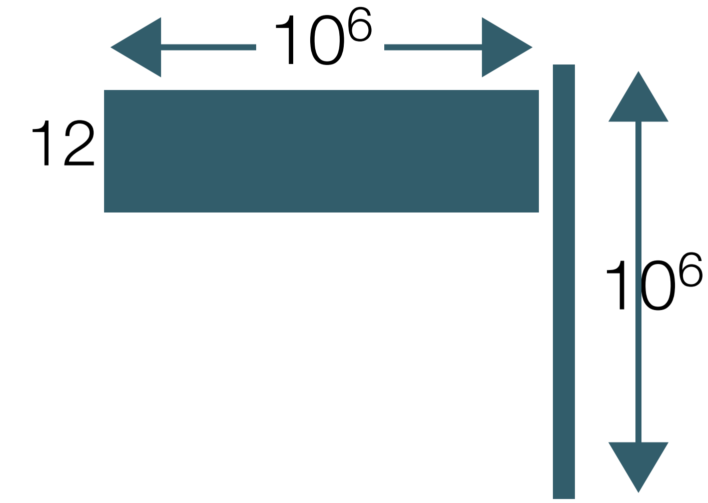
Part 2:
Codes for distributed linear data processing in presence of straggling/faults/errors

Motivation: nonideal computing systems

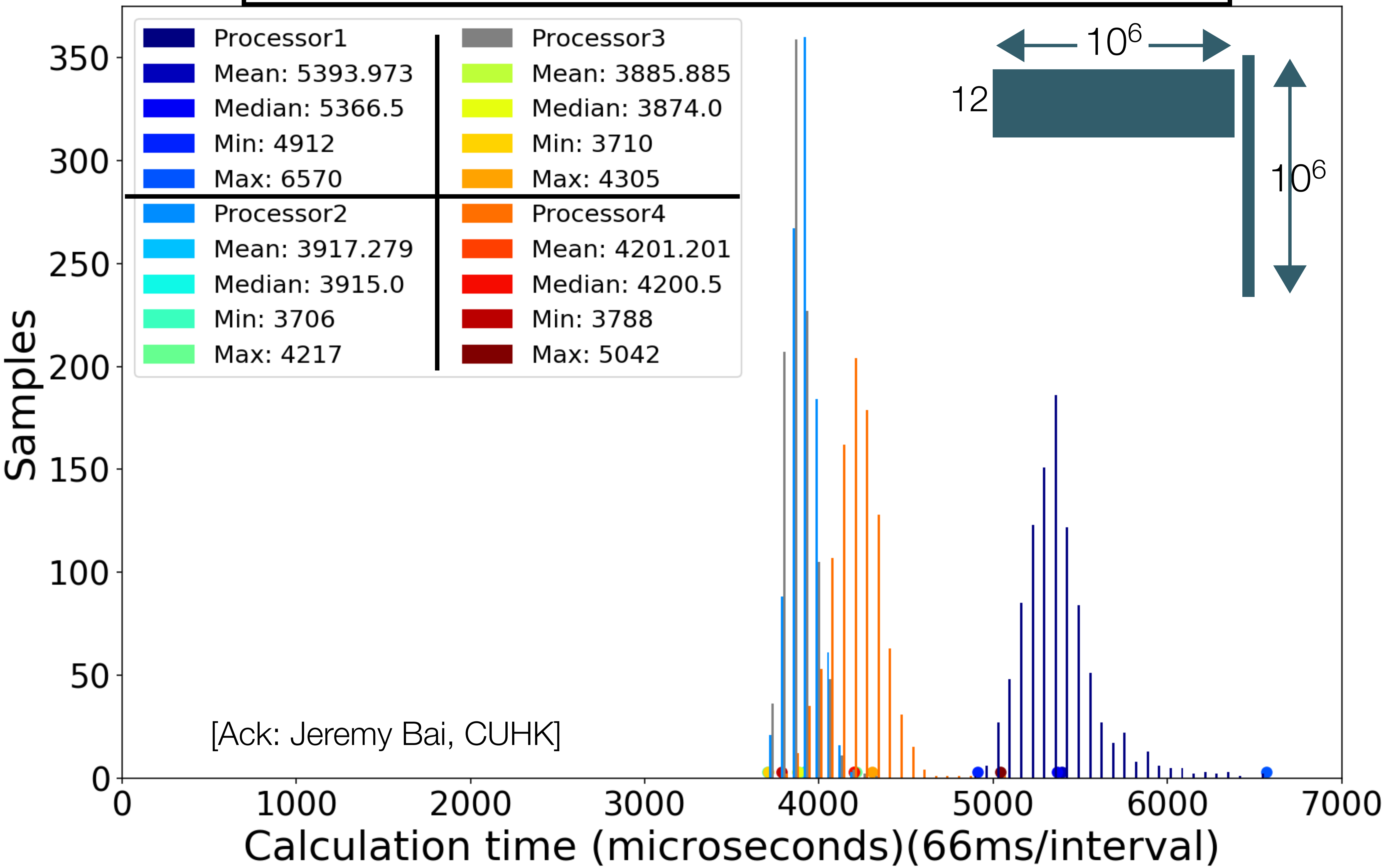
Motivation: nonideal computing systems

M x V for 4 processors on AmazonEC2 cloud system



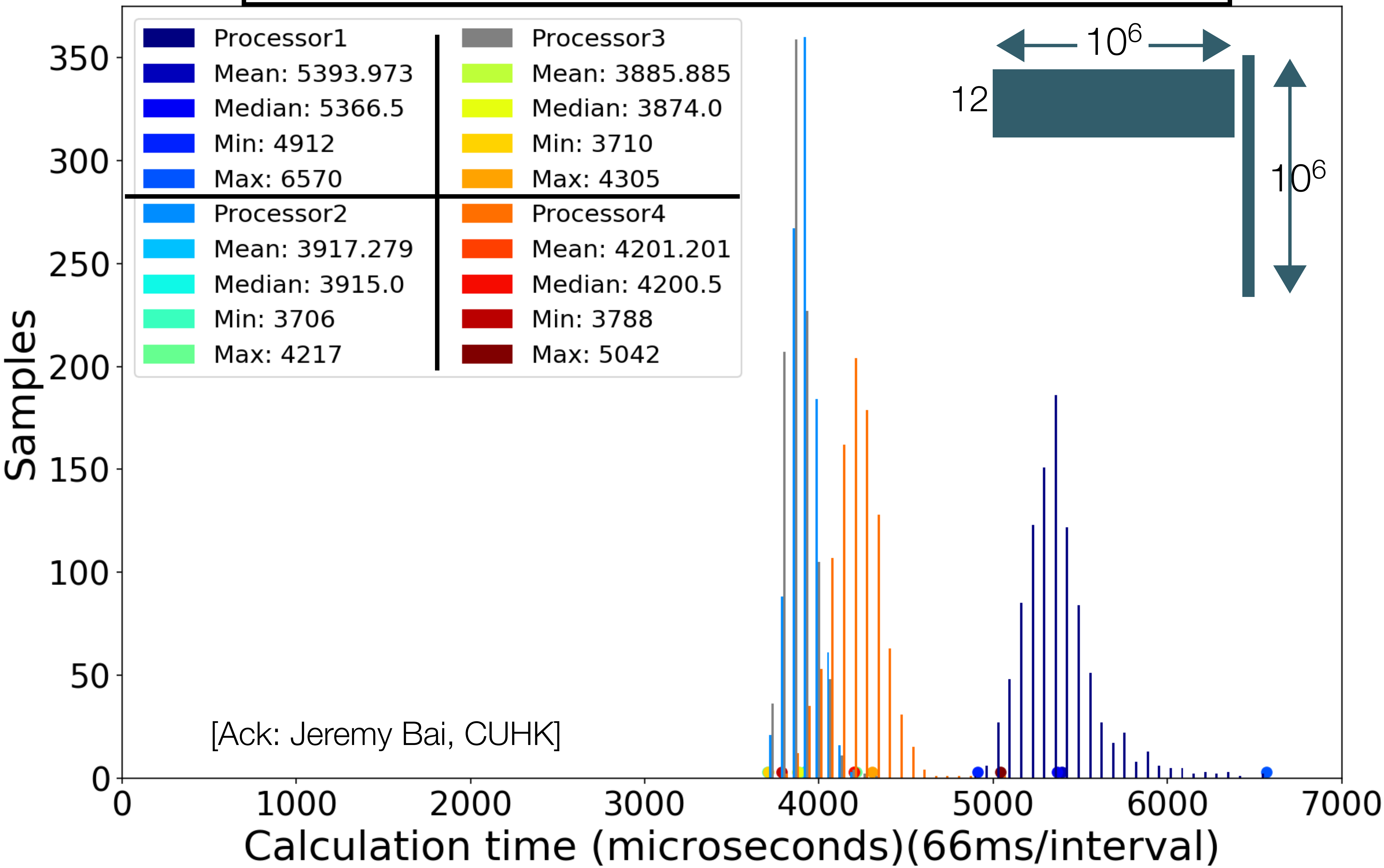
Motivation: nonideal computing systems

M x V for 4 processors on AmazonEC2 cloud system



Motivation: nonideal computing systems

M x V for 4 processors on AmazonEC2 cloud system

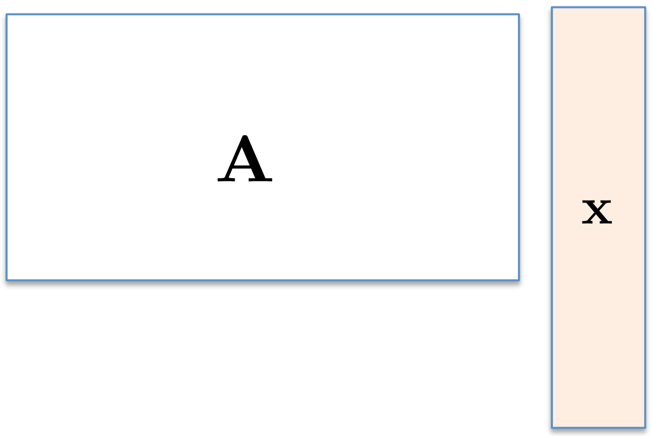


[Ack: Jeremy Bai, CUHK]

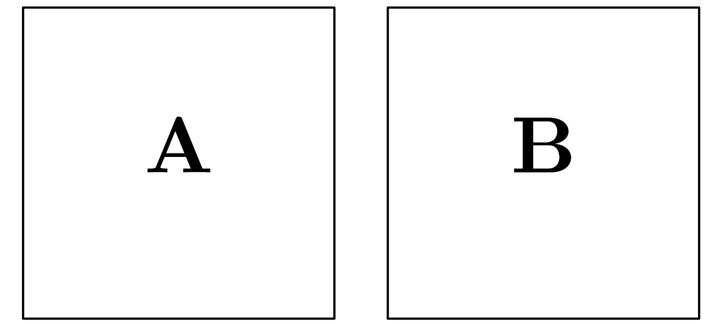
Practitioners are **already** using redundancy to address straggling

Organization: How to perform these computations?

1.



2.



efficiently, fast, in presence of faults/straggling/errors

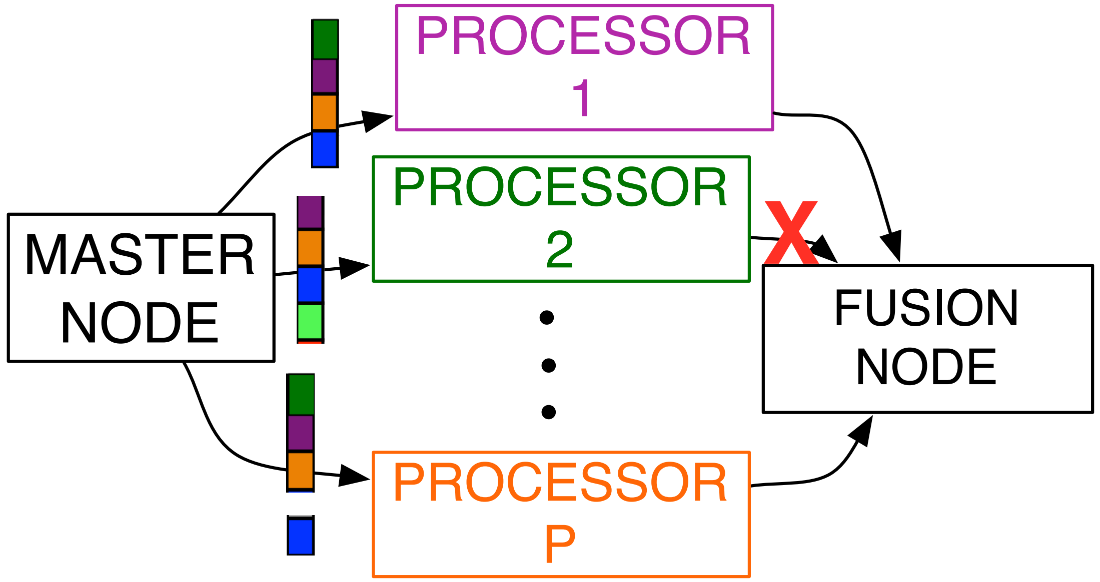
Motivation: *The* critical steps for many compute applications (Machine learning: neural nets, LDA, PCA, Regression, Projections. Scientific computing and physics simulations)

Rest of the tutorial is divided into two parts:

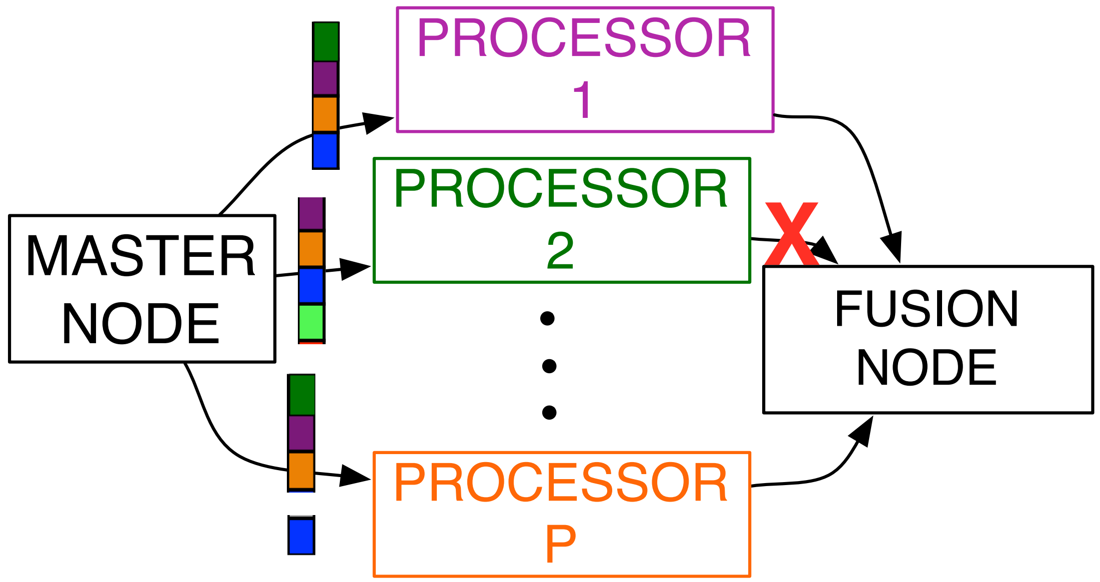
I. Big processors [Huang, Abraham '84]
II. Small processors [von Neumann '56]

Part I: Big processors

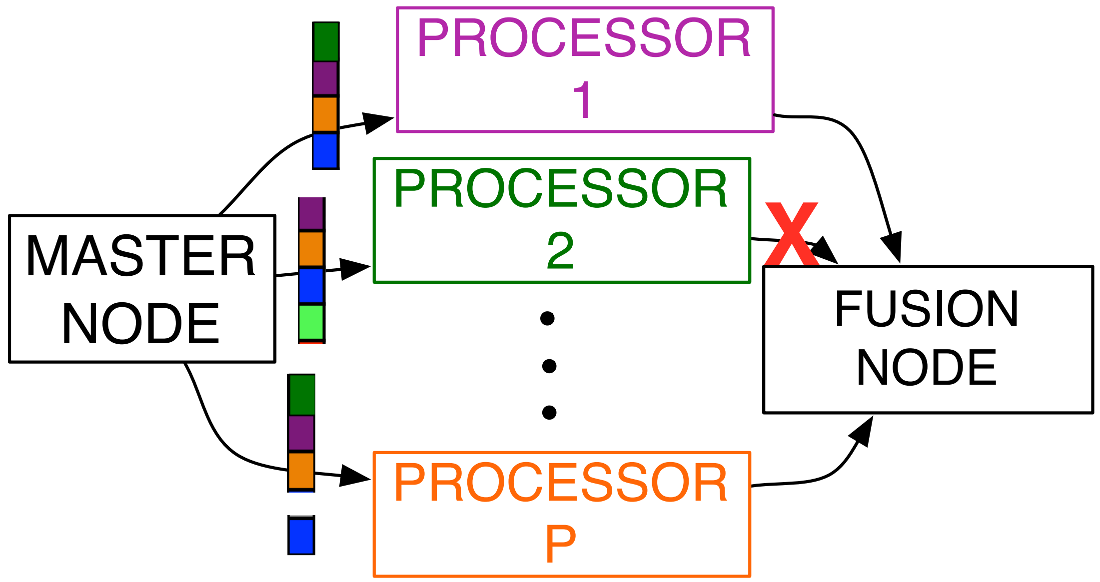
Processor memory scales with problem size



System metrics



System metrics



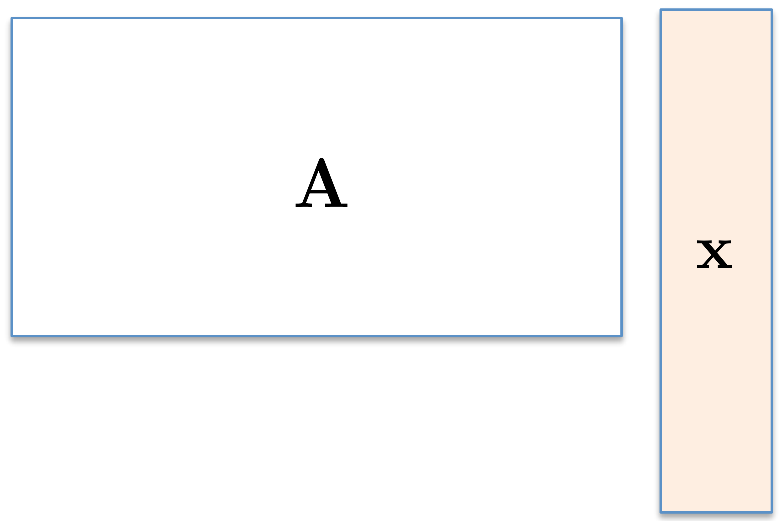
1. Per-processor computation costs:
   - # operations/processor

