



Università degli  
studi di Salerno

# ASSEMBLAGGIO DI GENOMI TRAMITE DE BRUIJN GRAPH:

Studio del sequenziamento,  
Bloom Filter e analisi  
dell'assemblatore Minia

**Studenti**  
Vittorio Guida  
Costantino Paciello

**Professori**  
Rosalba Zizza  
Rocco Zaccagnino  
Clelia De Felice



# Problema: Memoria Proibitiva



## → PROBLEMA

- **Genoma umano** = 3 miliardi bp → 300+ GB con rappresentazione esplicita
- **De Bruijn Graph standard** = tabelle hash impraticabili per grandi dataset

## → COME RAPPRESENTARE IL GRAFO IMPLICITAMENTE MANTENENDO CORRETTEZZA?

- **Soluzione Minia** = 5.7 GB (50x riduzione!)

# Architettura



## BLOOM FILTER

11 bit/k-mer - Rappresenta nodi k-mer



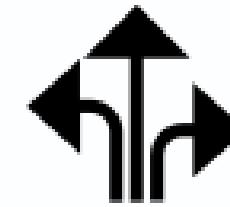
## STRUTTURA CFP

1.86 bit/k-mer - Corregge falsi positivi



## MARKING STRUCTURE

4.42 bit/k-mer - Traccia nodi visitati



Totale: 13.2 bit/k-mer  
(genoma umano)

# Self-information vs

# Minia

## Lower Bound:

- Teoria dell'informazione: servono 24 bit / k-mer per codificare esattamente un insieme di k-mer (genoma umano).



## Il paradosso di Minia:

- Minia utilizza solo 13.2 bit/k-mer.
- **Come è possibile?** Non supporta query di membership arbitrarie ("Esiste questo k-mer casuale?"), ma è ottimizzato solo per il traversal (seguire il cammino).
- **Risultato:** Scende sotto il limite teorico mantenendo l'esattezza dell'assemblaggio.

