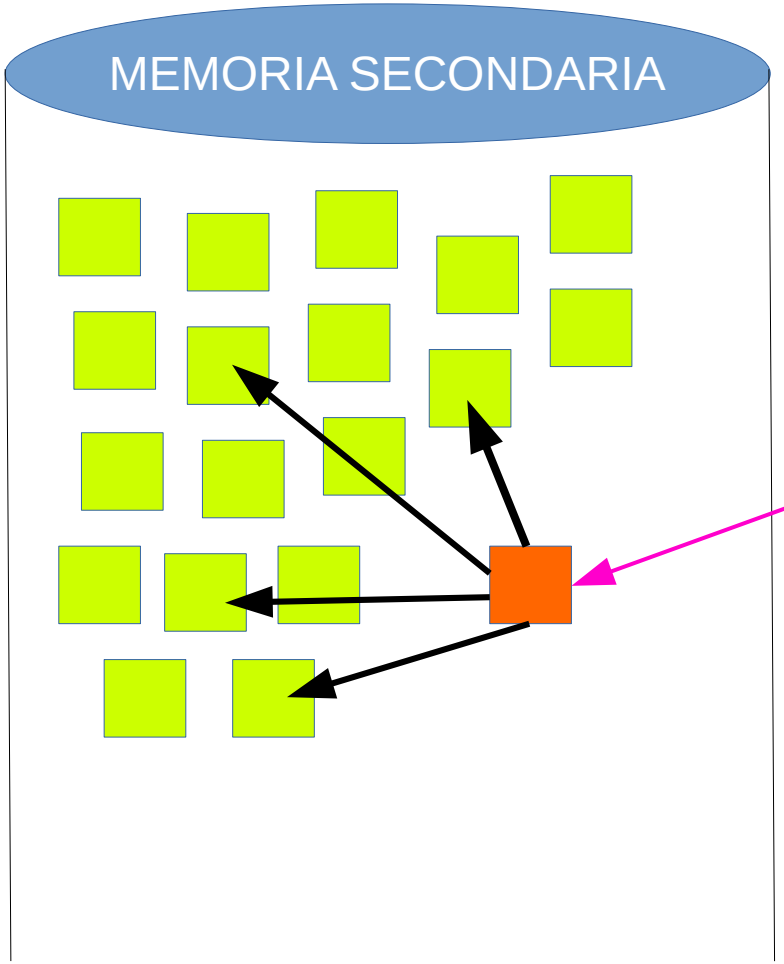
se non si usa una cache questo schema è piuttosto pesante perché la testina del disco fa la spola fra la FAT e i blocchi che costituiscono il file da leggere

Una FAT mantiene la struttura a lista concatenata esternamente ai blocchi dei file veri e propri

Allocazione indicizzata

- Tipo di allocazione che tenta di superare i limiti delle soluzioni precedenti introducendo un blocco indice
- Ogni file ha un **blocco indice**: **un array contenente gli indirizzi dei blocchi che costituiscono il file**
- I riferimenti ai blocchi indice dei file sono mantenuti nelle **directory**
- **Accesso**:
  - tramite la directory recupero il blocco indice
  - tramite i riferimenti contenuti nel blocco indice posso accedere ai vari blocchi dati

Blocco indice



Directory

File1 bloccoindice1
File2 bloccoindice2
...

Quando il file viene creato, tutti gli elementi del suo blocco indice sono NULL

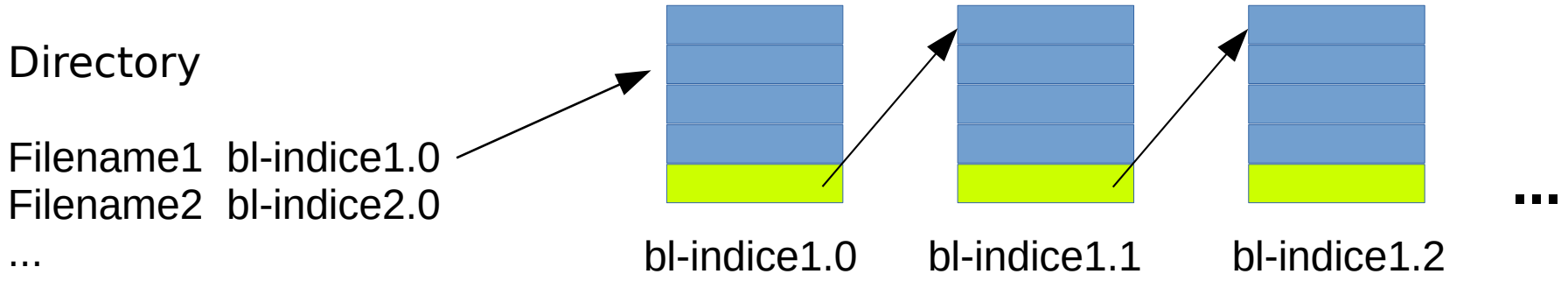
Man mano che il file cresce di dimensioni vengono allocati nuovi blocchi e i loro riferimenti sono inseriti in ordine nel blocco indice

Blocco indice

- La soluzione a blocco indice **elimina il problema della frammentazione esterna** in modo del tutto analogo all'organizzazione della RAM in pagine di pari dimensione, allocate quando c'è bisogno
- Si può avere **frammentazione interna** però solo a livello dell'ultima pagina che costituisce il file
- L'unico problema è il **dimensionamento del blocco indice**:
  - se è troppo piccolo il file potrebbe richiedere più blocchi dati quanti è possibile riferire
  - se è troppo grande si ha frammentazione interna al blocco indice stesso (spreco di memoria)
  - **soluzione ideale**: poter ridimensionare il blocco indice a seconda delle circostanze ...