2. Straggler tolerance (directly related to “recovery threshold”)
   - max # processors that can be ignored by fusion node

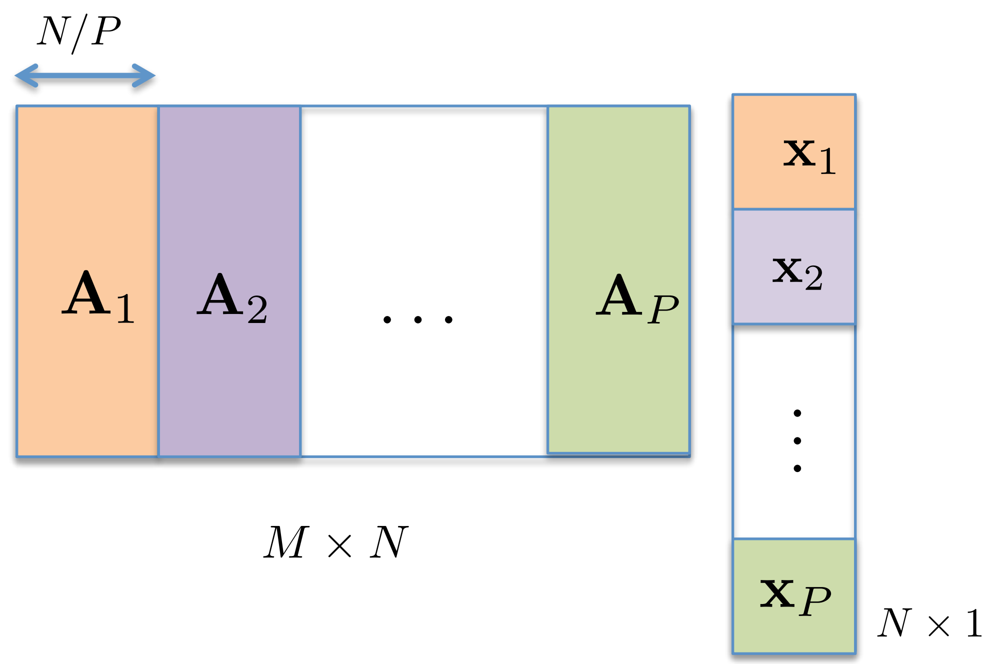
3. Communication costs
   - number of bits exchanged between all processors
   - can use more sophisticated metrics. See [Bruck et al.’97]

“Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems”
Bruck, Ho, Kipnis, Upfal, Weathersby ‘97

I.1



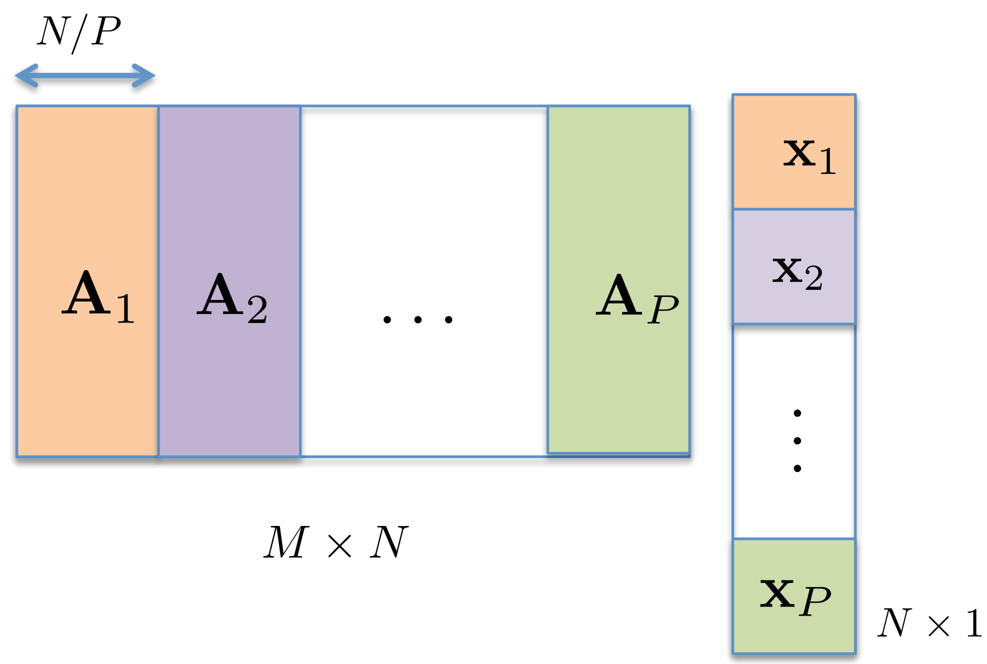
Parallelization for speeding up matrix-vector products



P processors (master node aggregates outputs)

Operations/processor: MN/P (e.g. P=3, each does 1/3rd computations)

Parallelization for speeding up matrix-vector products



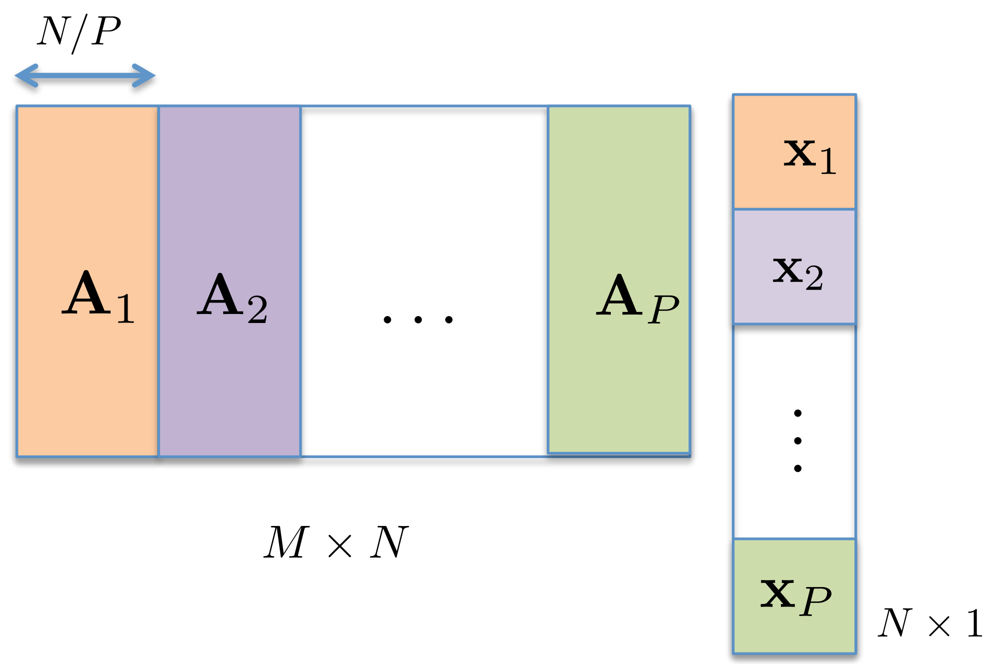
P processors (master node aggregates outputs)

Operations/processor: MN/P (e.g. P=3, each does 1/3rd computations)

In practice, processors can be delayed ("stragglers") or faulty

Recovery threshold = P *i.e., Straggler tolerance = 0*

Parallelization for speeding up matrix-vector products



P processors (master node aggregates outputs)

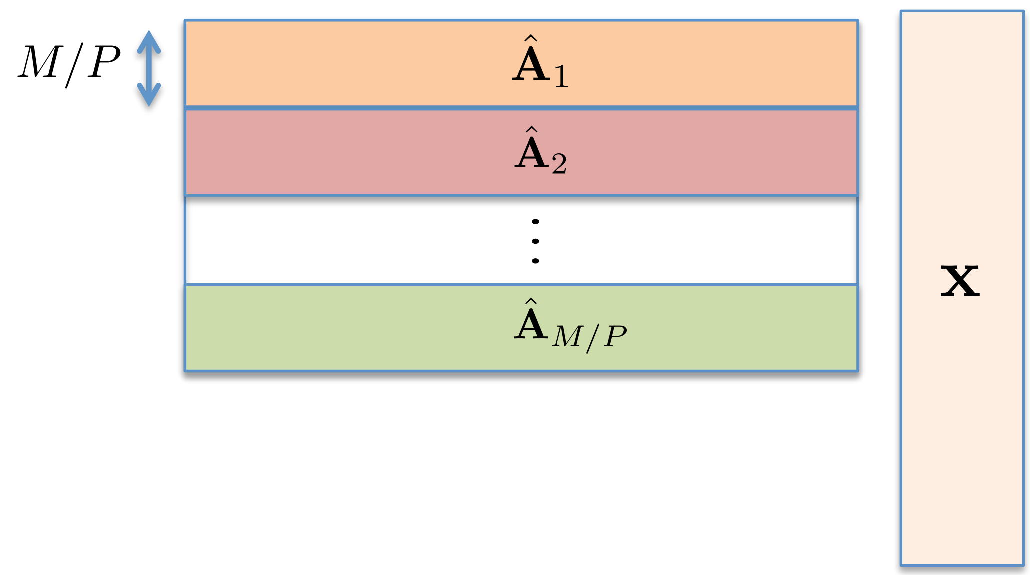
Operations/processor: MN/P (e.g. P=3, each does 1/3rd computations)

In practice, processors can be delayed ("stragglers") or faulty

Recovery threshold = P *i.e., Straggler tolerance = 0*

Note: can parallelize by dividing the matrix horizontally as well

Parallelization for speeding up matrix-vector products



P processors (master node aggregates outputs)

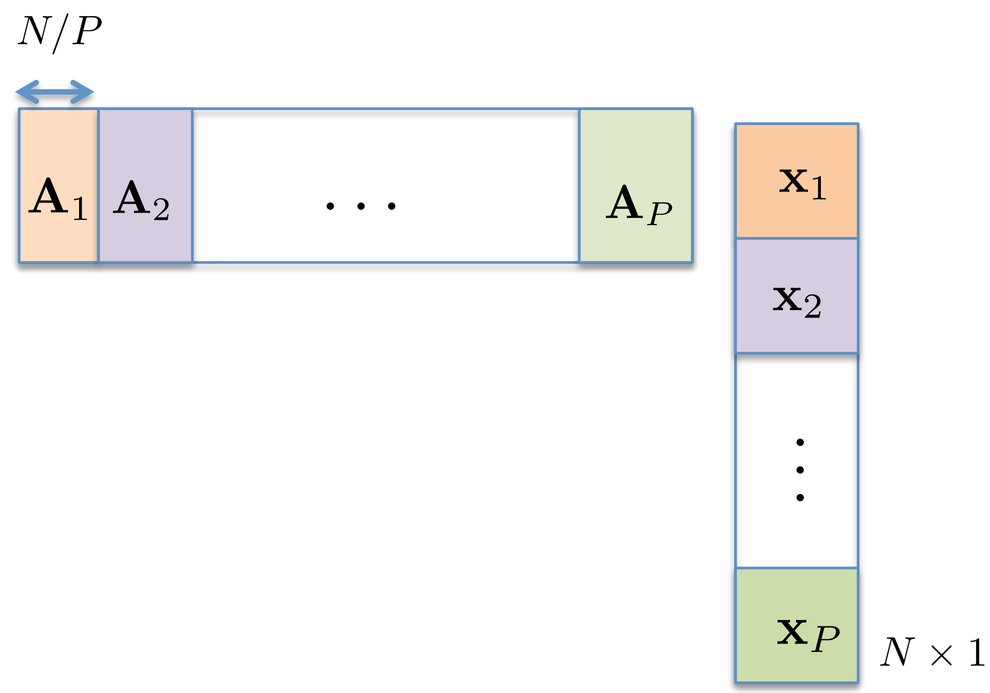
Operations/processor: MN/P (e.g. P=3, each does 1/3rd computations)

In practice, processors can be delayed ("stragglers") or faulty

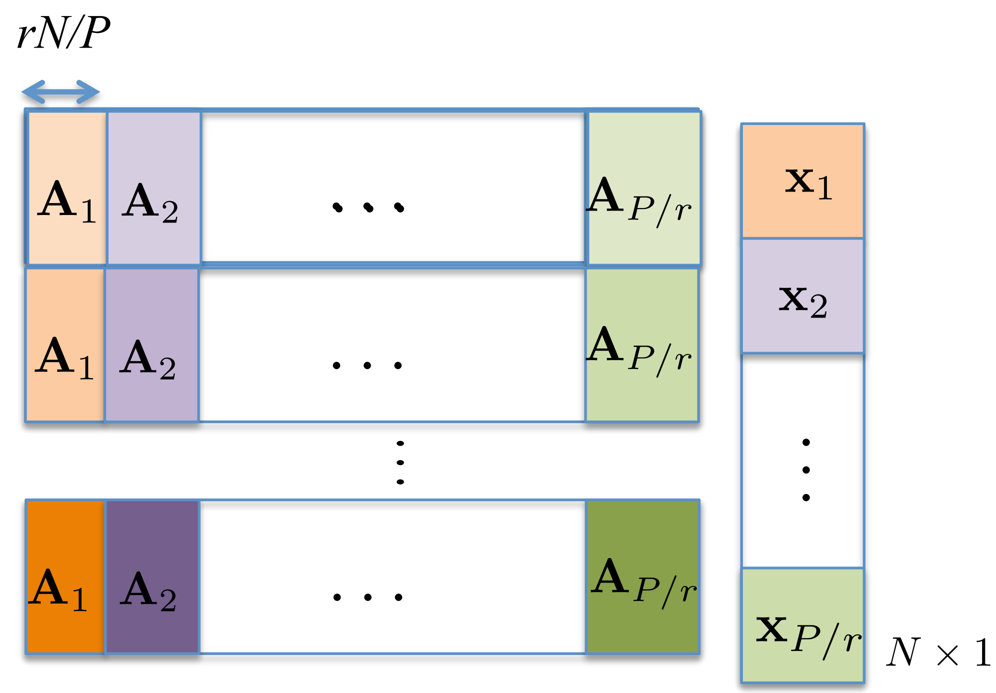
Recovery threshold = P *i.e., Straggler tolerance = 0*

Note: can parallelize by dividing the matrix horizontally as well

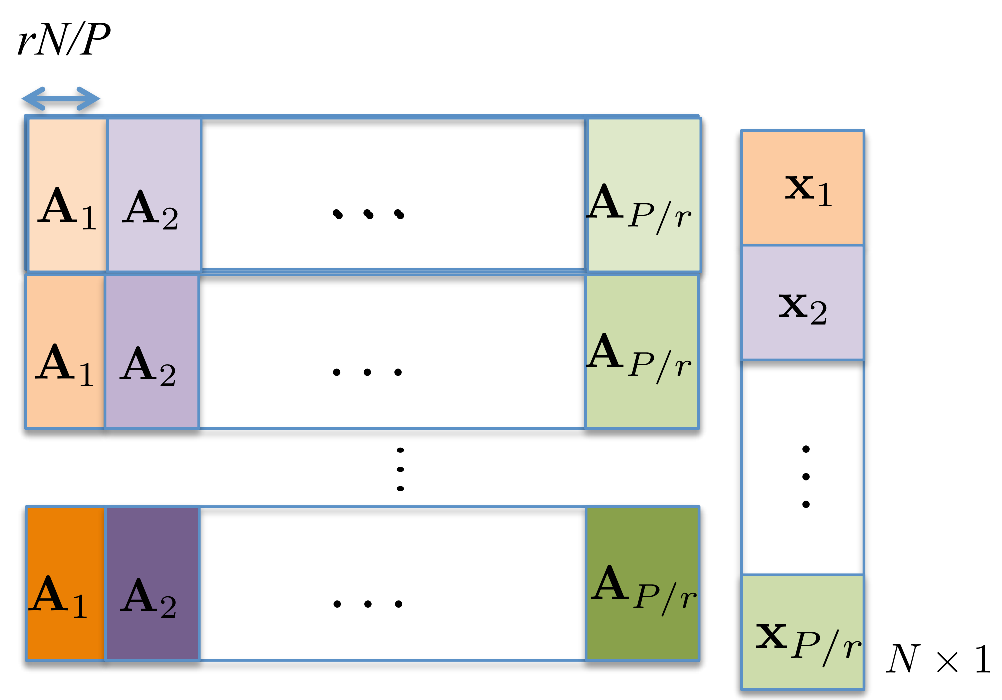
Replication: repeat Job *r* times



Replication: repeat Job *r* times



Replication: repeat Job r times

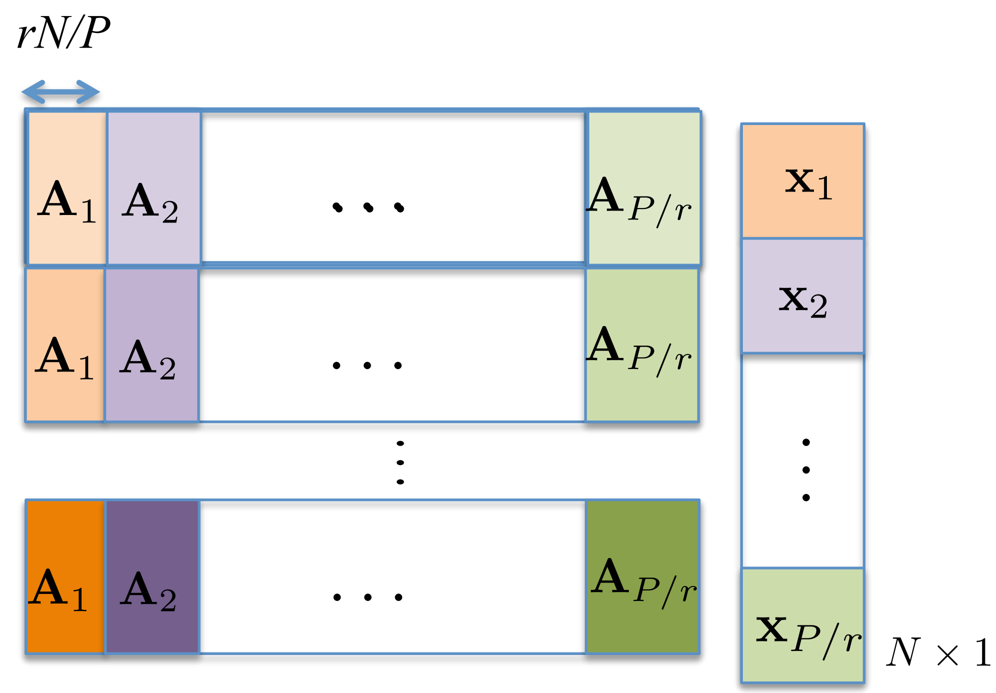


P processors

\# operations/processor: rMN/P

Straggler tolerance: r-1 Recovery threshold: P-r+1

Replication: repeat Job *r* times



P processors

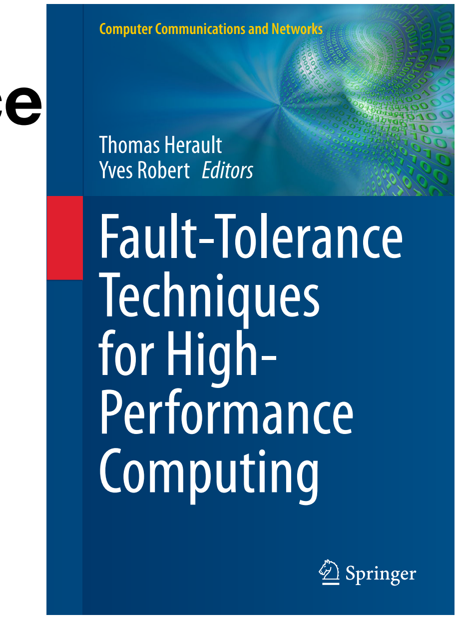
operations/processor: rMN/P

Straggler tolerance: r-1 Recovery threshold: P-r+1

Also see: recent works of [Joshi, Soljanin, Wornell]

