Succincter

Mihai Pătrașcu

$\longrightarrow$ unemployed $\xrightarrow{?}$ $\xrightarrow{?}$ at&t

Storing trits

Store $A[1..n] \in \{1,2,3\}^n$ to retrieve any $A[i]$ efficiently

Credits:

@Max Plank, 2005

Storing trits

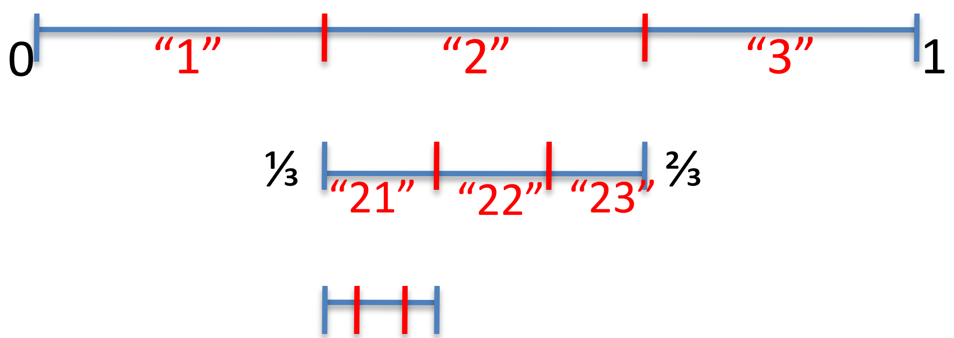
Store $A[1..n] \in \{1,2,3\}^n$ to retrieve any $A[i]$ efficiently

	Space	**Time**
trivial	2n	O(1)

Storing trits

Store $A[1..n] \in \{1,2,3\}^n$ to retrieve any $A[i]$ efficiently

	Space	**Time**
trivial	$2n$	$O(1)$
arithmetic coding	$\lceil n \cdot \log_2 3 \rceil$	$\tilde{O}(n)$



Storing trits

Store $A[1..n] \in \{1,2,3\}^n$ to retrieve any $A[i]$ efficiently

	Space	**Time**
trivial	$2n$	$O(1)$
arithmetic coding	$\lceil n\cdot\log_2 3\rceil$	$\tilde{O}(n)$
block coding	$\lceil n\cdot\log_2 3\rceil + O(n/\tau)$	$\tilde{O}(\tau)$

redundancy

Storing trits

Store $A[1..n] \in \{1,2,3\}^n$ to retrieve any $A[i]$ efficiently

	Space	**Time**
trivial	$2n$	$O(1)$
arithmetic coding	$\lceil n \cdot \log_2 3 \rceil$	$\tilde{O}(n)$
block coding	$\lceil n \cdot \log_2 3 \rceil + O(n/\tau)$	$\tilde{O}(\tau)$

Hmm... If a block uses $O(1)$ redundancy, the encoding must spread information around

Storing trits

Store $A[1..n] \in \{1,2,3\}^n$ to retrieve any $A[i]$ efficiently

	Space	**Time**
trivial	$2n$	$O(1)$
arithmetic coding	$\lceil n \cdot \log_2 3 \rceil$	$\tilde{O}(n)$
block coding	$\lceil n \cdot \log_2 3 \rceil + O(n/\tau)$	$\tilde{O}(\tau)$
succinct data structures	$\lceil n \cdot \log_2 3 \rceil + O(n/\lg n)$	$O(1)$

Succinct Data Structures

"Make data structures use space close to optimum (≈entropy)"

- dictionaries
  * classic hash tables use $O(n \cdot \lg u)$ bits
  * strive for $H = \log \binom{u}{n}$
- pattern matching
  * suffix trees use $O(n \cdot \lg n)$ bits
  * strive for $H = n \cdot \lg \Sigma$
- represent trees with fast navigation (also graphs, etc)
  * trivial representations use $O(n \cdot \lg n)$ bits
  * strive for $H = \lg C_n \approx 2n$
- etc

Succinct Data Structures

"Make data structures use space close to optimum (≈entropy)"

Down deep, common technique:

* blocks of $\varepsilon \lg n$ elements stored with redundancy ≤ 1
* tabulation to handle blocks

$\Rightarrow$ Space $\approx H + O(H/(\tau \lg n))$ with time $O(\tau)$

- * trivial representations use $O(n \cdot \lg n)$ bits
  * strive for $H = \lg C_n \approx 2n$
- etc

Storing trits

Store $A[1..n] \in \{1,2,3\}^n$ to retrieve any $A[i]$ efficiently

	Space	**Time**
trivial	$2n$	$O(1)$
arithmetic coding	$\lceil n \cdot \log_2 3 \rceil$	$\tilde{O}(n)$
block coding	$\lceil n \cdot \log_2 3 \rceil + O(n/\tau)$	$\tilde{O}(\tau)$
succinct data structures	$\lceil n \cdot \log_2 3 \rceil + O(n/\lg n)$	$O(1)$
[Golynski et al '07, '08]	$\lceil n \cdot \log_2 3 \rceil + O(n/\lg^2 n)$	$O(1)$

Storing trits

Store $A[1..n] \in \{1,2,3\}^n$ to retrieve any $A[i]$ efficiently

	Space	**Time**
trivial	$2n$	$O(1)$
arithmetic coding	$\lceil n \cdot \log_2 3 \rceil$	$\tilde{O}(n)$
block coding	$\lceil n \cdot \log_2 3 \rceil + O(n/\tau)$	$\tilde{O}(\tau)$
succinct data structures	$\lceil n \cdot \log_2 3 \rceil + O(n/\lg n)$	$O(1)$
[Golynski et al ’07, ’08]	$\lceil n \cdot \log_2 3 \rceil + O(n/\lg^2 n)$	$O(1)$
Here…	$\lceil n \cdot \log_2 3 \rceil + O(n/\lg^\tau n)$	$O(\tau)$

Discussion

Big new concept:

** use recursion to reduce redundancy **

Many applications:

succinct data structures with space $O(n/\lg^c n)$, $\forall c$

How am I getting away with improving on a constant?

A Succinct Proof

Spill-Over Encodings

Say I want to represent $x \in X$ (think $X=\{1,2,3\}^B$)

If $|X|$ is not a power of 2, how to get redundancy $<<1$?

New encoding framework:

$x \rightarrow$ M bits + a **spill** in $\{1,..,K\}$

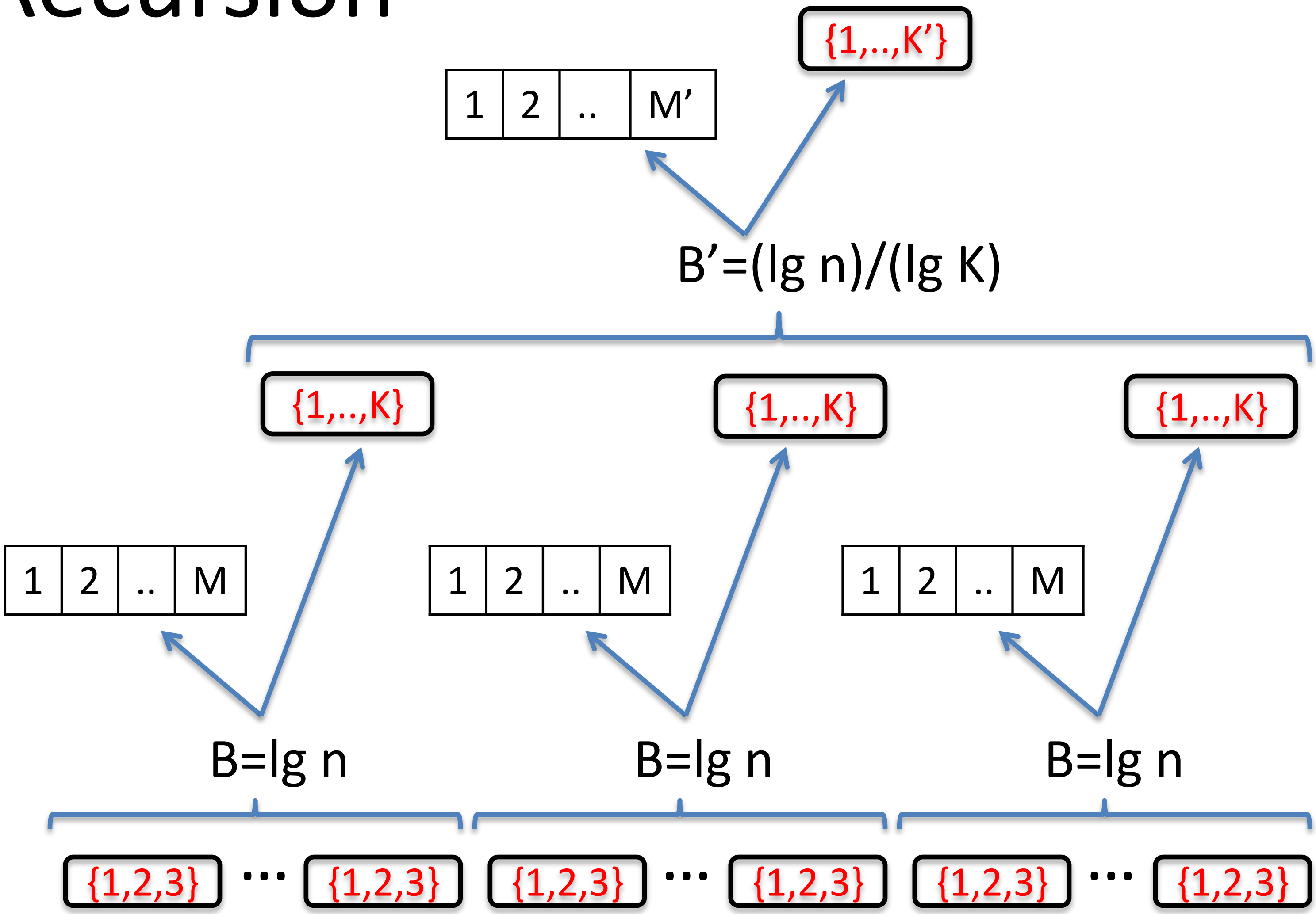
Redundancy = wasted entropy = $M + \lg K - \lg |X|$

... need to approximate $\lg|X| \approx$ integer + lg(integer)

$M + \lg(K-1) < \lg|X| < M + \lg K$

$\Rightarrow$ redundancy $\leq \lg K - \lg(K-1) = O(1/K)$

Recursion



{1,..,K'}
1
2
..
M'
B'=(lg n)/(lg K)
{1,..,K}
{1,..,K}
{1,..,K}
1
2
..
M
1
2
..
M
1
2
..
M
B=lg n
B=lg n
B=lg n
{1,2,3}
…
{1,2,3}
{1,2,3}
…
{1,2,3}
{1,2,3}
…
{1,2,3}

Analysis

- choose $K, K', K'', \ldots = \Theta(\kappa)$
- redundancy at each node = $O(1/\kappa)$
  times ≈n nodes ⇒ $O(n/\kappa)$ bits
- degree $B', B'', \ldots = \Theta(\,(\lg n) / (\lg \kappa)\,)$
- go up until $O(n/\kappa)$ nodes left, then waste 1 bit/node
  ⇒ query time $\tau = O(\lg_{B'} \kappa)$
  ⇒ $\kappa = ((\lg n) / \tau)^{\tau}$
  ⇒ redundancy $n / ((\lg n) / \tau)^{\tau} \approx n/\lg^{\tau} n$

Succinct(er) Data Structures

A Building Block

The "Rank/Select Problem":

Store $A[1..n] \in \{0,1\}^n$ subject to:

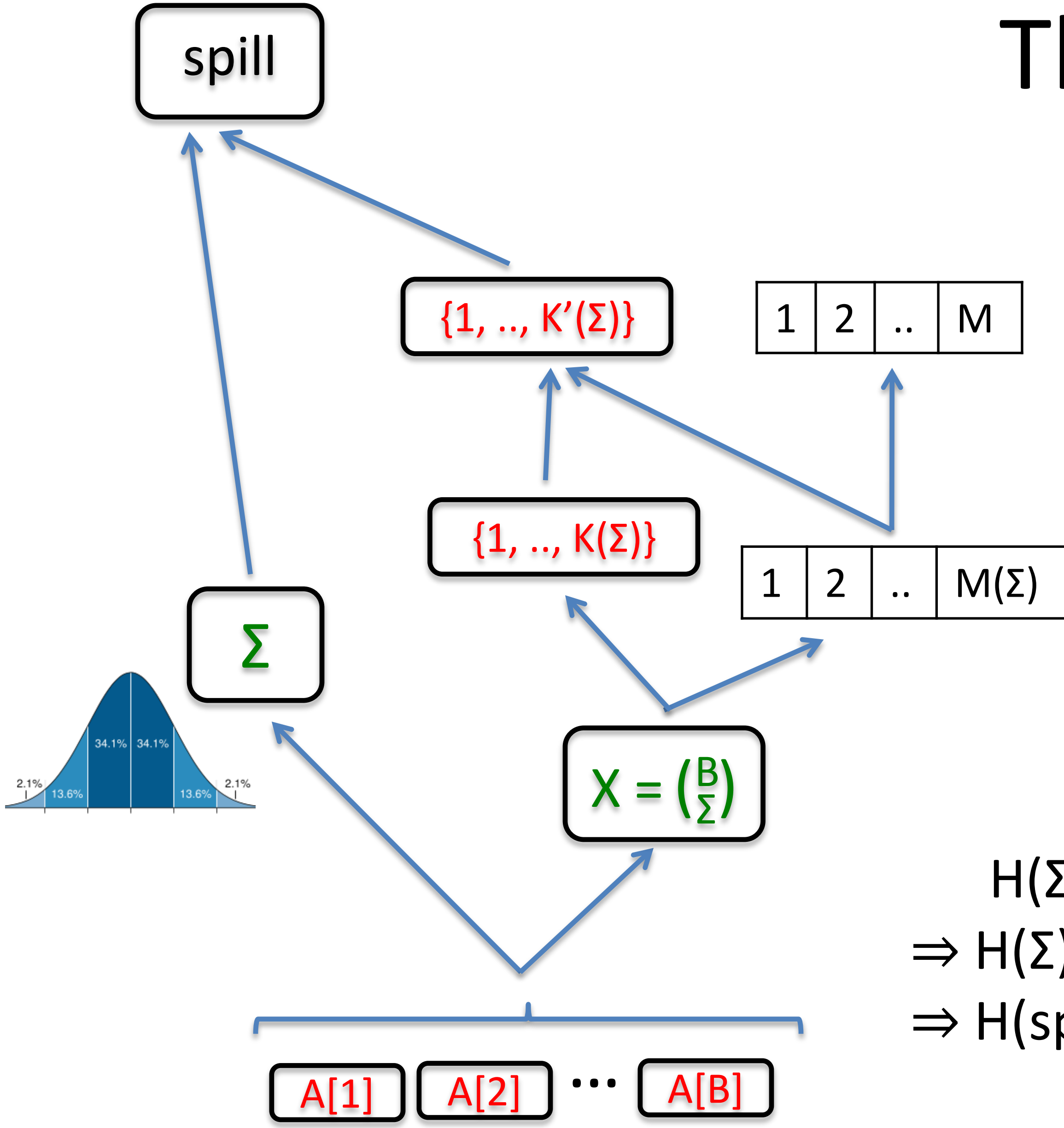
* rank(k): return $\sum_{i=1}^{k} A[i]$

* select(j): find k such that $\text{rank}(k)=j$

[Golynski et al '07] space $n + O(n \cdot \lg\lg n \;/\; \lg^2 n)$, query $O(1)$

[P. FOCS'08] space $n + O(n / \lg^c n)$, query $O(c)$

The Algorithm



$$H(\Sigma) + H(A \mid \Sigma) = n$$

$$\Rightarrow H(\Sigma) + \lg K(\Sigma) + M \approx n$$

$$\Rightarrow H(\text{spill}) \approx n - M$$

The End

Questions?