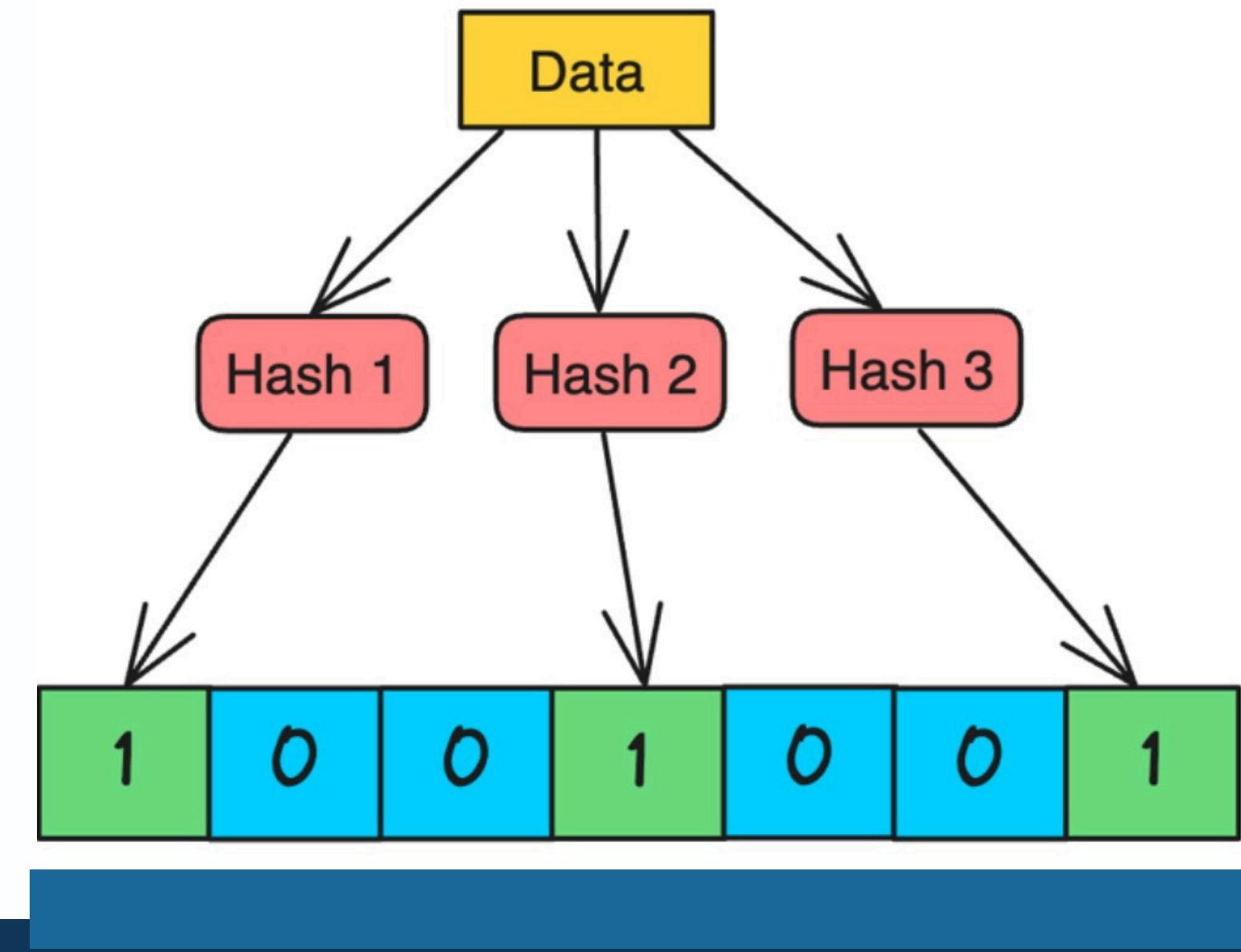
# Componente 1: Bloom Filter

## Struttura:

- **Nodi:** K-mer presenti nel Bloom Filter.
- **Archi:** Dedotti implicitamente (estensioni destra/sinistra).

## Meccanismo:

- Non salva il dato intero (il k-mer).
- Usa un Bit Array + k Funzioni Hash.
- "Accende" i bit corrispondenti all'elemento inserito.



## → VANTAGGI

- Riduzione della memoria del 90-95% rispetto alle Hash Table classiche.
- Nessun Falso Negativo: Se il filtro dice "No", il k-mer non c'è sicuramente.

## → COMPROMESSO

- Possibili Falsi Positivi: Può dire "Sì" per errore.

# De Bruijn Graph Probabilistico



## DEFINIZIONE FORMALE

Dato un insieme  $S$  di  $k$ -mer, il grafo probabilistico  $G_p = (V_p, E_p)$  è definito come:

- Nodi ( $V_p$ ): L'insieme di tutti i  $k$ -mer  $v$  per cui il filtro risponde positivamente.
  - $V_p = \{v \mid \text{BloomFilter.contains}(v) = \text{true}\}$
- Archi ( $E_p$ ): Esistono se l'estensione di un nodo è presente nel filtro.
  - $(u, v) \in E_p \Leftrightarrow v \in \text{ext}(u) \text{ & } v \in V_p$



## OVER-APPROSSIMAZIONE

Poiché i Bloom Filter producono falsi positivi ma mai falsi negativi, il grafo probabilistico è un'overso-approssimazione del grafo esatto  $G$