A coding alternative to replication: MDS compute codes (“ABFT”)

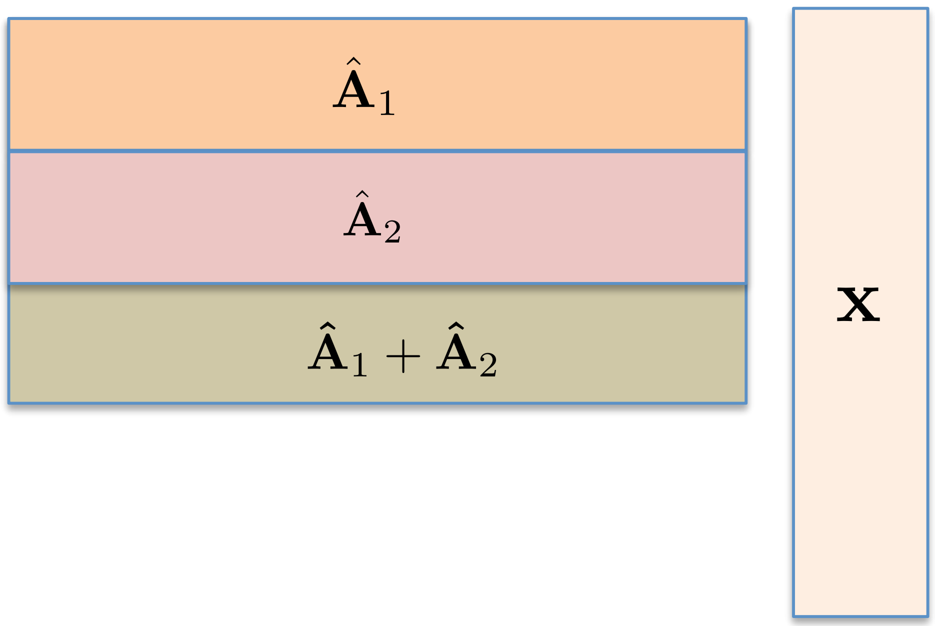
Algorithm-Based Fault Tolerance
[Huang, Abraham ’84]
[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran ’16]



A coding alternative to replication: MDS compute codes (“ABFT”)

Algorithm-Based Fault Tolerance
[Huang, Abraham ’84]
[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran ’16]

A coding alternative to replication: MDS compute codes (“ABFT”)



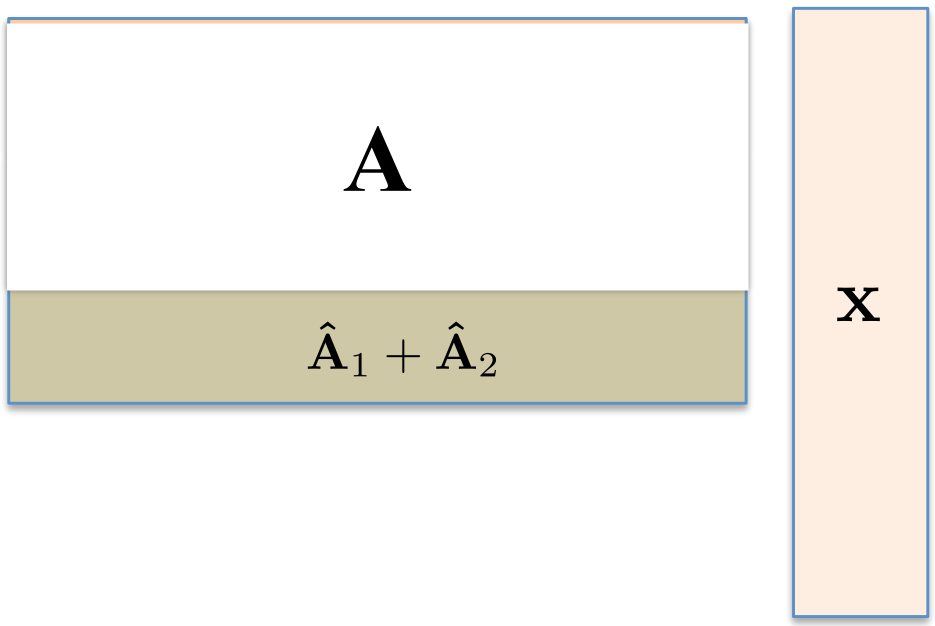
Algorithm-Based Fault Tolerance

[Huang, Abraham ’84]

[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran ’16]

Example: P=3, K=2

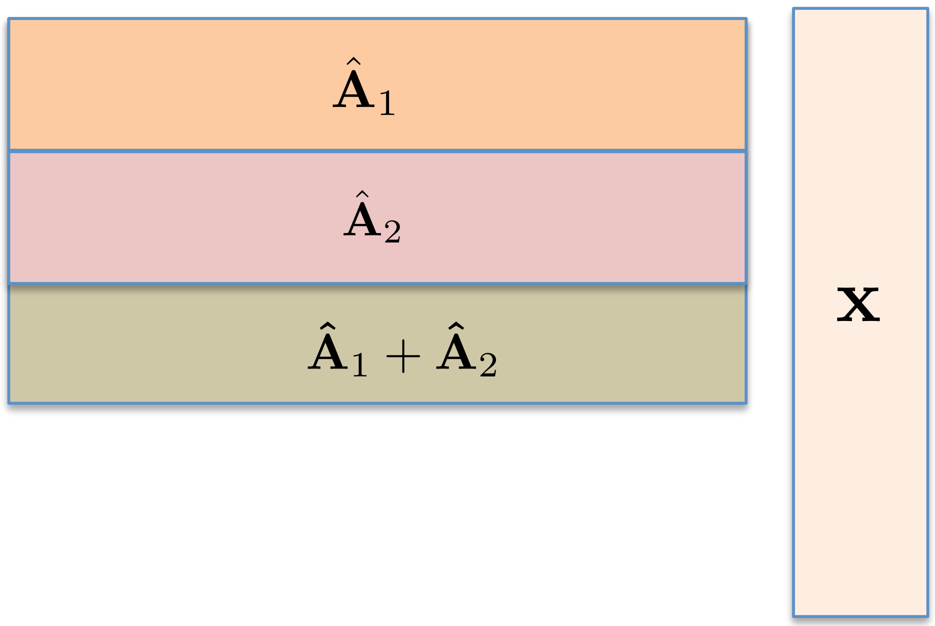
A coding alternative to replication: MDS compute codes (“ABFT”)



Algorithm-Based Fault Tolerance
[Huang, Abraham ’84]
[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran ’16]

Example: P=3, K=2

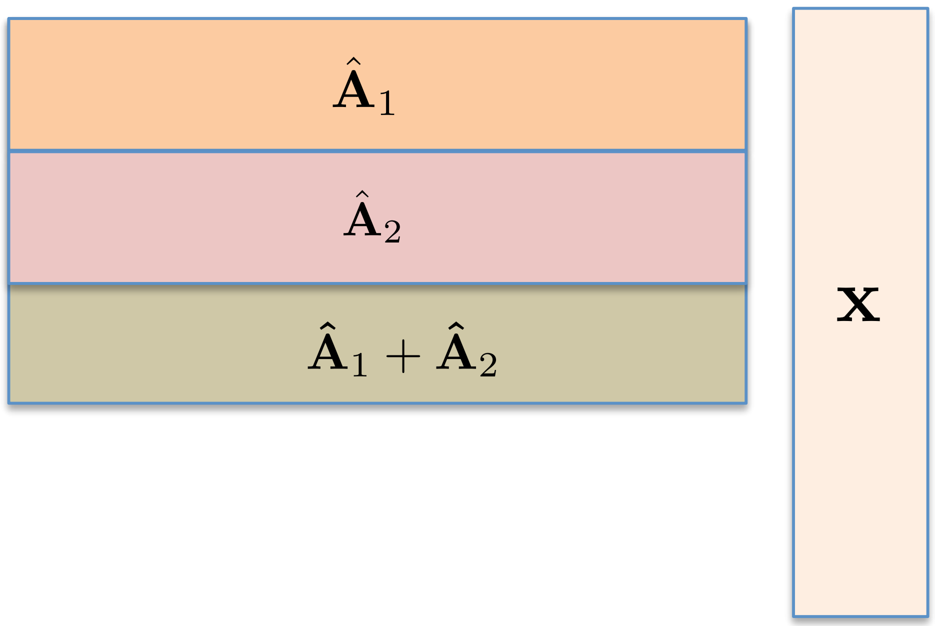
A coding alternative to replication: MDS compute codes (“ABFT”)



Algorithm-Based Fault Tolerance
[Huang, Abraham ’84]
[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran ’16]

Example: P=3, K=2

A coding alternative to replication: MDS compute codes (“ABFT”)



Algorithm-Based Fault Tolerance

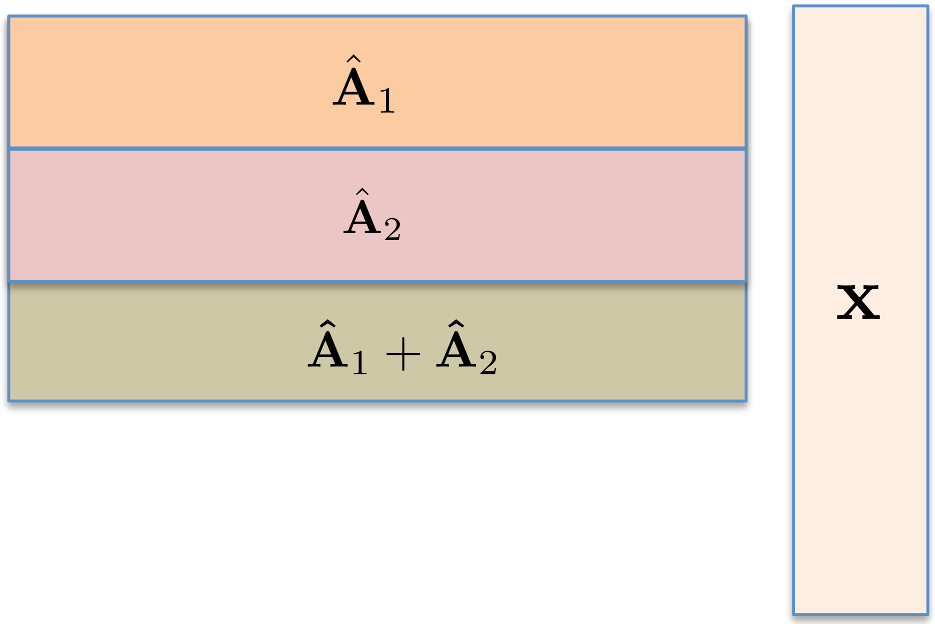
[Huang, Abraham ’84]

[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran ’16]

Example: P=3, K=2

Assumption: $\mathbf{A}$ known in advance

A coding alternative to replication: MDS compute codes ("ABFT")



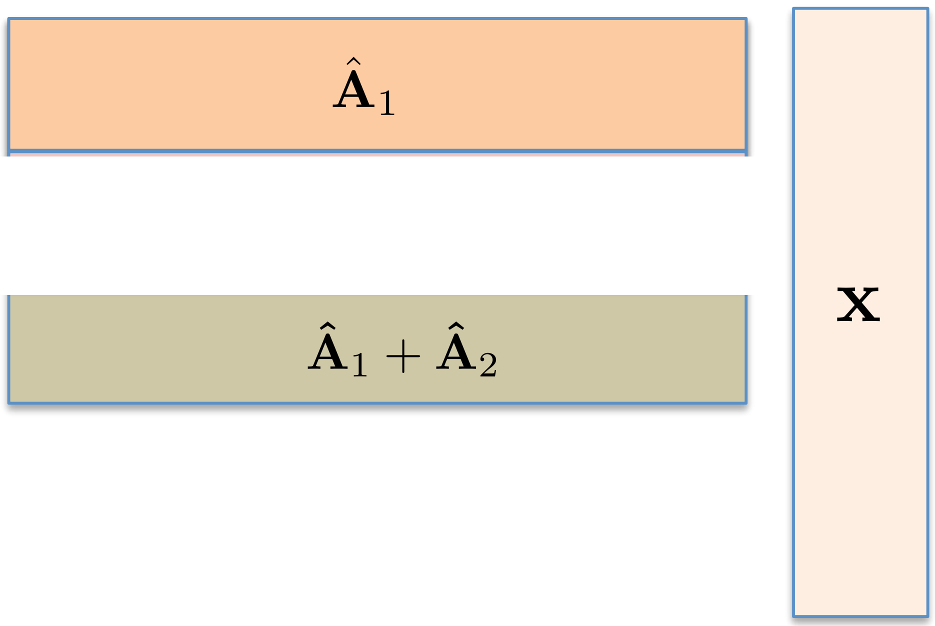
Algorithm-Based Fault Tolerance
[Huang, Abraham '84]
[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran '16]

Example: P=3, K=2

Assumption: $\mathbf{A}$ known in advance

Can tolerate 1 straggler
operations per processor = $MN/2$

A coding alternative to replication: MDS compute codes ("ABFT")



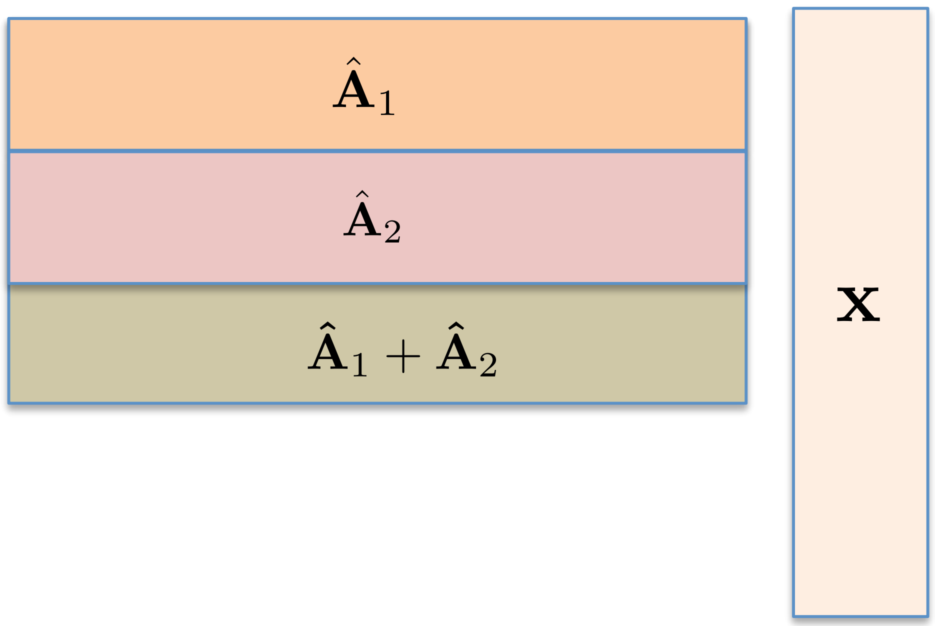
Algorithm-Based Fault Tolerance
[Huang, Abraham '84]
[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran '16]

Example: P=3, K=2

Assumption: $\mathbf{A}$ known in advance

Can tolerate 1 straggler
operations per processor = $MN/2$

A coding alternative to replication: MDS compute codes ("ABFT")



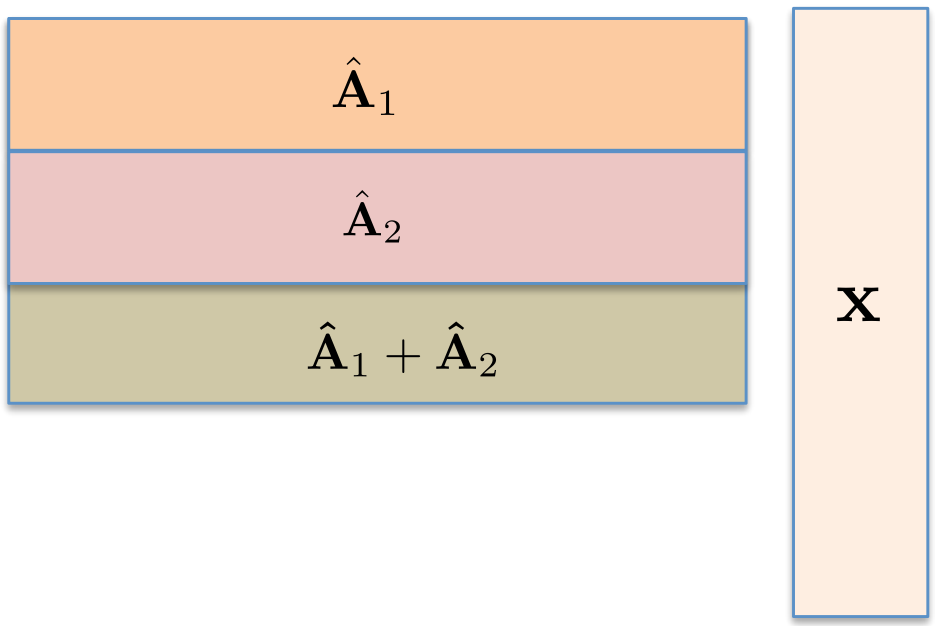
Algorithm-Based Fault Tolerance
[Huang, Abraham '84]
[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran '16]

Example: P=3, K=2

Assumption: $\mathbf{A}$ known in advance

Can tolerate 1 straggler
operations per processor = $MN/2$

A coding alternative to replication: MDS compute codes (“ABFT”)



Algorithm-Based Fault Tolerance
[Huang, Abraham ’84]
[Lee, Lam, Pedarsani, Papailopoulos, Ramchandran ’16]