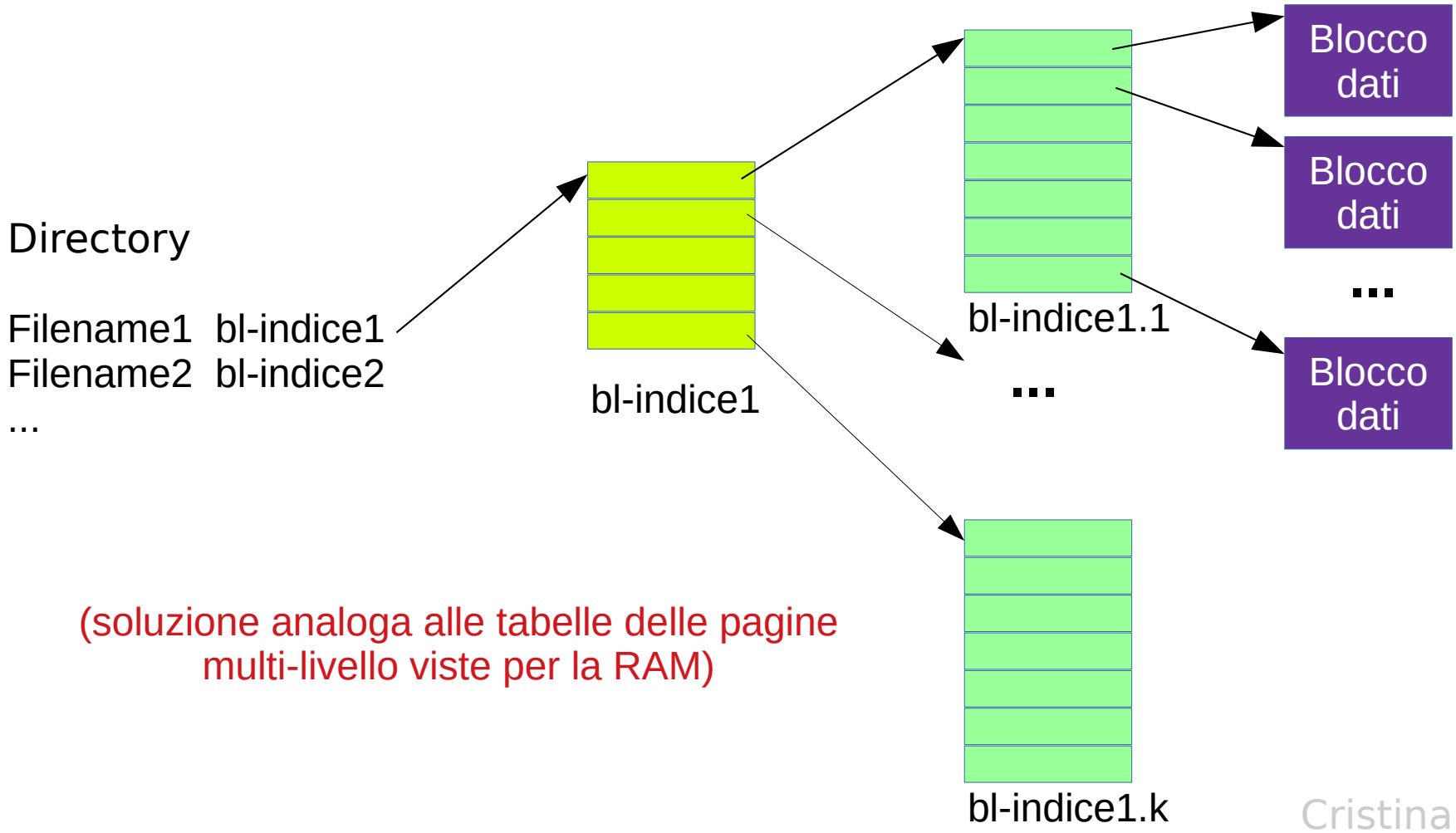
Implementazione a schema concatenato

Utilizzare blocchi indice piuttosto piccoli e interpretare l'ultimo indirizzo contenuto nel blocco indice come riferimento a un'estensione del medesimo, da usare solo se serve



Implementazione con indice a più livelli

Struttura gerarchica. Il primo livello punta a una serie di blocchi indice, ciascuno dei quali consente l'accesso a blocchi dati. I blocchi indice/dati vengono aggiunti solo se serve