- $V \subseteq V_p$  e  $E \subseteq E_p$

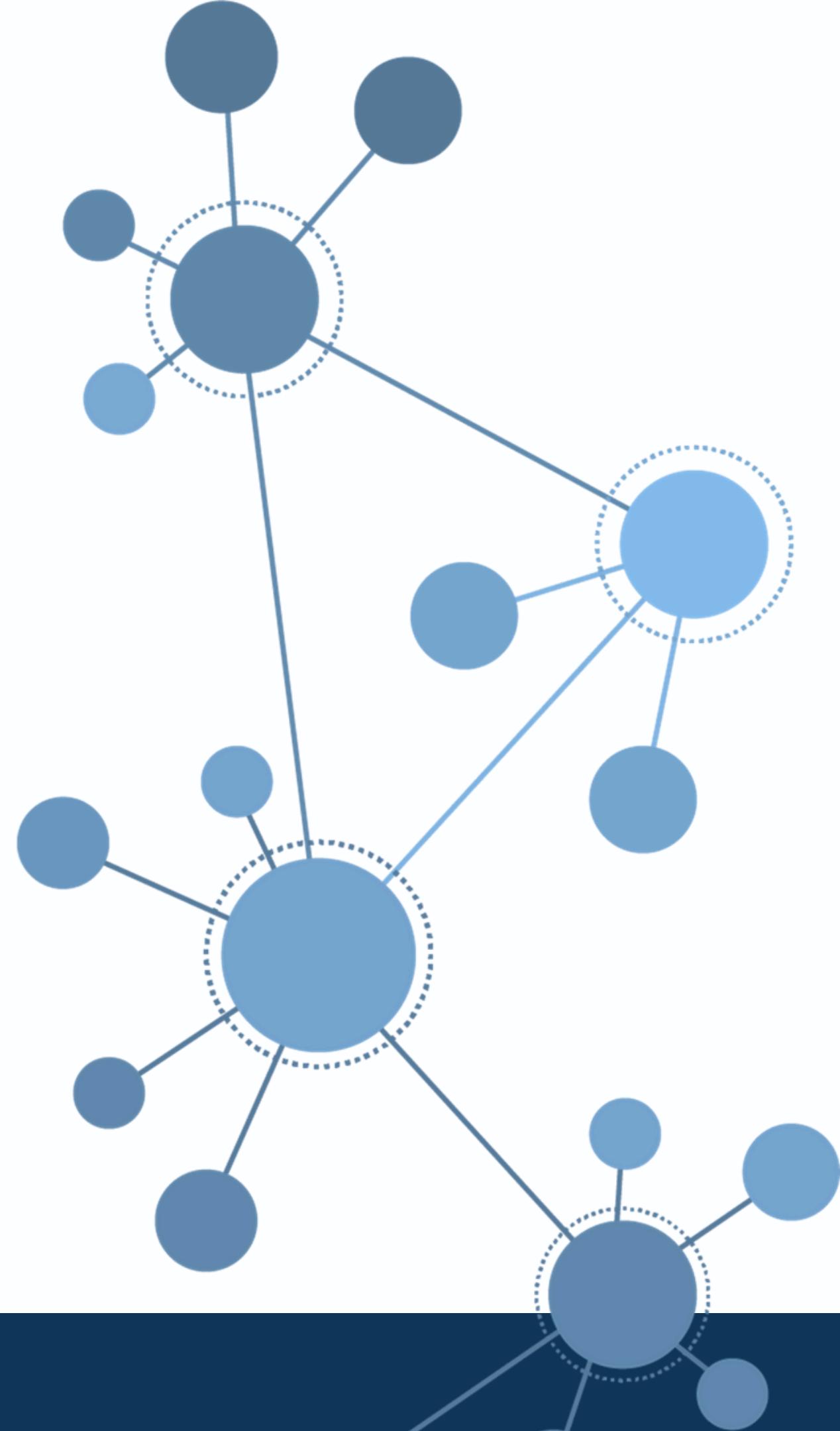


# Il Problema: Falsi Positivi

- Bloom Filter introduce ramificazioni spurie (false branching)
- Esempio:
  - Nodo reale V con 4 estensioni.  
BF riporta 5 (1 FP) → branching artificiale → contigs frammentati
- Con  $F=1\%$  FP: ~192 milioni estensioni spurie
- Assembly di qualità = impossibile

→ **SOLUZIONE**

Identificare e memorizzare i falsi positivi critici



# Componente 2: Critical False Positives (cFP)

**Definizione cFP:** Sono i falsi positivi che sono estensioni dirette di nodi reali.

- Solo questi vengono incontrati durante l'attraversamento.
  - $cFP = P \setminus S \rightarrow$  insieme cFP
  - $E[|cFP|] = 8 \cdot n \cdot F \rightarrow$  numero i cFP