Example: P=3, K=2

Assumption: $\mathbf{A}$ known in advance

Can tolerate 1 straggler
\# operations per processor = $MN/2$

P processors
In general, use a (P,K)-MDS code ($K < M$):
Recovery Threshold = K, *i.e.*, Straggler tolerance = P-K
\# operations/processor = MN/K (> MN/P in uncoded)

MDS coded computing of M x V outperforms replication

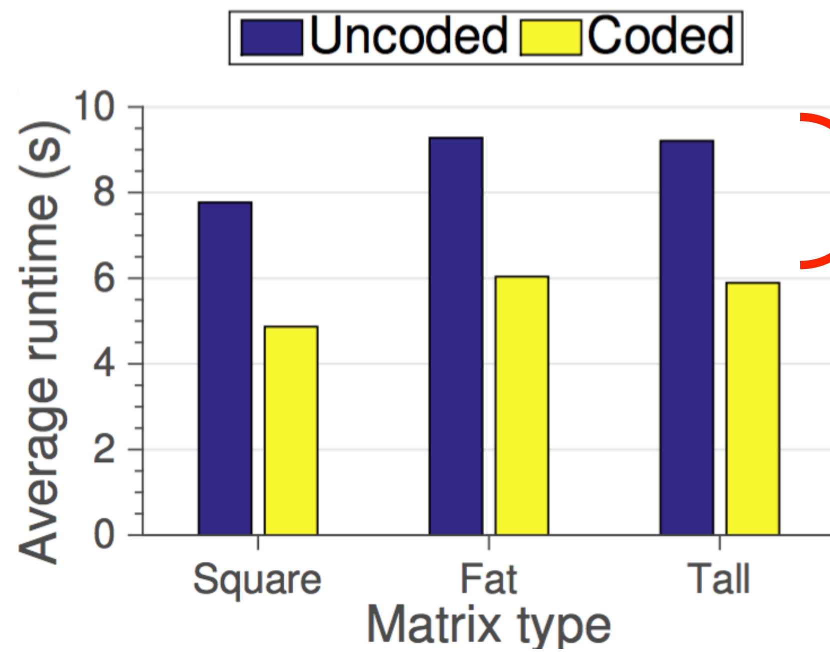
MDS coded computing of M x V outperforms replication

[Lee et al]: MDS beats replication in expected time (exponential tail models)

MDS coded computing of M x V outperforms replication

[Lee et al]: MDS beats replication in expected time (exponential tail models)

Experiments on AmazonEC2:
[Lee at al]



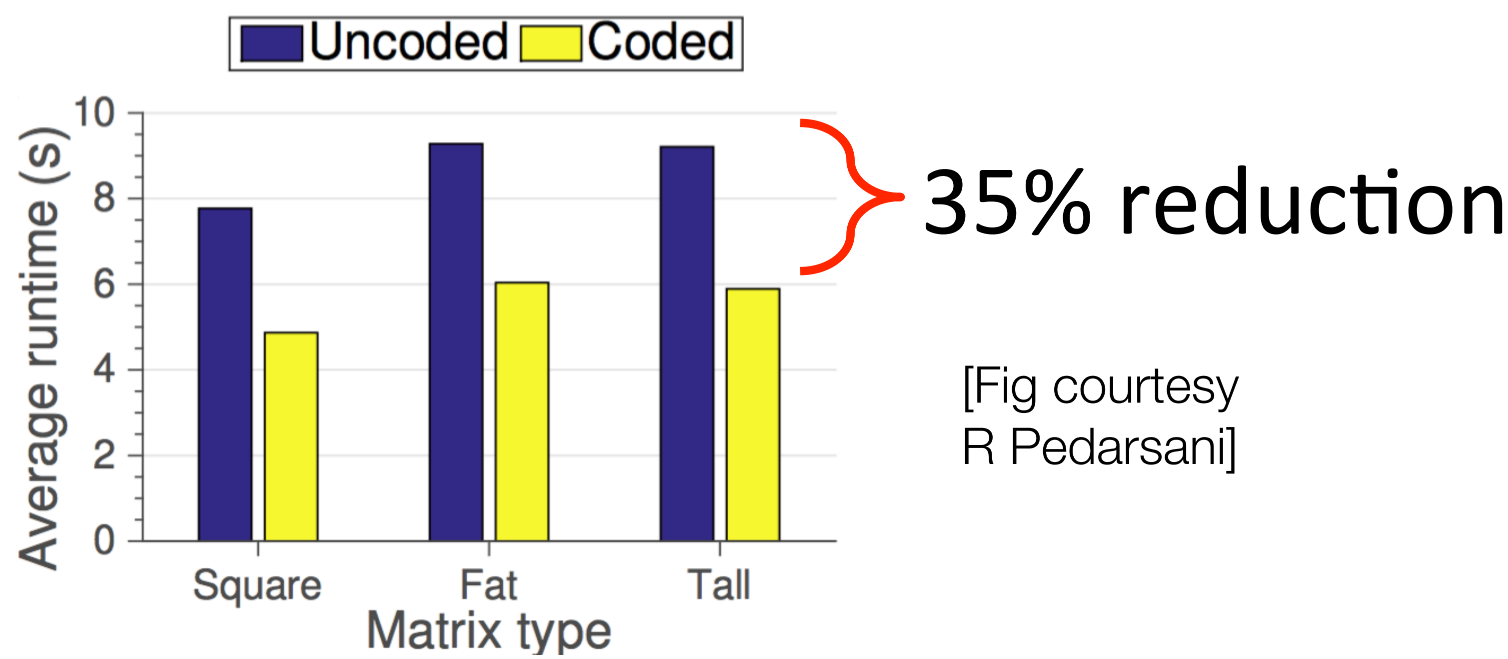
35% reduction

[Fig courtesy
R Pedarsani]

MDS coded computing of M x V outperforms replication

[Lee et al]: MDS beats replication in expected time (exponential tail models)

Experiments on AmazonEC2:
[Lee at al]

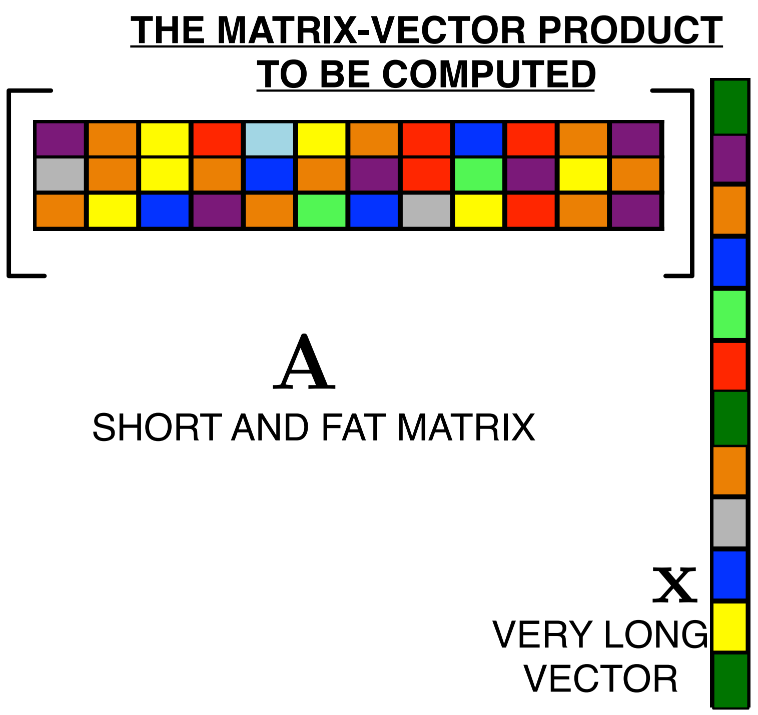


[Fig courtesy
R Pedarsani]

Can tradeoff # operations/processor for straggler tolerance
Codes for # operations/processor < N ?

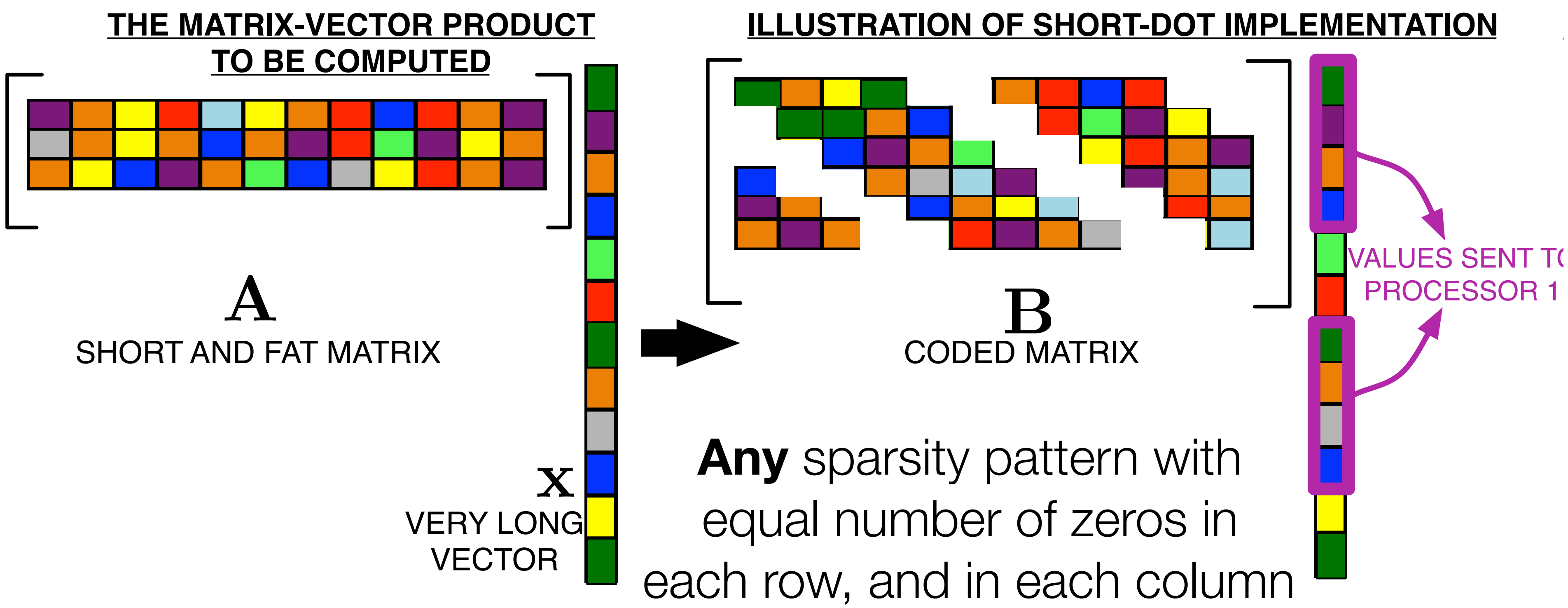
Short-Dot codes

[Dutta, Cadambe, Grover '16]
[Tandon, Lei, Dimakis, Karampatziakis '16]



Short-Dot codes

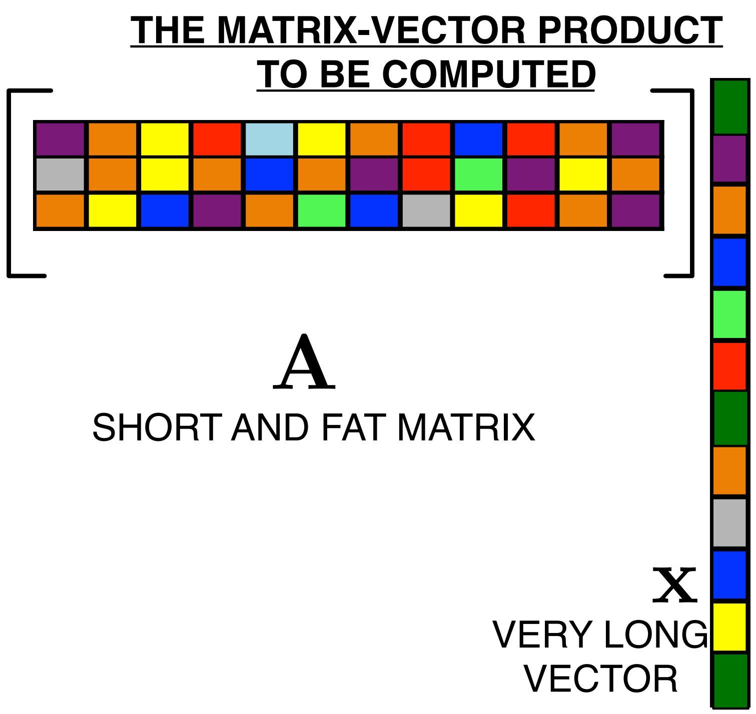
[Dutta, Cadambe, Grover '16]
[Tandon, Lei, Dimakis, Karampatziakis '16]

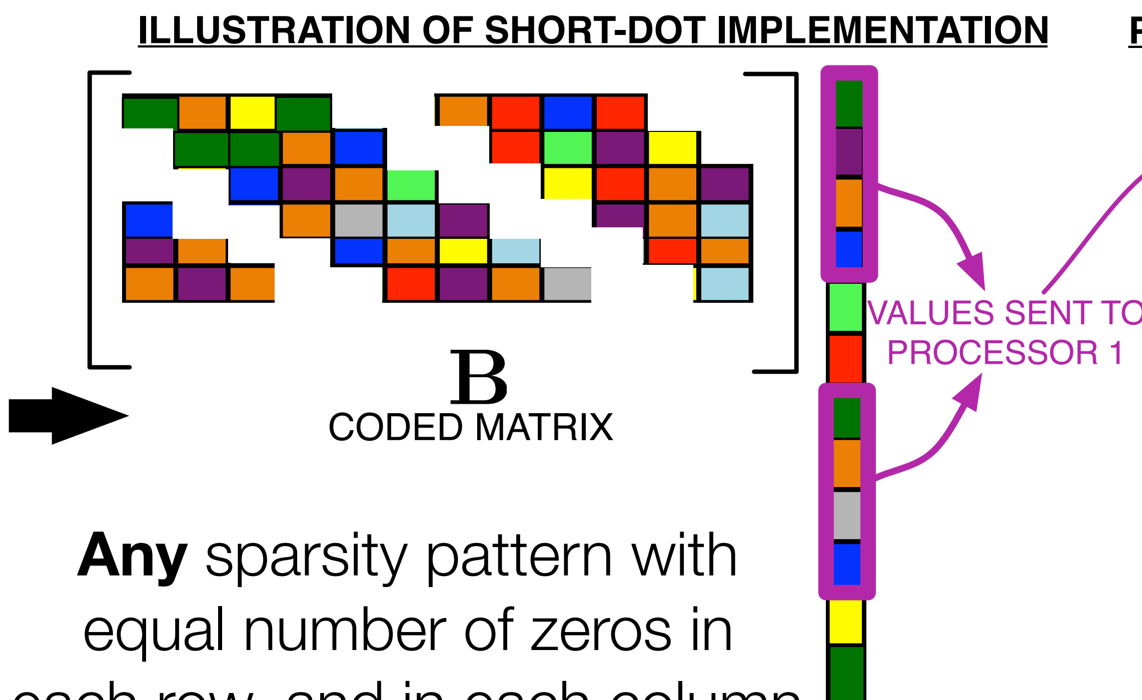


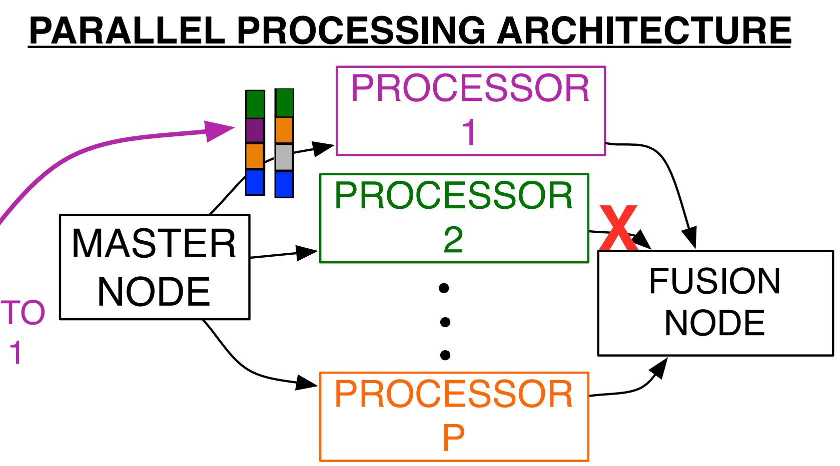
Any sparsity pattern with equal number of zeros in each row, and in each column

Short-Dot codes

[Dutta, Cadambe, Grover ’16]
[Tandon, Lei, Dimakis, Karampatziakis ‘16]

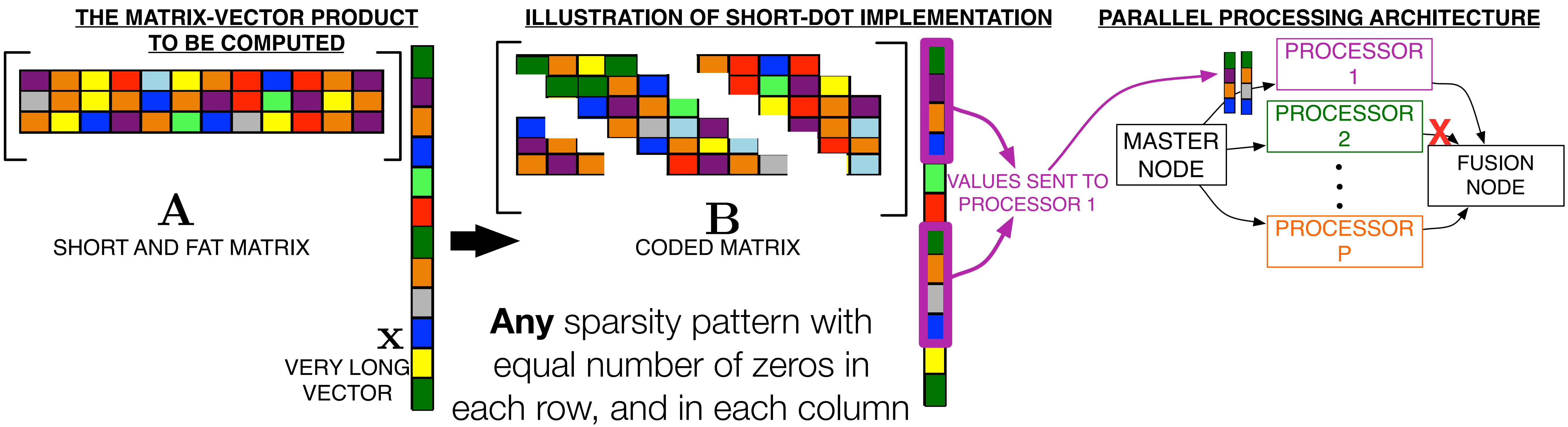






Any sparsity pattern with equal number of zeros in each row, and in each column

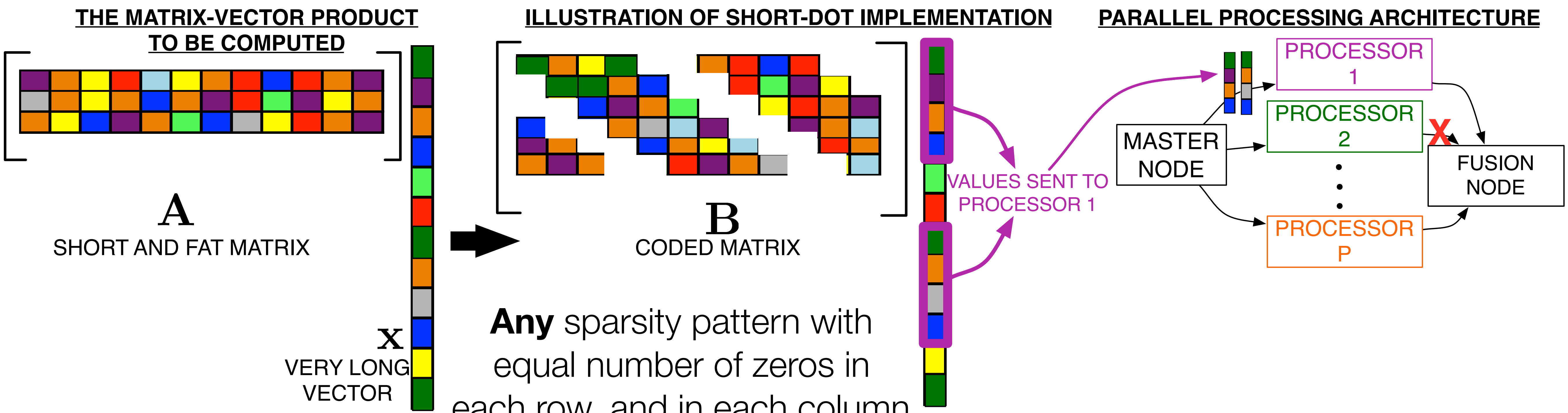
Short-Dot codes [Dutta, Cadambe, Grover '16] [Tandon, Lei, Dimakis, Karampatziakis '16]