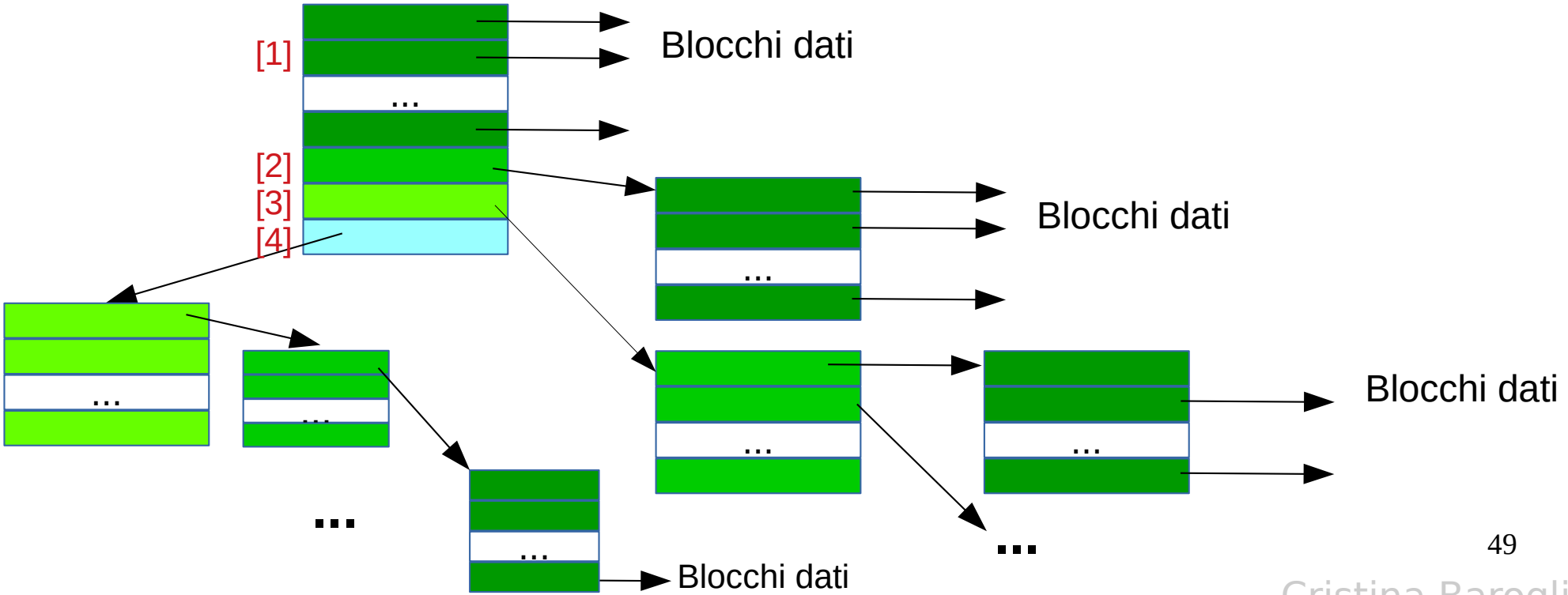


(soluzione analoga alle tabelle delle pagine multi-livello viste per la RAM)

Soluzione ibrida

Viene mantenuta una tabella di accesso ai dati, una parte dei cui elementi consente l'accesso diretto a blocchi mentre un'altra parte consente l'accesso attraverso un indice a più livelli

Soluzione adottata in Unix (per esempio), dove la tabella è inclusa nell'inode



Soluzione ibrida

[1] ogni entry è un riferimento a un blocco dati

[2] ogni entry è un riferimento a una tabella di riferimenti a blocchi dati

[3] ogni entry è un riferimento a una tabella di riferimenti a una tabella di riferimenti a blocchi dati

[4] ogni entry è un riferimento a una tabella di riferimenti a una tabella di riferimenti a una tabella di riferimenti a blocchi dati

si hanno da 15 a 19 blocchi a indirzzamento diretto [2], [3] e [4] sono allocati solo se serve

Soluzione ibrida

Vantaggio

- con questa tecnica si possono costruire file che raggiungono dimensioni molto grandi (terabyte)

Svantaggio

- laddove si usino i livelli di indicizzazione indiretta questa tecnica soffre un po' dei limiti della tecnica di concatenazione

Commenti generali

- **Allocazione concatenata**: più adeguata a accessi sequenziali
- **Allocazione a indice**: più adeguata ad accessi diretti
- L'allocazione indicizzata richiede di mantenere in RAM una parte dei blocchi indice, possono occorre due o più accessi al disco se la RAM non è sufficiente:
  - uno (o più) per accedere al blocco indice giusto
  - uno per raggiungere il dato di interesse
- Alcuni sistemi combinano allocazione contigua e indicizzata: finché il file rimane di piccole dimensioni si usa l'allocazione contigua, oltre un certo limite si comincia ad usare un indice