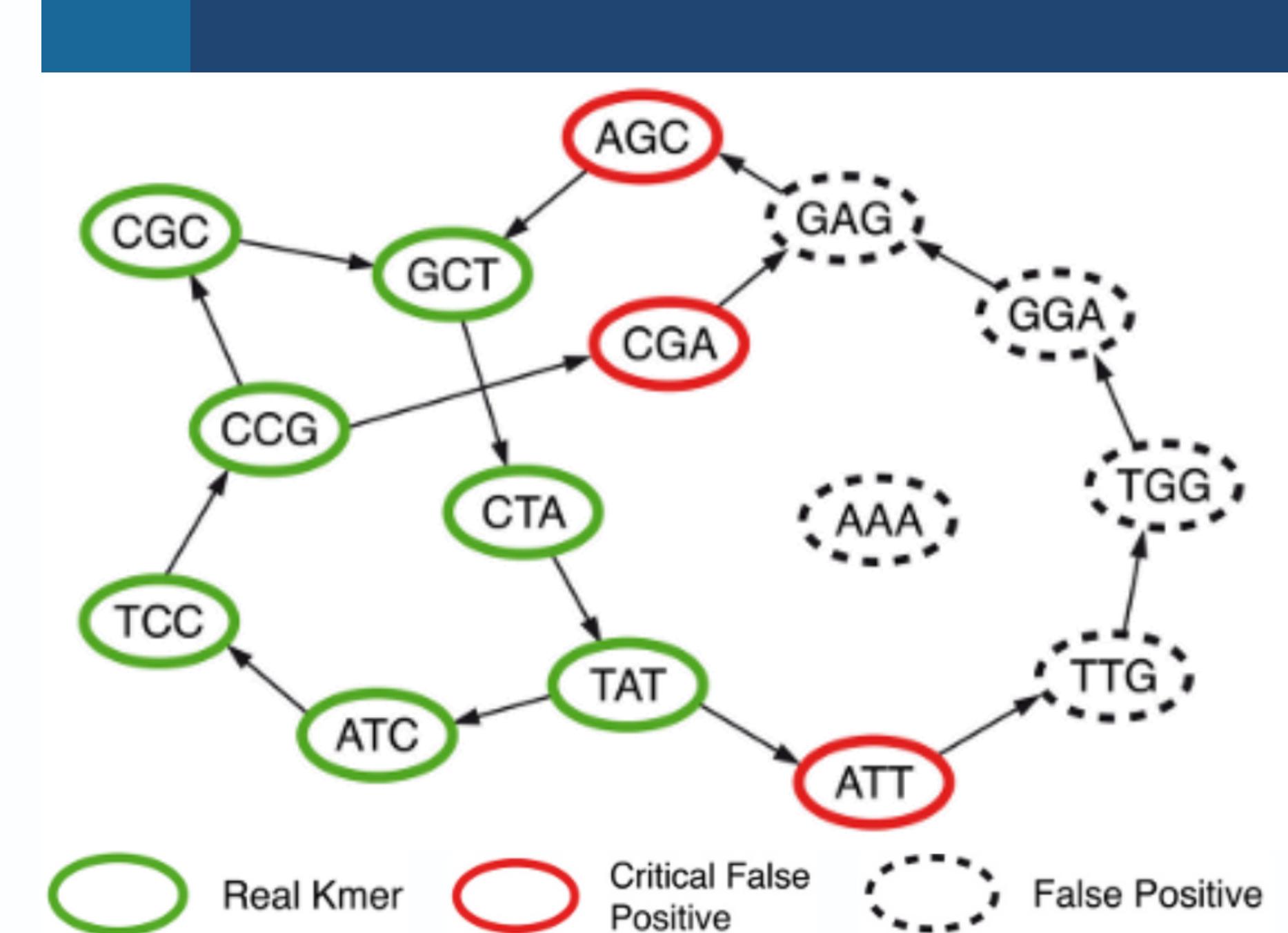
**Impatto per il genoma umano:**

- Solo il 2.9% dei k-mer totali è un cFP.
  - 1.86 bit/k-mer

Vengono salvati a parte in una struttura dedicata (Hash Table).

→ **INTUIZIONE**

Non tutti i falsi positivi danno fastidio.