Sparsity
(i) allows tradeoff between computation per-processor and straggler tolerance;
(ii) reduces communication to each processor

Short-Dot codes

[Dutta, Cadambe, Grover '16]
[Tandon, Lei, Dimakis, Karampatziakis '16]



Any sparsity pattern with equal number of zeros in each row, and in each column

Sparsity
(i) allows tradeoff between computation per-processor and straggler tolerance;
(ii) reduces communication to each processor

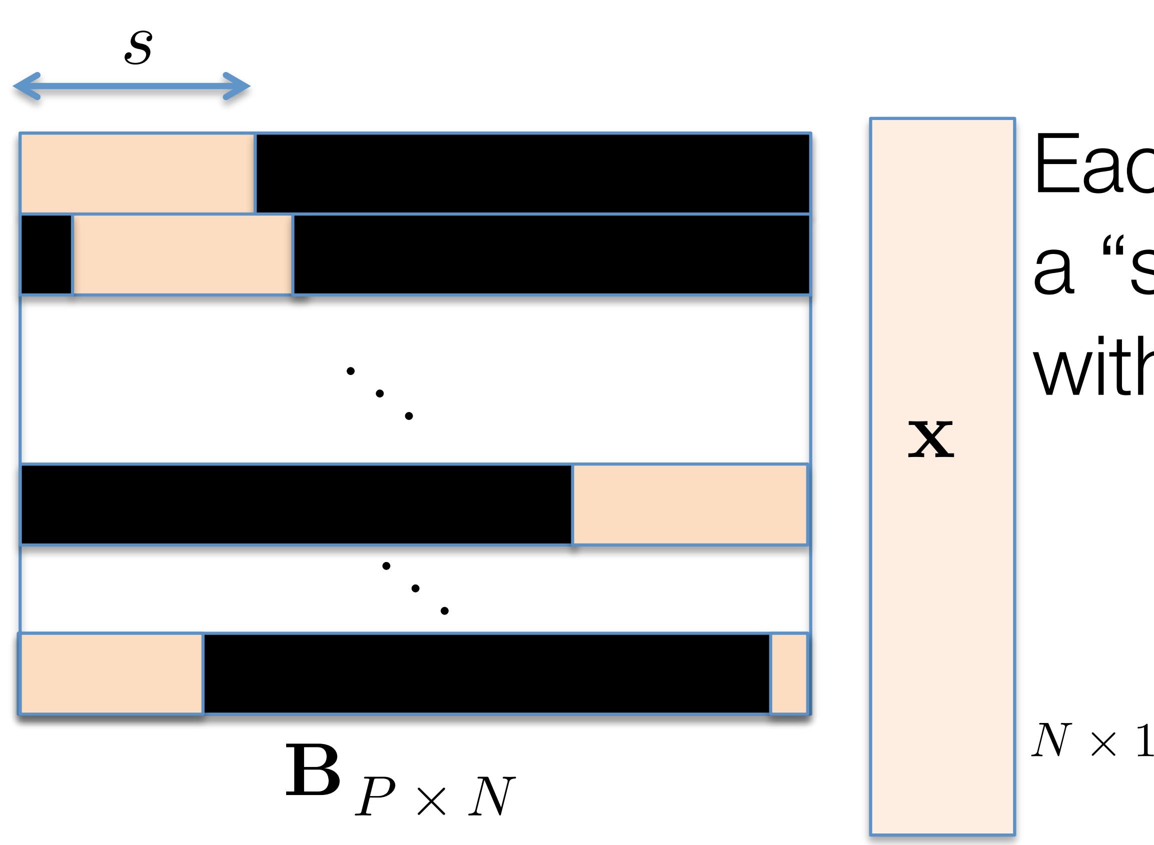
\# operations/processor = $s < N$
Recovery threshold = $K = P(1\text{-}s/N)+M$

Short-Dot codes: the construction

Given $\mathbf{A}$, an M x N matrix, $M < P$, and a parameter K, $M < K < P$, an (s,K) Short-Dot code consists of a P x N matrix $\mathbf{B}$ satisfying:

1) $\mathbf{A}$ is contained in span of any K rows of $\mathbf{B}$

2) Every row of $\mathbf{B}$ is s-sparse



Each processor computes a "short" dot product of $\mathbf{x}$ with one row of $\mathbf{B}$

"Short-Dot": Computing Large Linear Transforms Distributedly Using Coded Short Dot Products [Dutta, Cadambe, Grover, NIPS 2016]

Achievability and outer bound

Achievability: For any M x N matrix $\mathbf{A}$, an (s, K) Short-Dot code exists s.t.:

$$s \leq \frac{N}{P}(P - K + M)$$

…and outputs of any K processors suffice, i.e., Straggler tolerance = P-K

Proof overviews in appendices of this talk

Achievability and outer bound

Achievability: For any M x N matrix $\mathbf{A}$, an (s, K) Short-Dot code exists s.t.:

$$s \leq \frac{N}{P}(P - K + M)$$

…and outputs of any K processors suffice, i.e., Straggler tolerance = P-K

Outer bound: Any Short-Dot code satisfies:

$$\bar{s} \geq \frac{N}{P}(P - K + M) - \frac{M^2}{P}\binom{P}{K - M + 1}$$

… for "sufficiently dense" $\mathbf{A}$

Proof overviews in appendices of this talk

Achievability and outer bound

Achievability: For any M x N matrix $\mathbf{A}$, an (s, K) Short-Dot code exists s.t.:

$$s \leq \frac{N}{P}(P - K + M)$$

…and outputs of any K processors suffice, i.e., Straggler tolerance = P-K

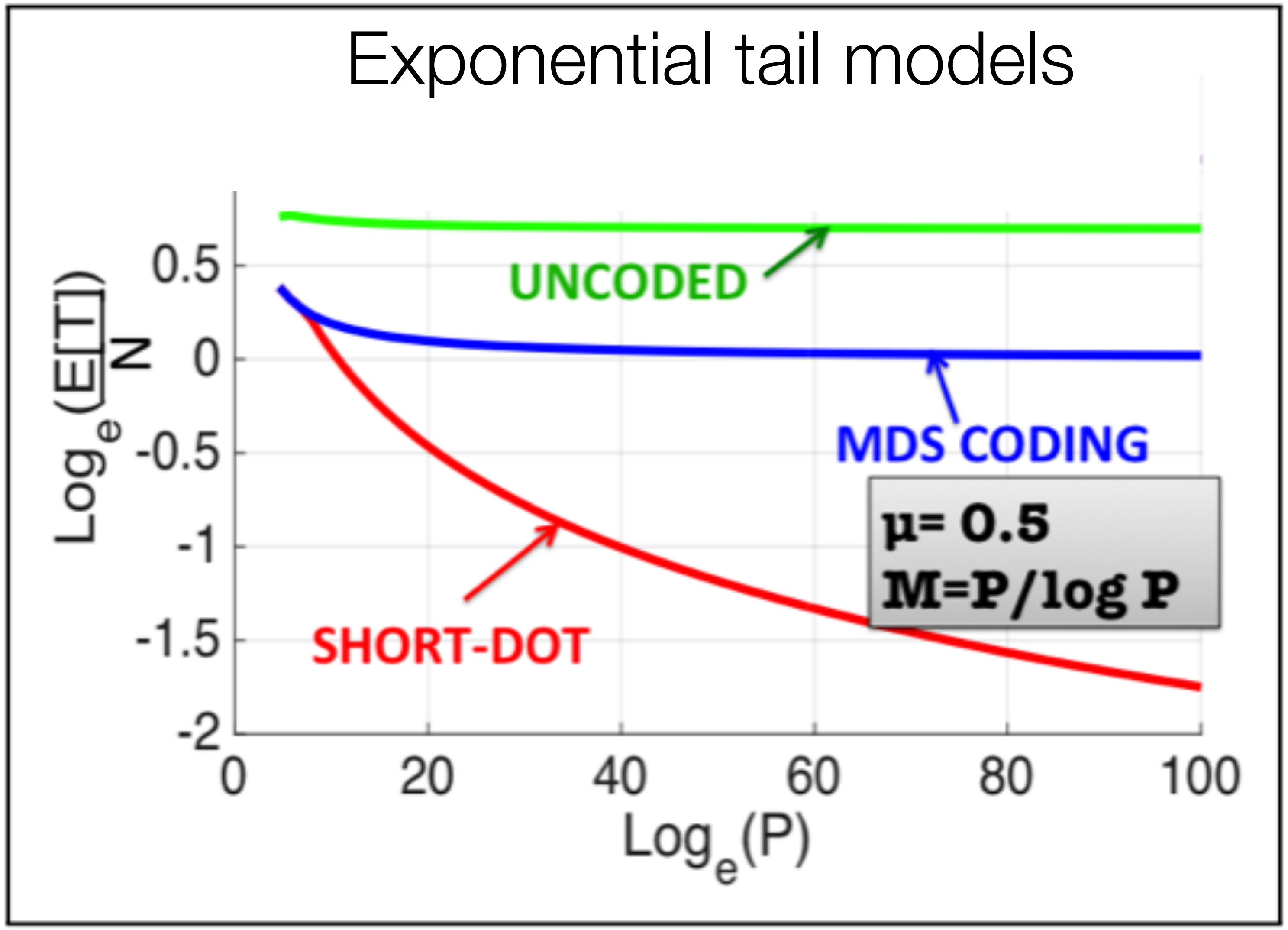
Outer bound: Any Short-Dot code satisfies:

$$\bar{s} \geq \frac{N}{P}(P - K + M) - o(N)$$

… for "sufficiently dense" $\mathbf{A}$

Proof overviews in appendices of this talk

Short-Dot strictly and significantly outperforms Uncoded/Replication/ABFT (MDS)

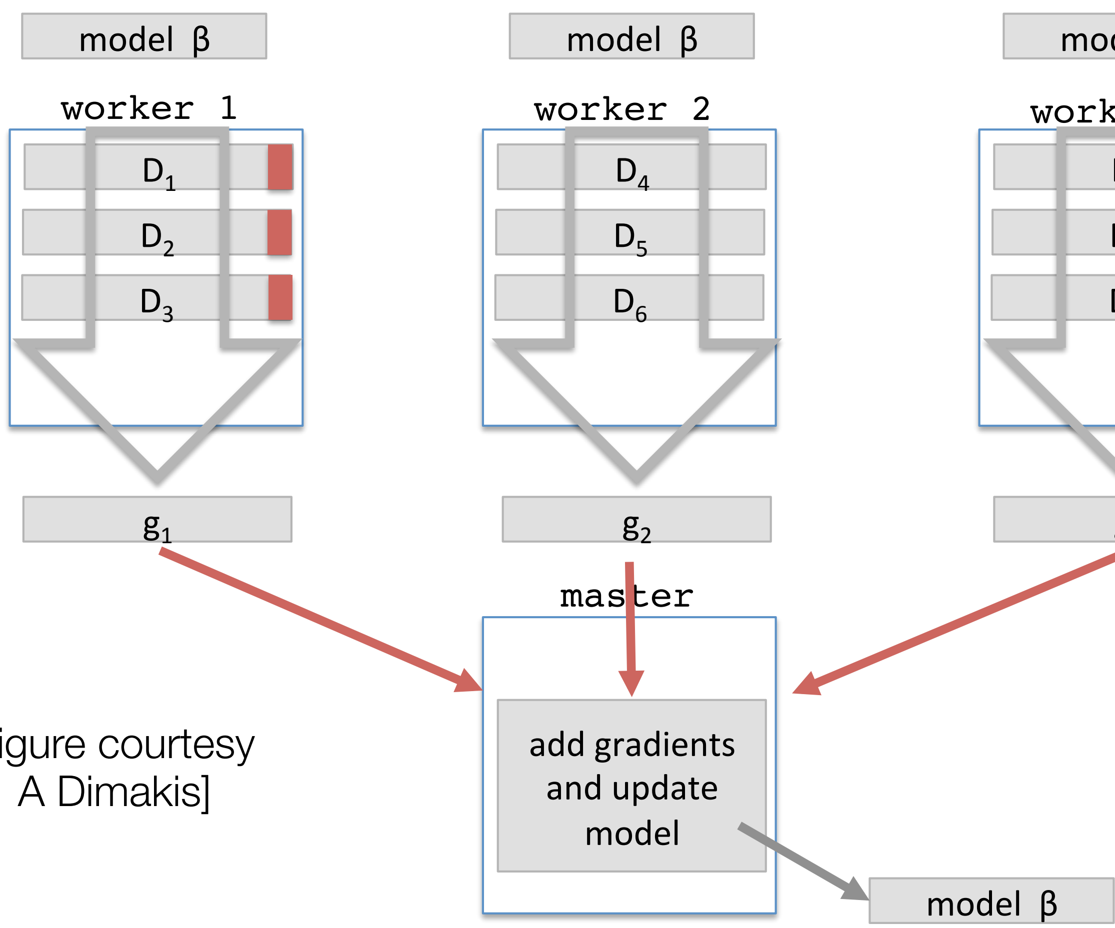


Paper contains expected completion time analysis for exponential service time model, and experimental results.

For $N >> M$, decoding complexity negligible compared to per-processor computation

Related result: Gradient coding

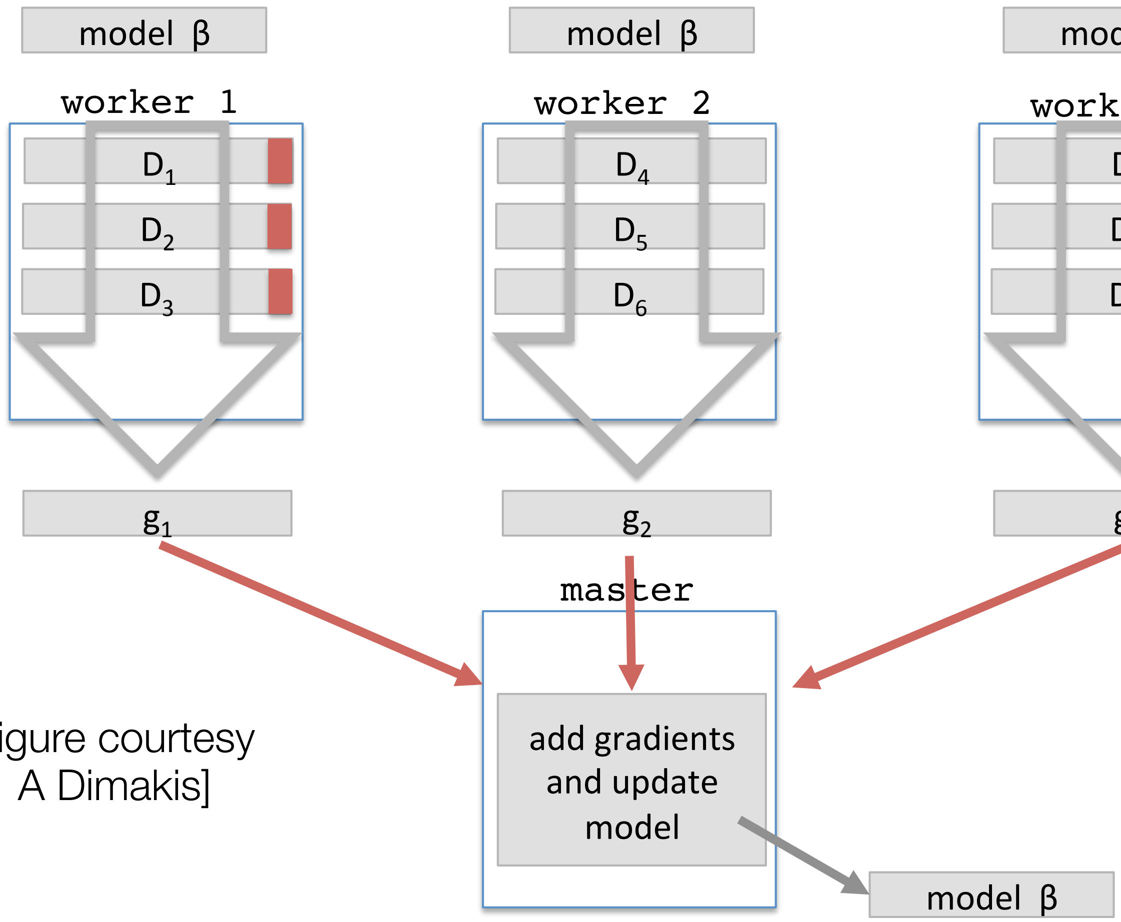
[Tandon, Lei, Dimakis, Karampatziakis'17]



[Figure courtesy A Dimakis]

What if some gradient-computing workers straggle?