## → PSEUDOCODICE

**Require:** Insieme  $S$  dei k-mer reali, Bloom Filter  $BF$ , parametro  $M$  (max elementi per partizione)

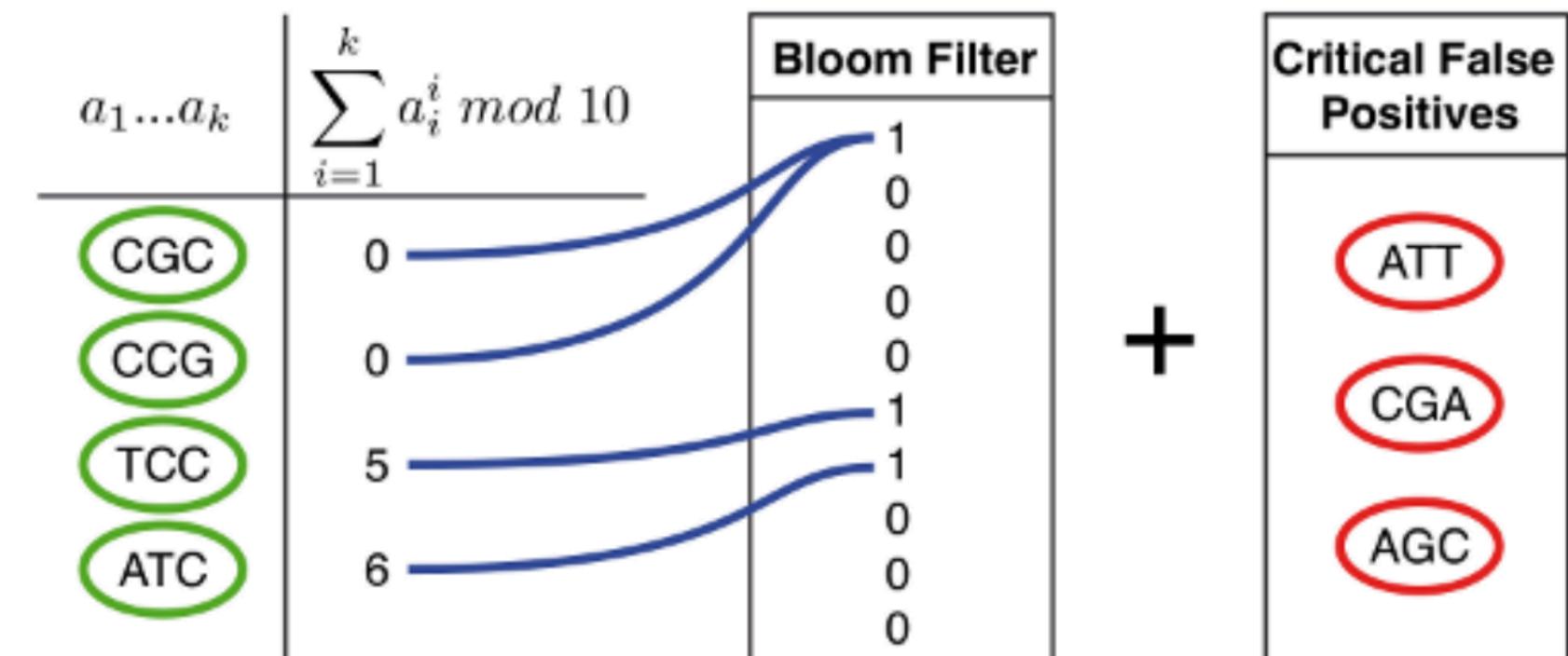
**Ensure:** Insieme cFP dei falsi positivi critici

```

1: Memorizzare su disco l'insieme  $P$  delle estensioni di  $S$  per cui  $BF$  risponde true
2: Liberare  $BF$  dalla memoria RAM
3:  $D_0 \leftarrow P$ 
4:  $i \leftarrow 0$ 
5: while non è stata raggiunta la fine di  $S$  do
6:    $P_i \leftarrow \emptyset$ 
7:   while  $|P_i| < M$  do
8:      $P_i \leftarrow P_i \cup \{\text{prossimo k-mer in } S\}$ 
9:   end while
10:  for ogni k-mer  $m \in D_i$  do
11:    if  $m \notin P_i$  then
12:       $D_{i+1} \leftarrow D_{i+1} \cup \{m\}$ 
13:    end if
14:  end for
15:  Eliminare  $D_i$  e  $P_i$  dal disco
16:   $i \leftarrow i + 1$ 
17: end while
18: cFP  $\leftarrow D_i$ 
19: return cFP

```

## Algoritmo 1: Permette di calcolare i cFP usando memoria costante





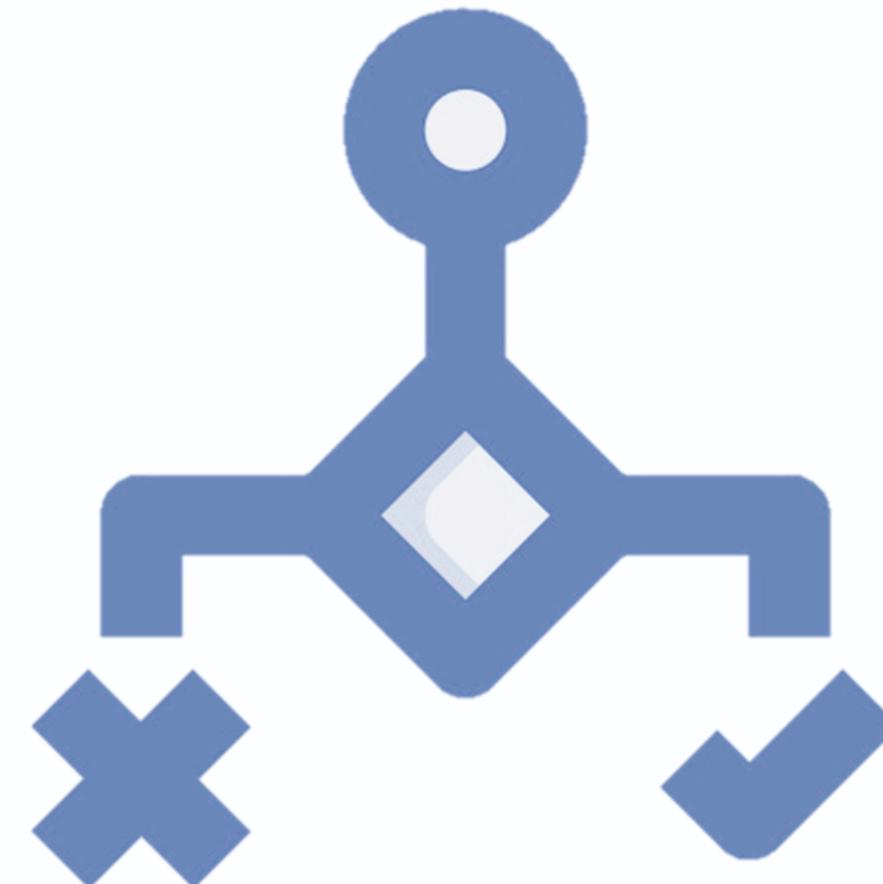
OGNI QUERY AL BLOOM FILTER  
VIENE "CORRETTA" COME SEGUE:

1. Se BF dice "assente" → return false
2. Se  $k\text{-mer} \in cFP$  → return false  
(CORREGGI FALSO POSITIVO)
3. Altrimenti → return true

**Risultato:**

Topologia = identica a grafo esatto deterministico

# Modifica delle query al Bloom Filter



# Dimensione ottimale Bloom Filter



## TRADE-OFF

- Bloom Filter grande → Pochi cFP (ma spreco RAM nel filtro).
- Bloom Filter piccolo → Tanti cFP (spreco RAM nella struttura cFP).