Related result: Gradient coding



[Figure courtesy A Dimakis]

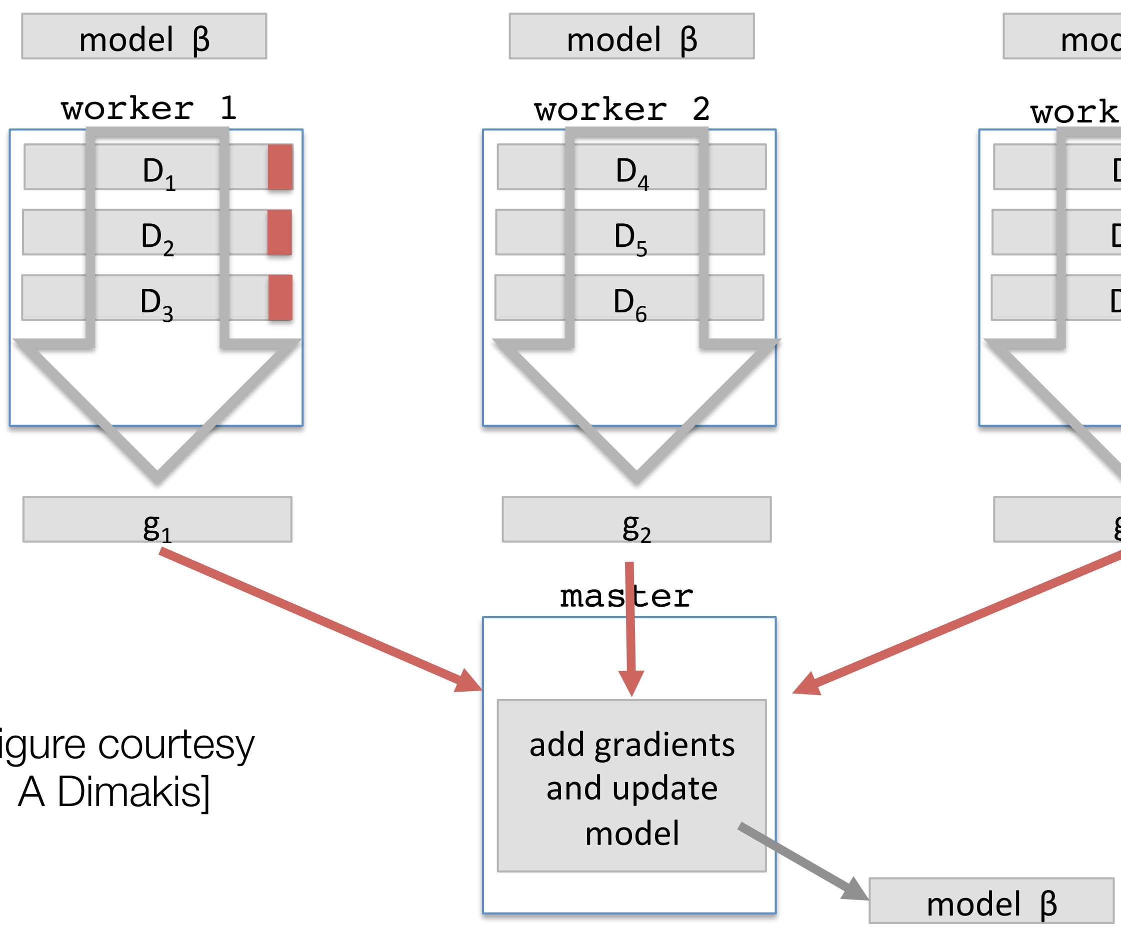
[Tandon, Lei, Dimakis, Karampatziakis'17]

Want to compute:

$$\sum_i g_i$$

What if some gradient-computing workers straggle?

Related result: Gradient coding



[Figure courtesy A Dimakis]

[Tandon, Lei, Dimakis, Karampatziakis'17]

Want to compute:

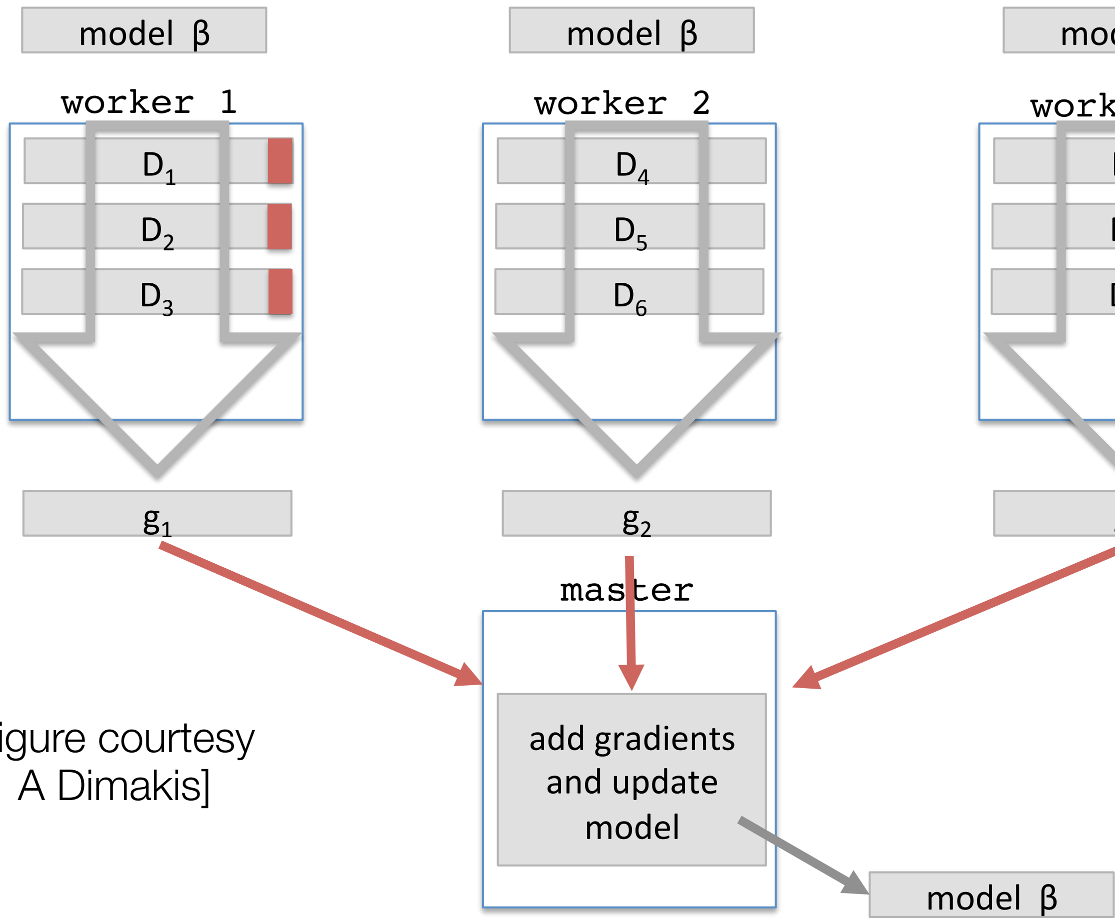
$$\sum_i g_i = [1, 1, \ldots, 1] \begin{bmatrix} g_1 \\ g_2 \\ \cdot \\ \cdot \\ g_N \end{bmatrix}$$

known "matrix"

vector computed distributedly

What if some gradient-computing workers straggle?

Related result: Gradient coding



[Figure courtesy A Dimakis]

[Tandon, Lei, Dimakis, Karampatziakis'17]

Want to compute:

$$\sum_i g_i = [1, 1, \ldots, 1] \begin{bmatrix} g_1 \\ g_2 \\ \cdot \\ \cdot \\ g_N \end{bmatrix}$$

known "matrix"

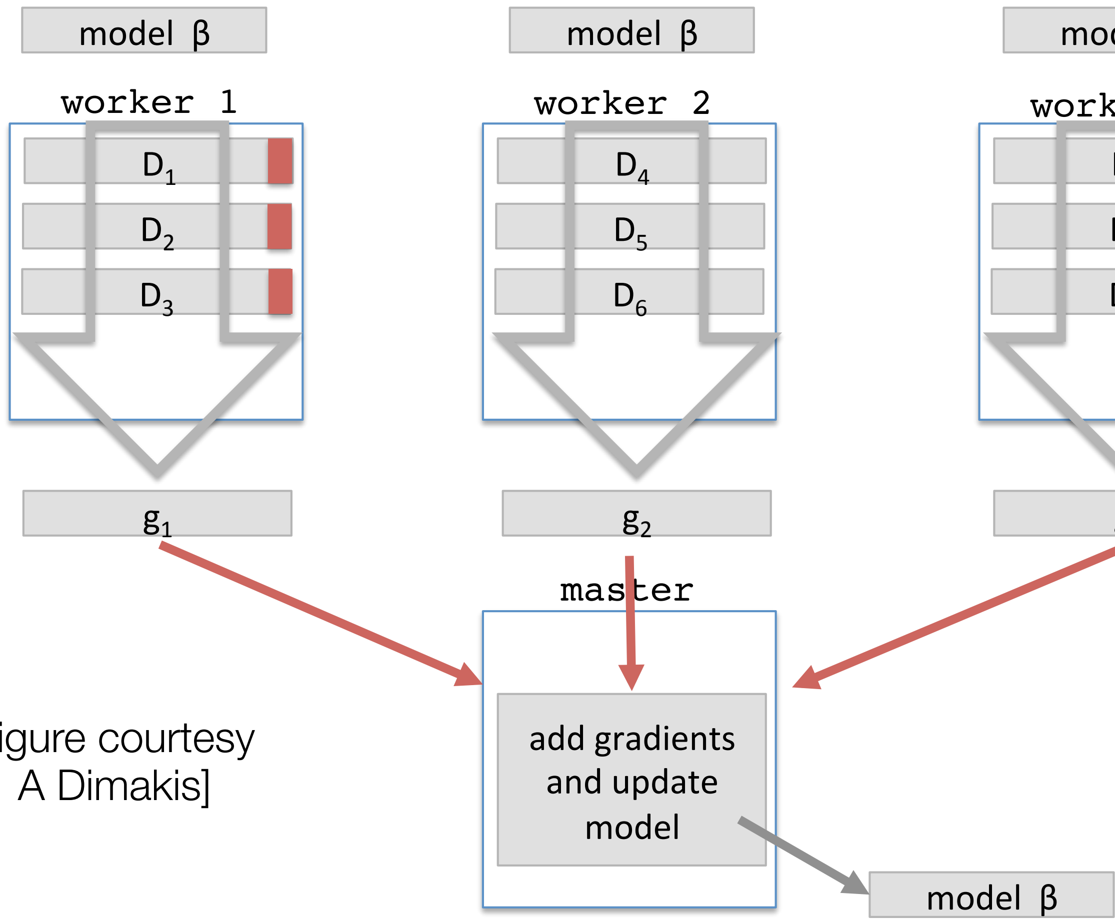
vector computed distributedly

What if some gradient-computing workers straggle?

Solution: code "matrix" $\mathbf{A}$ (i.e., [1 1 … 1]) using a Short-Dot code
- introduce redundancy in datasets consistent with the Short-Dot pattern
- computes the correct (redundant) gradients at each processor

Can also be viewed as a novel "distributed storage code for computation"

Related result: Gradient coding



[Figure courtesy A Dimakis]

[Tandon, Lei, Dimakis, Karampatziakis'17]

Want to compute:

$$\sum_i g_i = [1, 1, \ldots, 1] \begin{bmatrix} g_1 \\ g_2 \\ \cdot \\ \cdot \\ g_N \end{bmatrix}$$

known "matrix"

vector computed distributedly

What if some gradient-computing workers straggle?

Solution: code "matrix" $\mathbf{A}$ (i.e., [1 1 … 1]) using a Short-Dot code

- introduce redundancy in datasets consistent with the Short-Dot pattern
- computes the correct (redundant) gradients at each processor

Can also be viewed as a novel "distributed storage code for computation"

For $\mathbf{V}^{\mathrm{T}}\,\mathbf{V}$, coding can beat replication only due to integer effects.

No scaling-sense gain, at least in this coarse model, over replication.

(See also [Halbawi, Azizan-Ruhi, Salehi, Hassibi '17])

Trend:

- V x V : offers some advantage over replication
- M x V: arbitrary gains over replication, MDS coding

Trend:

- V x V : offers some advantage over replication
- M x V: arbitrary gains over replication, MDS coding
- Next: M x M: ?

Trend:

- V x V : offers some advantage over replication
- M x V: arbitrary gains over replication, MDS coding
- Next: M x M: ?

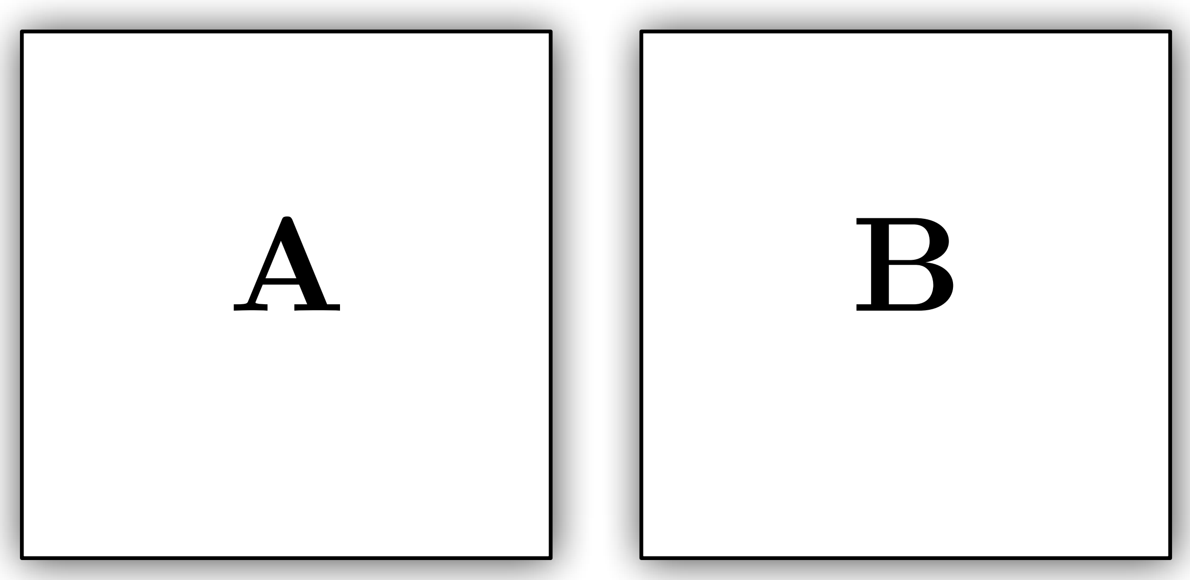
Answer: arbitrarily large gains over M x V-type coding!

Trend:

- V x V : offers some advantage over replication
- M x V: arbitrary gains over replication, MDS coding
- Next: M x M: ?

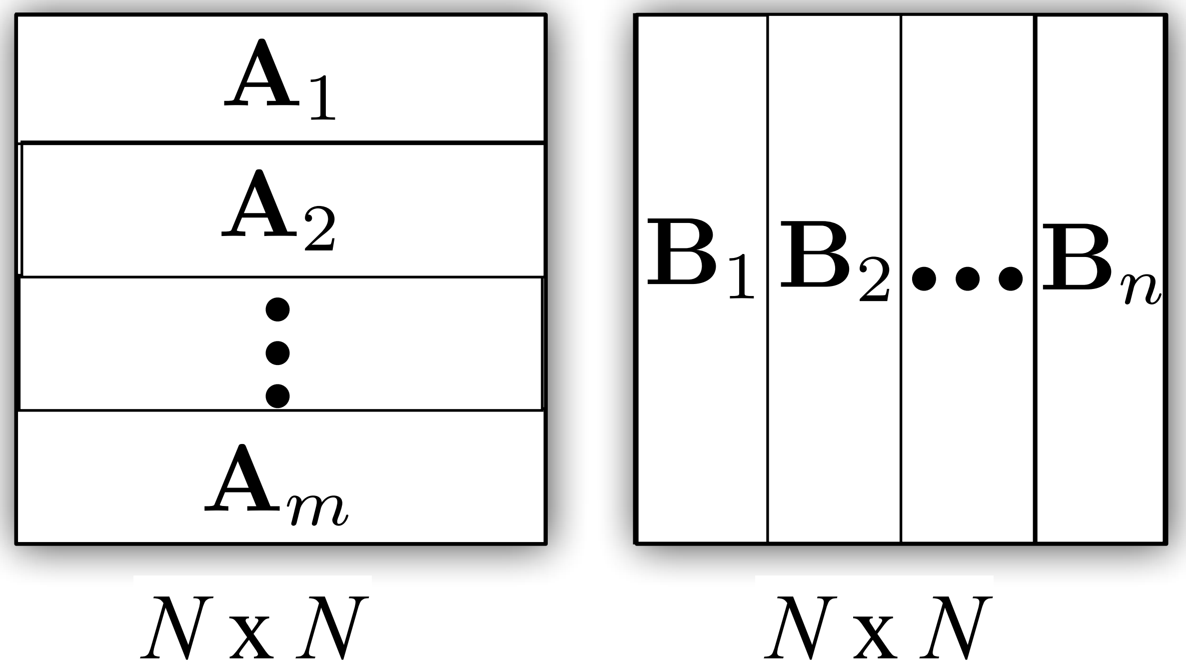
Answer: arbitrarily large gains over M x V-type coding!

break!