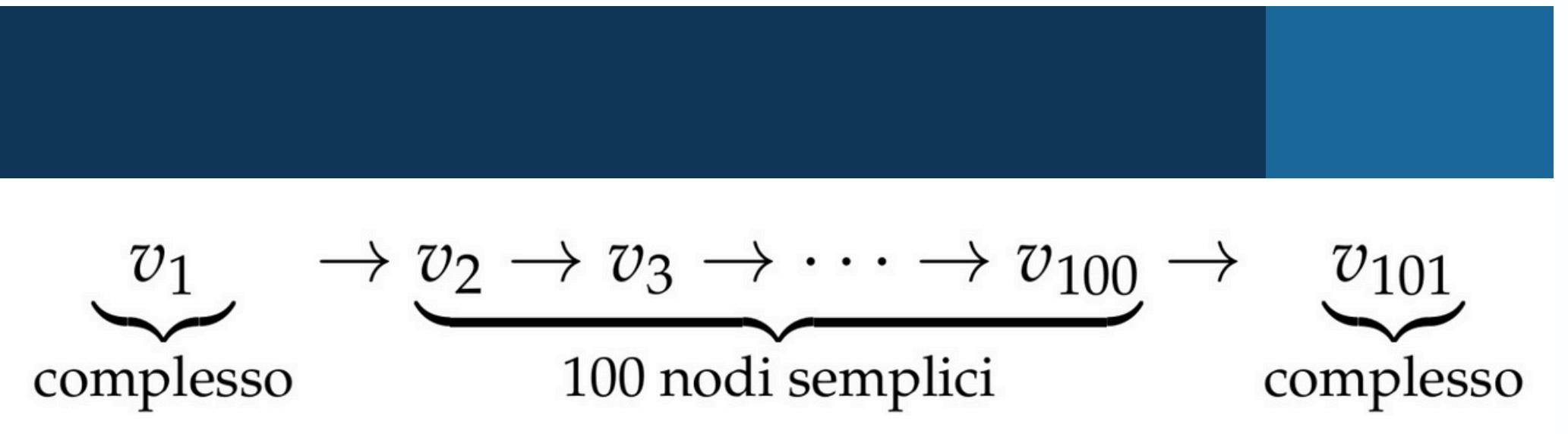
### Punto di ottimo:

- Per  $k=27$ , il minimo si ha a 11.1 bit/k-mer per il Bloom Filter.
- Struttura cFP: 1.86 bit/k-mer.
- Totale: 13.2 bit/k-mer.

**Scalabilità:** il costo di cFP è costante quando si sceglie  $F_{opt}$ , mentre il Bloom Filter cresce solo logaritmicamente rispetto a  $k$ .



# Componente 3: Marking Structure



## → PROBLEMA

Bisogna evitare cicli infiniti durante la visita (DFS/BFS).

## → LIMITI

Il Bloom Filter è immutabile (non possiamo scriverci "visitato").

## → SOLUZIONE Marcatura selettiva

- **Nodi Semplici** (lineari): Non vengono marcati. Condividono lo stato con il cammino.
- **Nodi Complessi** (biforazioni): Solo questi vengono memorizzati nella MarkingStructure.

## → RISPARMIO

Solo il 6.1% dei nodi è complesso:

- Memoria ridotta a 4.42 bit/k-mer.

# Algoritmo 3: Attraversamento del grafo con marcatura selettiva

→ PSEUDOCODICE

```
1: procedure TRAVERSEGRAPH
2:   for ogni k-mer  $v \in S$  non ancora processato do
3:     if  $v$  è complesso and  $v \in$  markingSet then
4:       continue
5:     end if
6:     contig  $\leftarrow []$ 
7:     EXTENDPATH( $v$ , contig)
8:     Output contig
9:   end for
10: end procedure

11: procedure EXTENDPATH( $v$ , contig)
12:   if  $v$  è complesso then
13:     markingSet  $\leftarrow$  markingSet  $\cup \{v\}$ 
14:   end if
15:   Aggiungi  $v$  a contig
16:   neighbors  $\leftarrow$  GETNEIGHBORS( $v$ )
17:   if |neighbors| = 1 then
18:      $w \leftarrow$  neighbors[0]
19:     EXTENDPATH( $w$ , contig)
20:   else
21:     return
22:   end if
23: end procedure

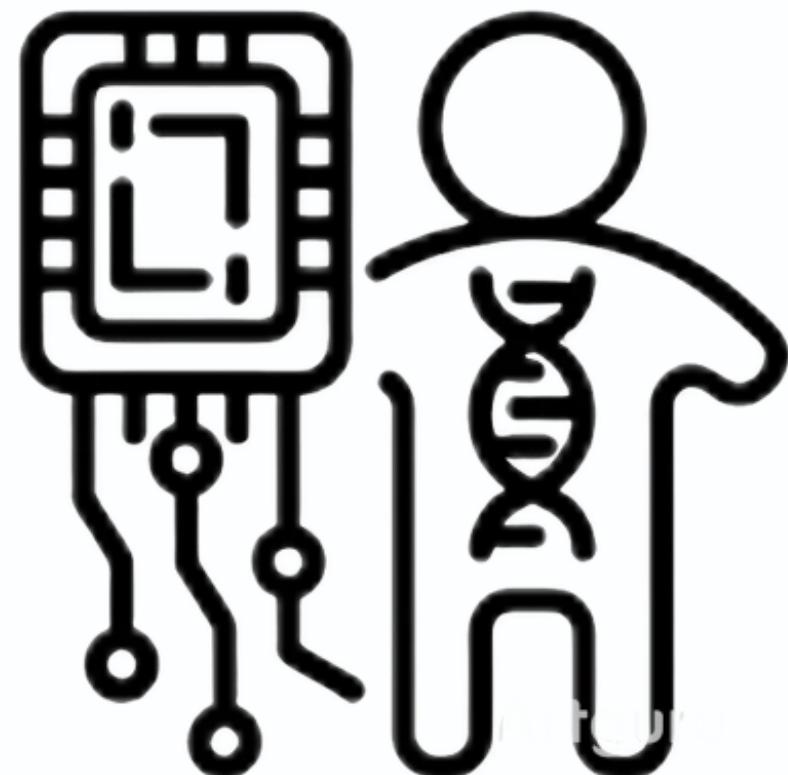
24: procedure GETNEIGHBORS( $v$ )
25:   neighbors  $\leftarrow []$ 
26:   for  $x \in \{A, C, G, T\}$  do
27:      $w \leftarrow$  rightExtension( $v$ ,  $x$ )
28:     if CONTAINS( $w$ ) then
29:       neighbors  $\leftarrow$  neighbors  $\cup \{w\}$ 
30:     end if
31:   end for
32:   return neighbors
33: end procedure
```



## GENOMA UMANO

Componente	Dimensione	bit/k-mer	% totale
Bloom Filter	3.75 GB	11.1	65.8%
cFP	0.53 GB	1.86	9.3%
Marking structure	1.29 GB	4.42	22.6%
Overhead sistema	0.13 GB	0.48	2.3%
<b>Totale</b>	<b>5.7 GB</b>	<b>17.4</b>	<b>100%</b>

# Memoria totale



# Validazione Sperimentale

## → RIDUZIONE DRASTICA DELLA MEMORIA

5.7 GB per il genoma umano, 59x meno di ABYSS

## → ACCURATEZZA PRESERVATA

94.6% dei contigs corretti, comparabile agli assemblatori tradizionali

## → CORRETTEZZA GARANTITA

La struttura cFP elimina gli errori introdotti dai falsi positivi del Bloom Filter

## → SCALABILITÀ

Primo assemblatore in grado di processare un genoma umano completo su un desktop consumer



## GENOMA UMANO

Metrica	Minia	C.&B.	ABYSS	SOAPdenovo
Parametro $k$	27	27	27	25
N50 (bp)	<b>1,156</b>	250	870	886
Basi assemblate (Gbp)	2.09	1.72	2.10	2.08
Cores utilizzati	1	8	21,168	116
Tempo (ore)	23	50	15	33
<b>Memoria (GB)</b>	<b>5.7</b>	<b>32</b>	<b>336</b>	<b>140</b>

# Innovazione & Impatto



## SUCCESSI

Minia dimostra che l'approccio probabilistico (Bloom Filter) è utilizzabile per l'assemblaggio esatto.

Elementi chiave del successo:

1. Struttura CFP (corregge gli errori con poca memoria).
2. Marcatura selettiva (marca solo i nodi complessi).



## LIMITI E SVILUPPI FUTURI

Produce contigs e non scaffolds (manca info paired-end).





Università degli  
studi di Salerno

# GRAZIE PER L'ATTENZIONE



[www.github.com/minia-analysis](https://www.github.com/minia-analysis)



v.guid16@studenti.unisa.it

c.paciello7@studenti.unisa.it



[www.unisa.it](http://www.unisa.it)

**Studenti**

Vittorio Guida  
Costantino Paciello

**Professori**

Rosalba Zizza  
Rocco Zaccagnino  
Clelia De Felice