A
B

Uncoded parallelization

Let's assume that each processor can store 1/m of **A** and 1/n of **B**



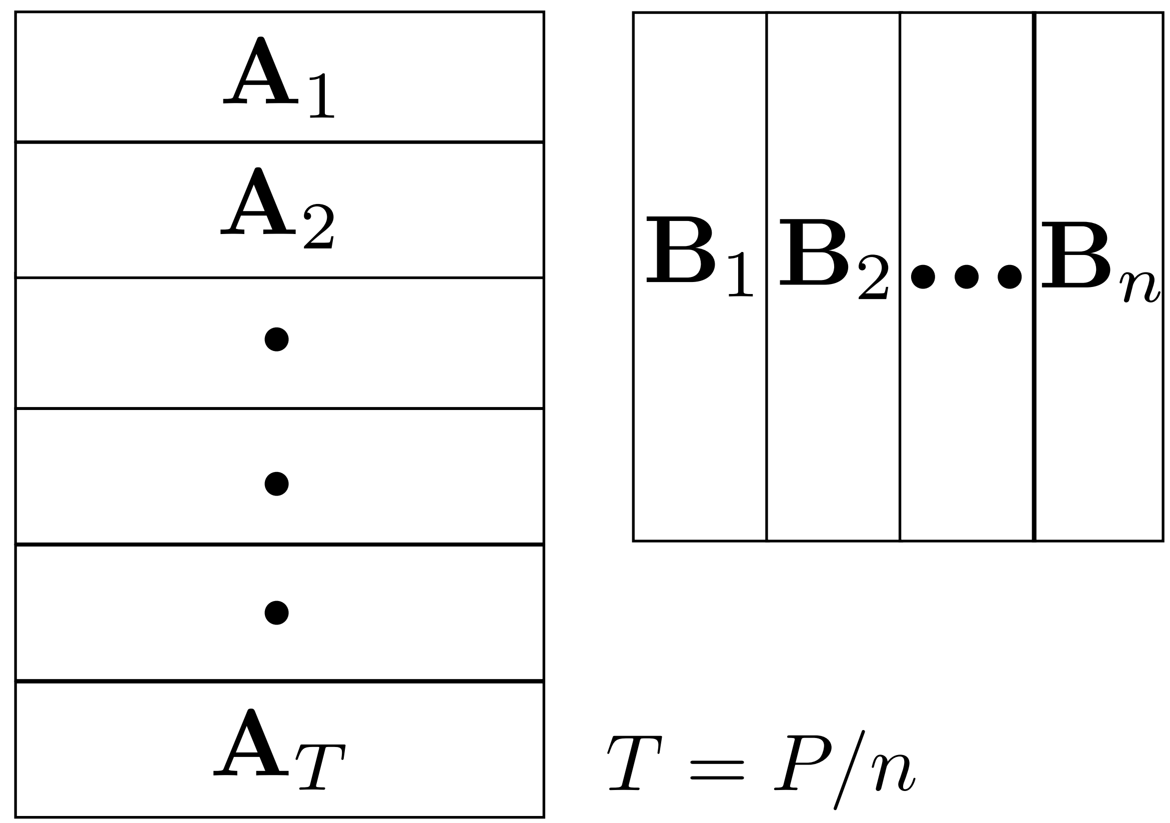
Total mn processors

(i,j)-th Processor receives $\mathbf{A_i}$, $\mathbf{B_j}$, computes $\mathbf{A_i} \times \mathbf{B_j}$, sends them to fusion center

operations/processor = N^3/mn *(we'll keep this constant across strategies)*

Recovery Threshold = P; Straggler tolerance = 0

Strategy I: M x V → M x M



$T = P/n$

Each processor computes a product $\mathbf{A_i}\,\mathbf{B_j}$

Recovery threshold $= P - P/n + m = \Theta(P)$

\# operations/processor: N^3/mn

Algorithm-based Fault Tolerance (ABFT)

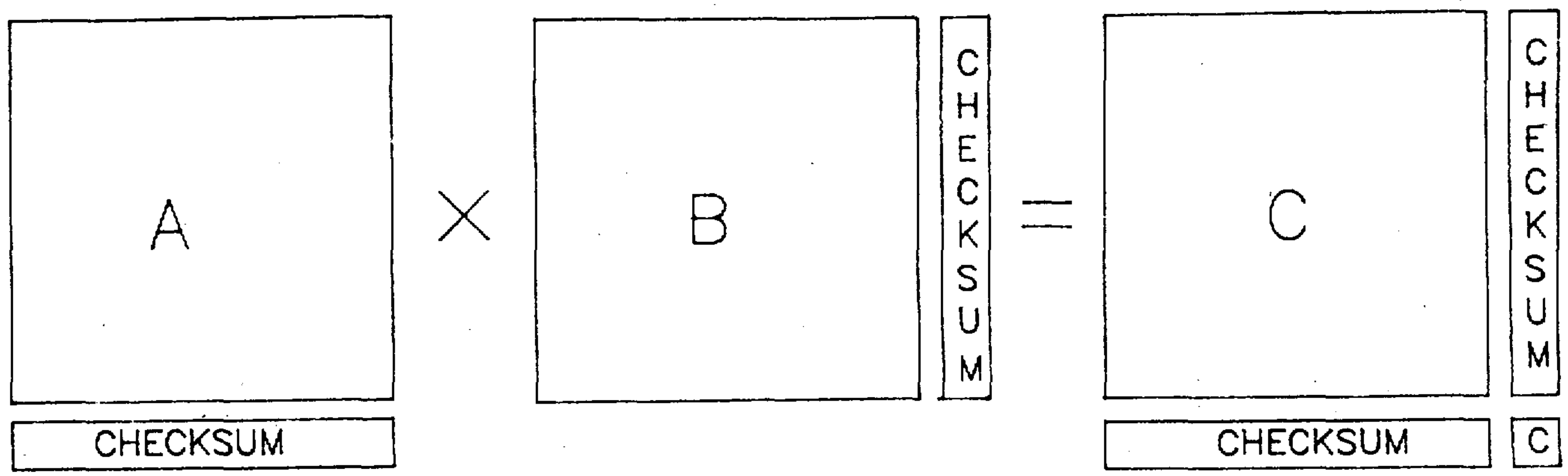


Fig. 1. A checksum matrix multiplication.

[Huang, Abraham’84]
[Lee, Suh, Ramchandran’17]

Algorithm-based Fault Tolerance (ABFT)

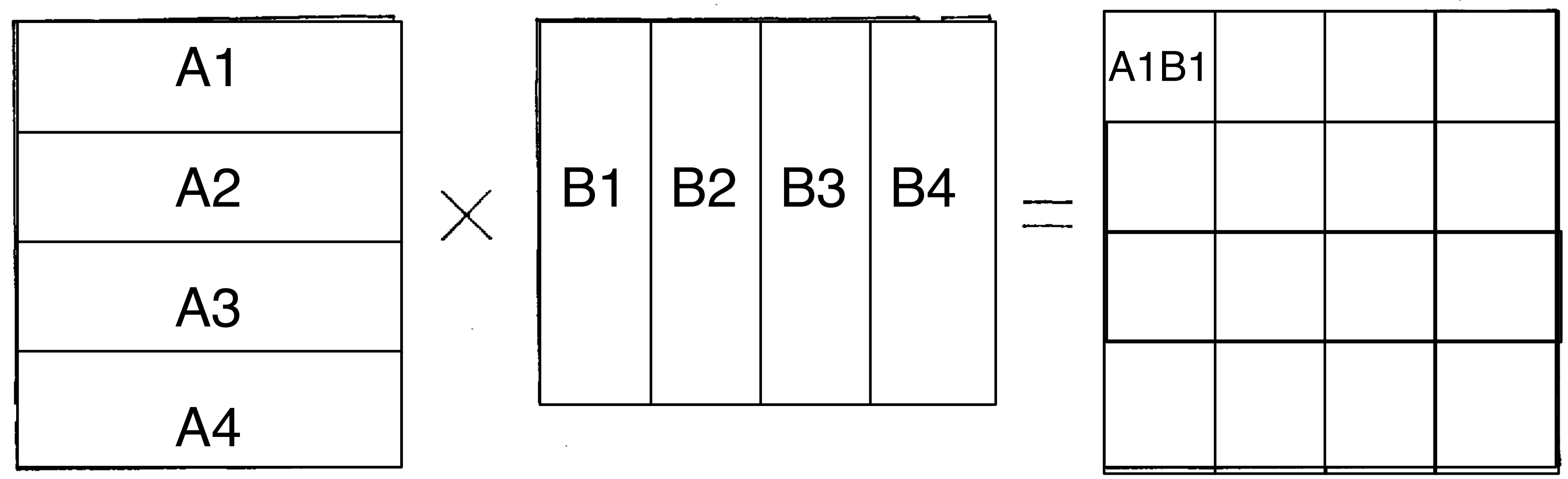


Fig. 1. A checksum matrix multiplication.

[Huang, Abraham'84]
[Lee, Suh, Ramchandran'17]

Algorithm-based Fault Tolerance (ABFT)

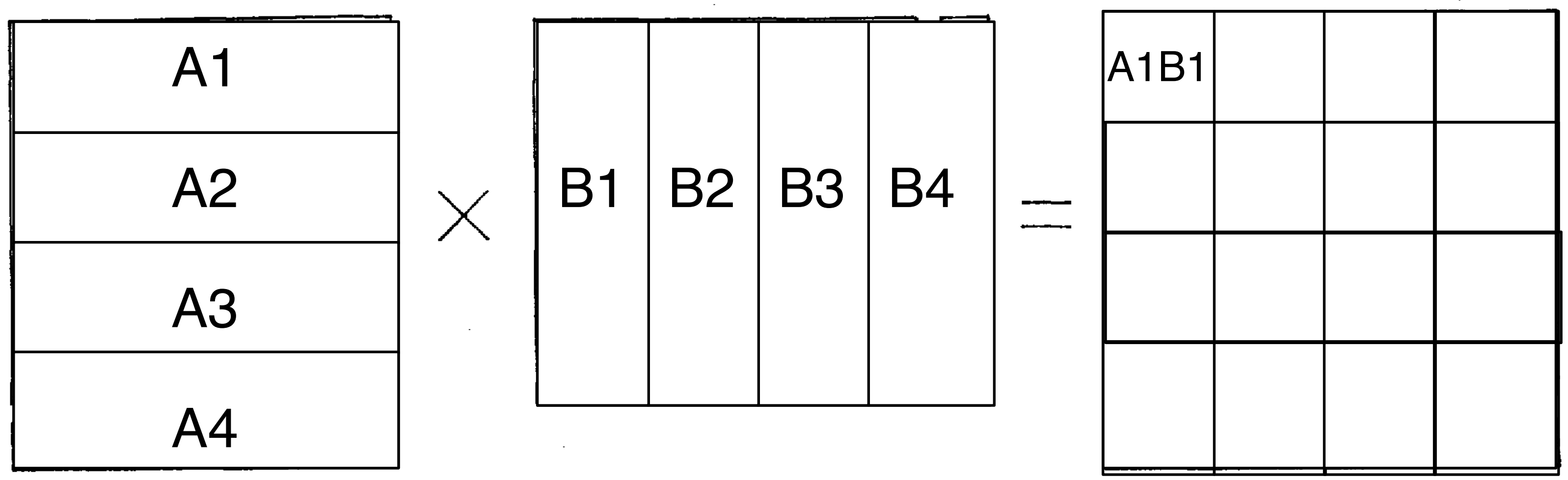


Fig. 1. A checksum matrix multiplication.

[Huang, Abraham'84]
[Lee, Suh, Ramchandran'17]

Recovery threshold: $K = 2(m-1)\sqrt{P} - (m-1)^2 + 1 = \Theta(\sqrt{P})$
Straggler resilience: $P - K$ [Lee, Suh, Ramchandran'17]
\# operations/processor: N^3/mn

Algorithm-based Fault Tolerance (ABFT)

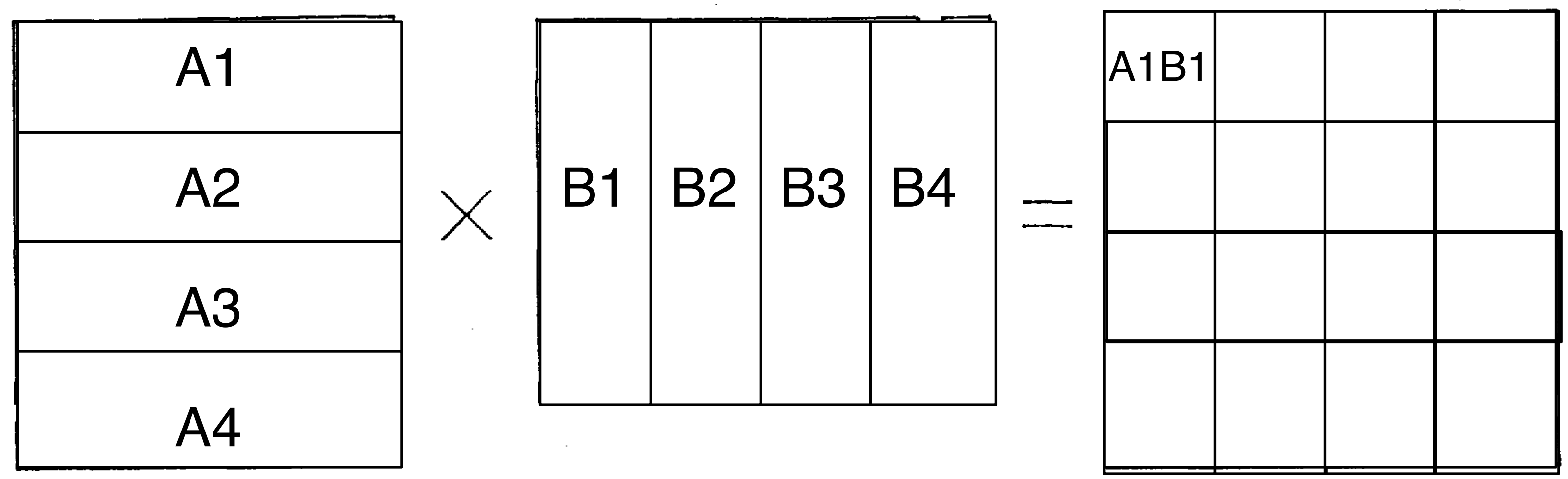


Fig. 1. A checksum matrix multiplication.

[Huang, Abraham'84]
[Lee, Suh, Ramchandran'17]

Recovery threshold: $K = 2(m-1)\sqrt{P} - (m-1)^2 + 1 = \Theta(\sqrt{P})$
Straggler resilience: $P - K$ [Lee, Suh, Ramchandran'17]
operations/processor: N^3/mn

Next: Polynomial codes [Yu, Maddah-Ali, Avestimehr '17]

Recovery threshold: $K = mn$
operations/processor: N^3/mn

Polynomial codes [Yu, Maddah-Ali, Avestimehr ’17]

Intuition: forget matrices for this slide

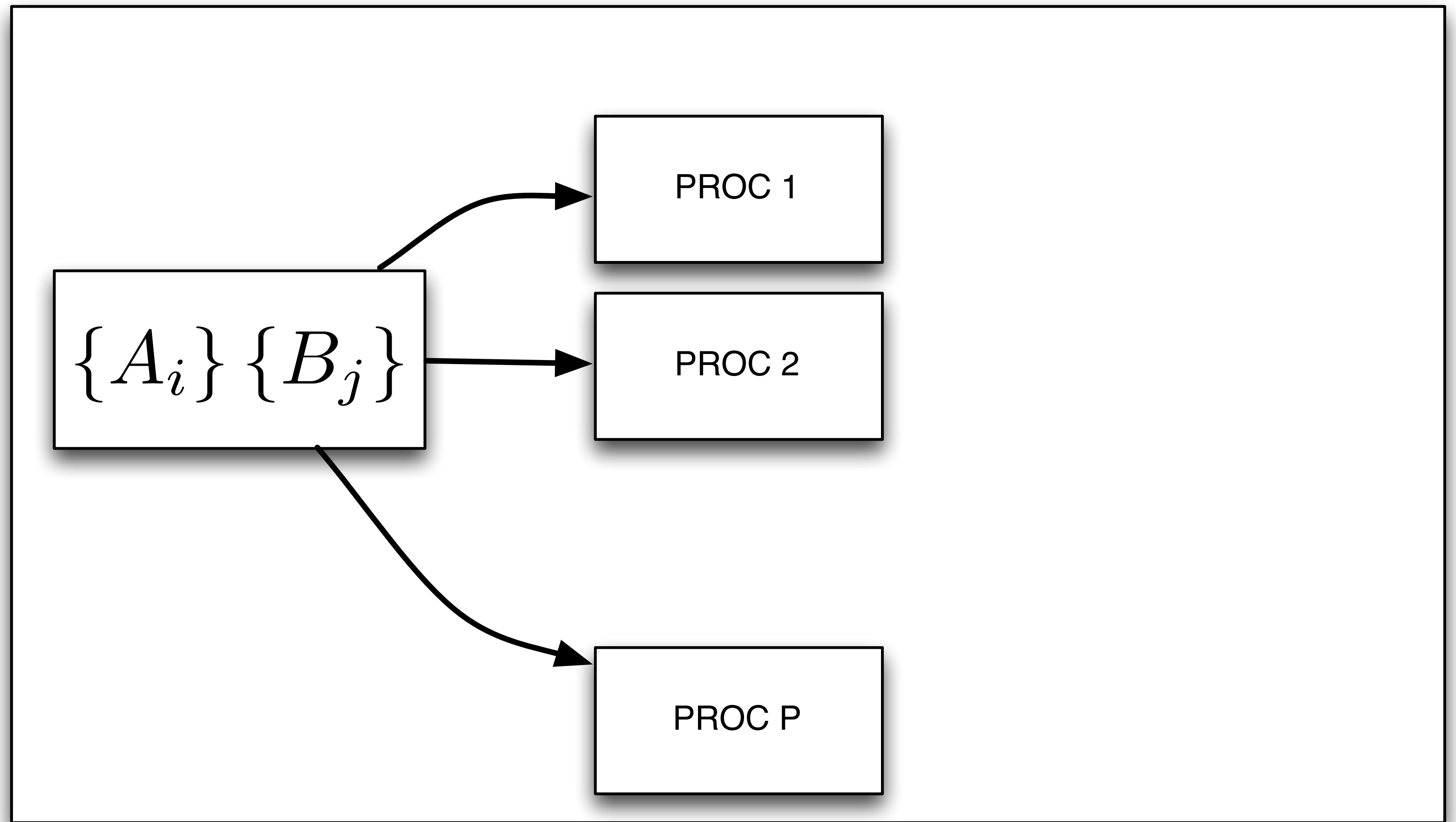
Polynomial codes [Yu, Maddah-Ali, Avestimehr ’17]

Intuition: forget matrices for this slide

$$\{A_i\}\ \{B_j\}$$

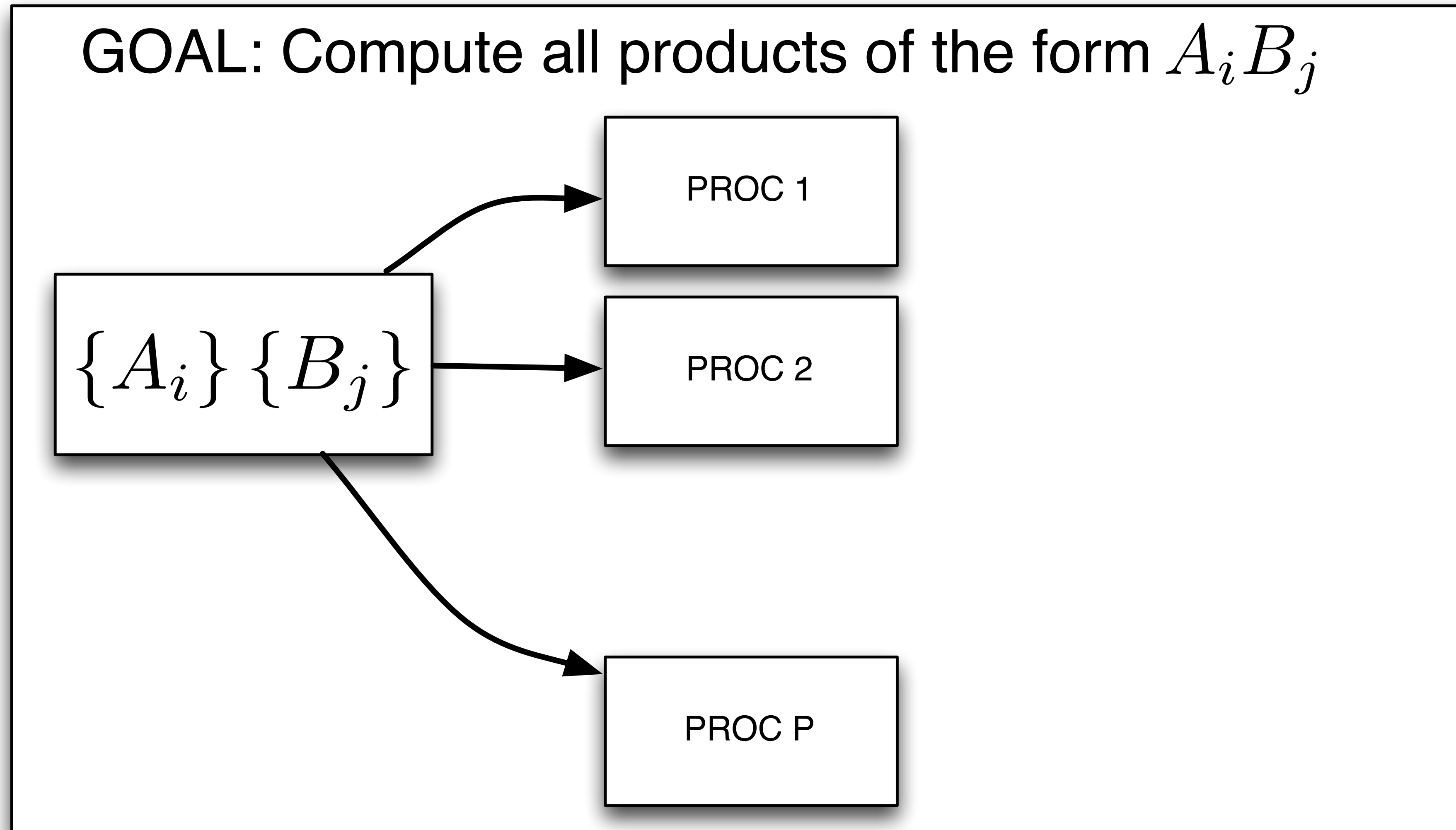
Polynomial codes [Yu, Maddah-Ali, Avestimehr '17]

Intuition: forget matrices for this slide



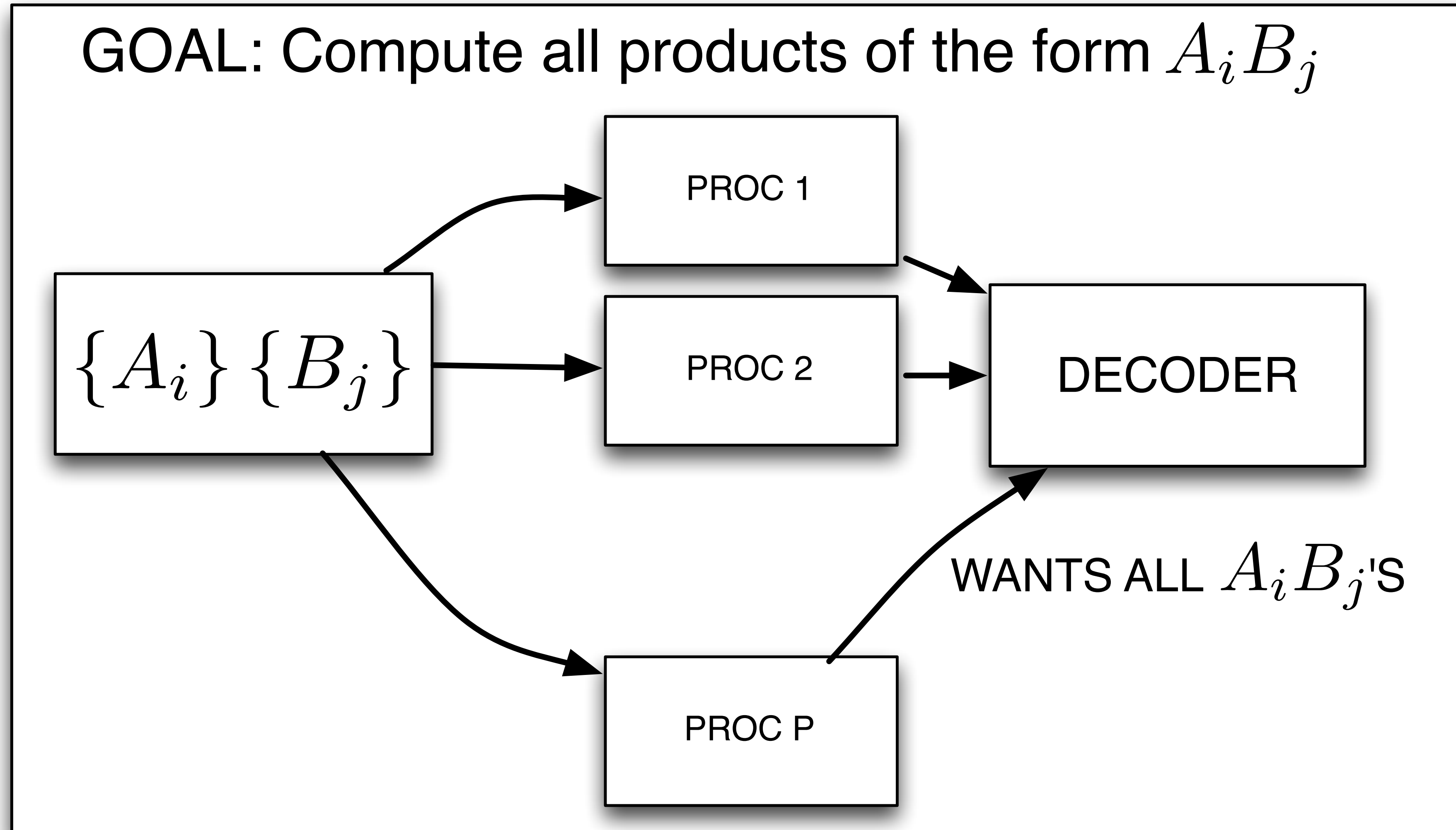
Polynomial codes [Yu, Maddah-Ali, Avestimehr '17]

Intuition: forget matrices for this slide



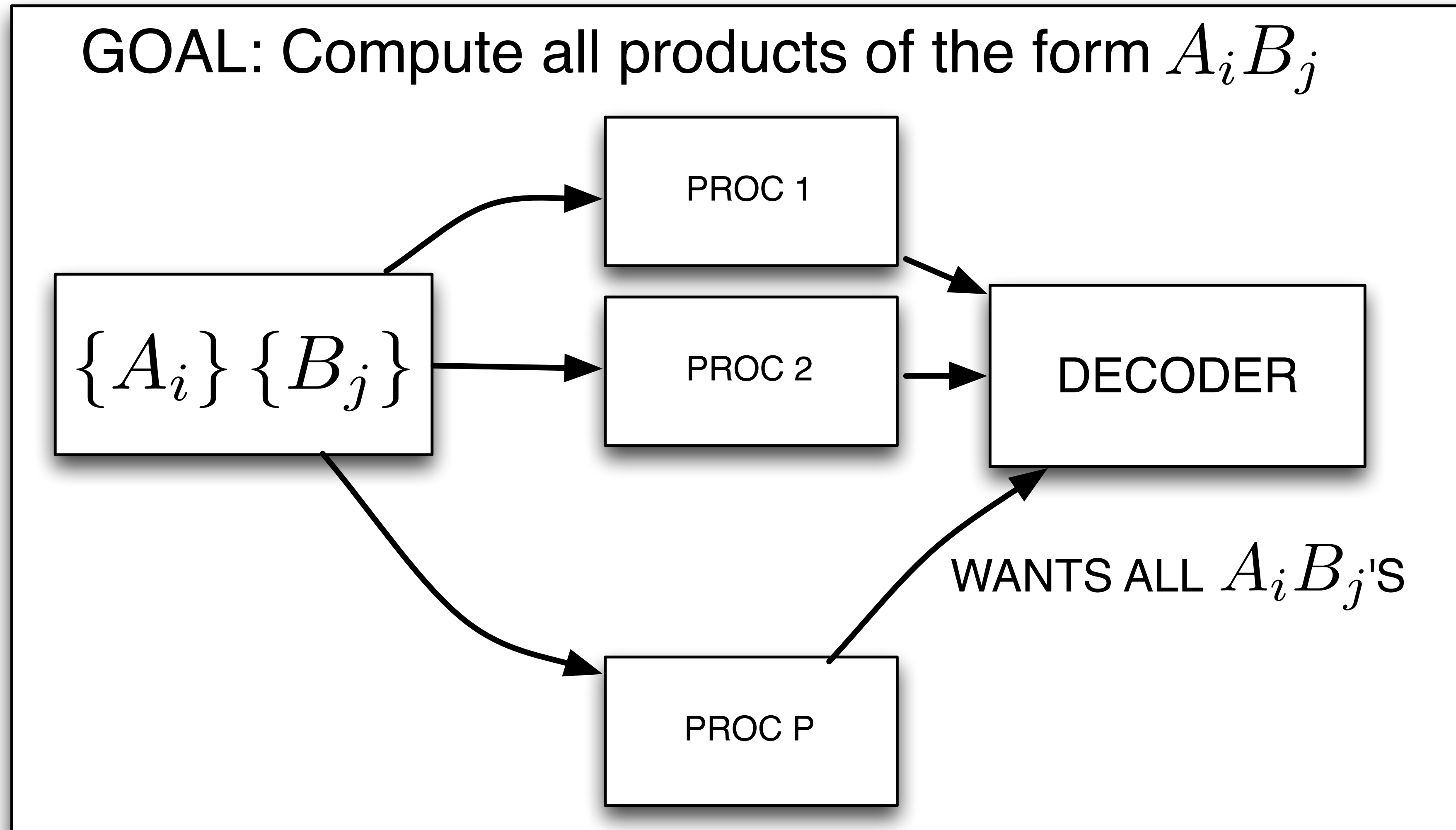
Polynomial codes [Yu, Maddah-Ali, Avestimehr ’17]

Intuition: forget matrices for this slide



Polynomial codes [Yu, Maddah-Ali, Avestimehr '17]

Intuition: forget matrices for this slide

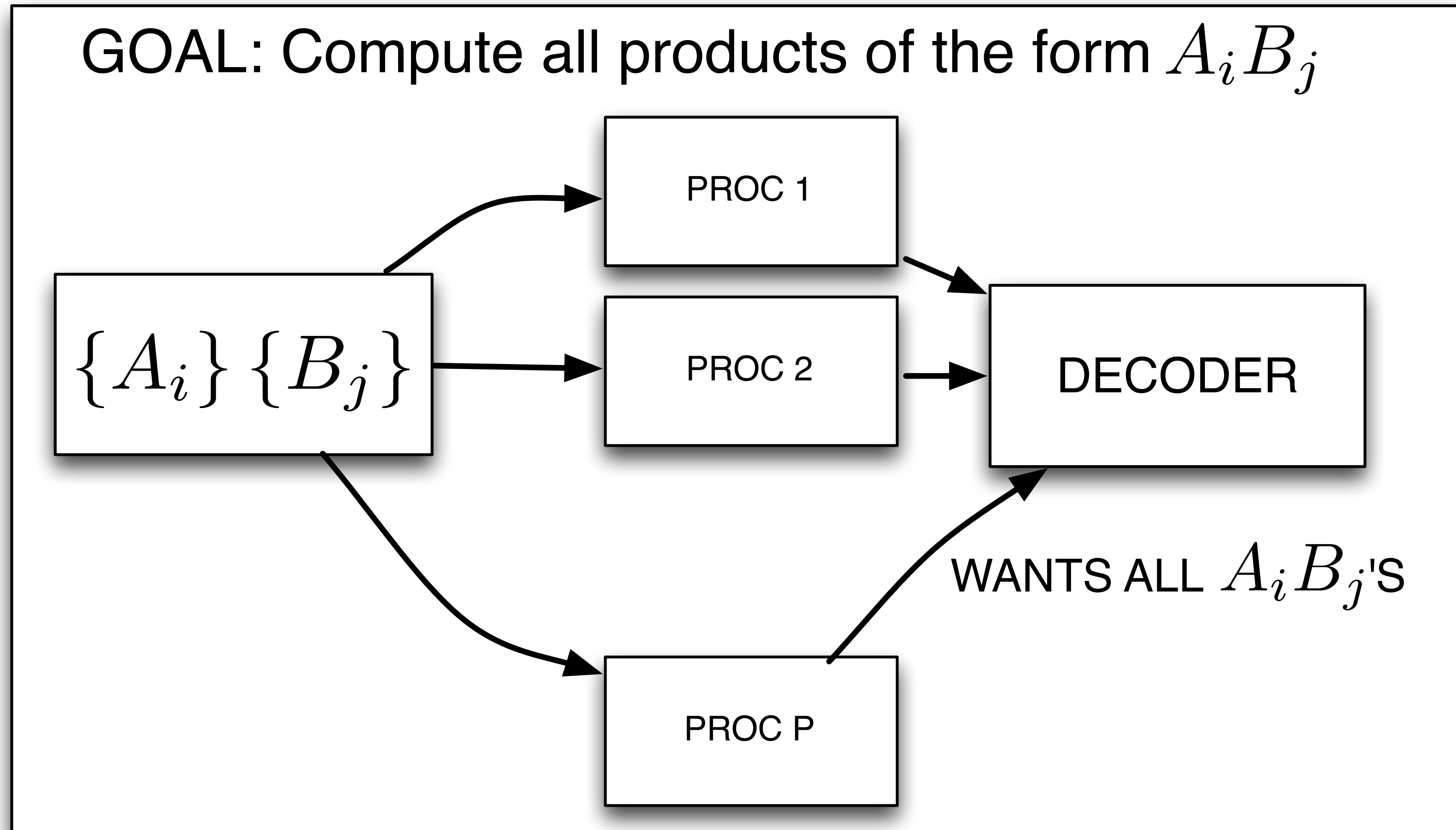


<u>Constraints</u>:

1) Can only send information of *size of* one A_{i} and one B_{j}
2) Processor can only compute a product of its inputs

Polynomial codes [Yu, Maddah-Ali, Avestimehr '17]

Intuition: forget matrices for this slide



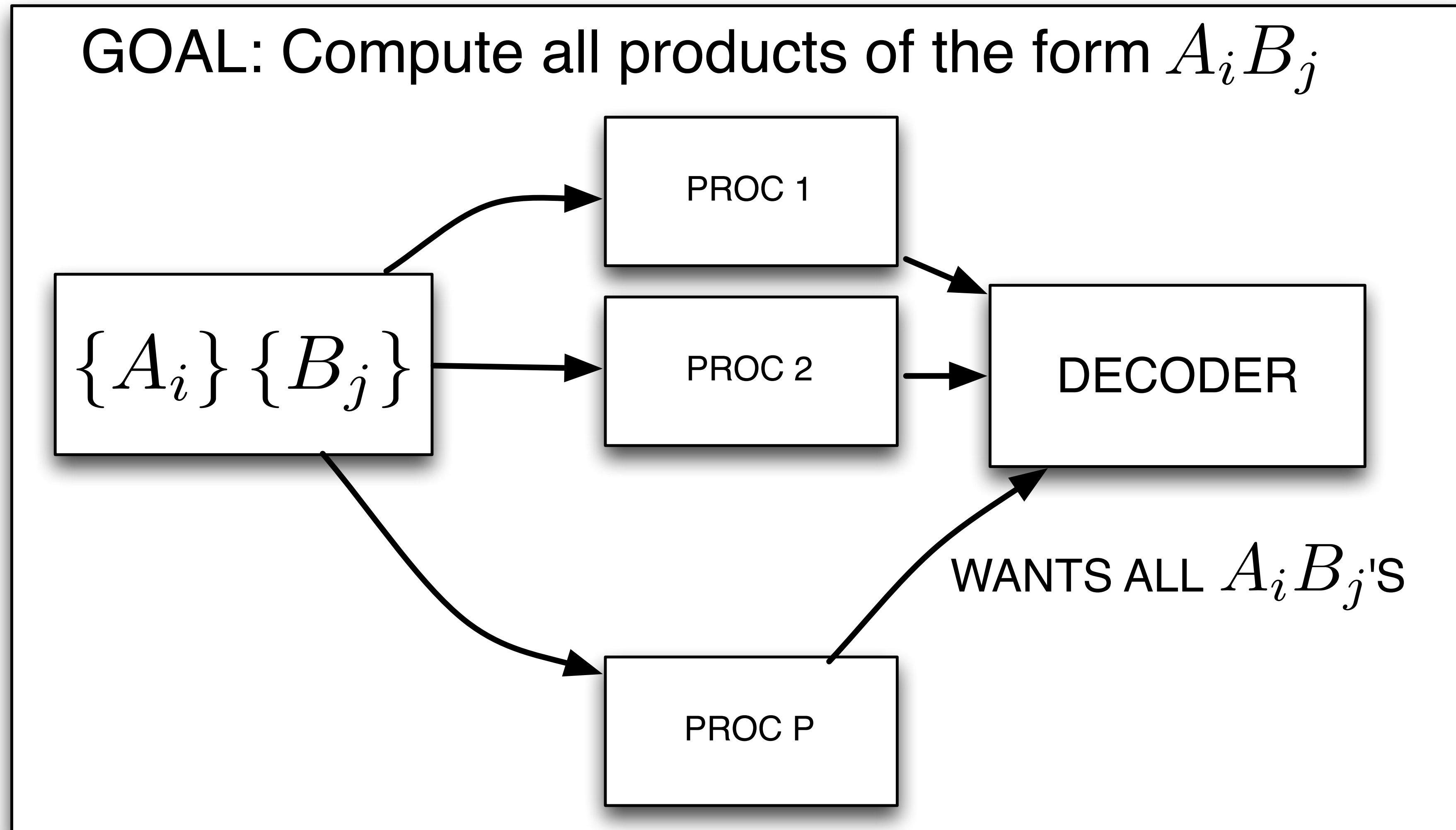
Constraints:

1) Can only send information of *size of* one A_{i} and one B_{j}
2) Processor can only compute a product of its inputs

Solution:
Send $\sum_i \gamma_i A_i$ and $\sum_i \delta_i B_i$

Polynomial codes [Yu, Maddah-Ali, Avestimehr '17]

Intuition: forget matrices for this slide

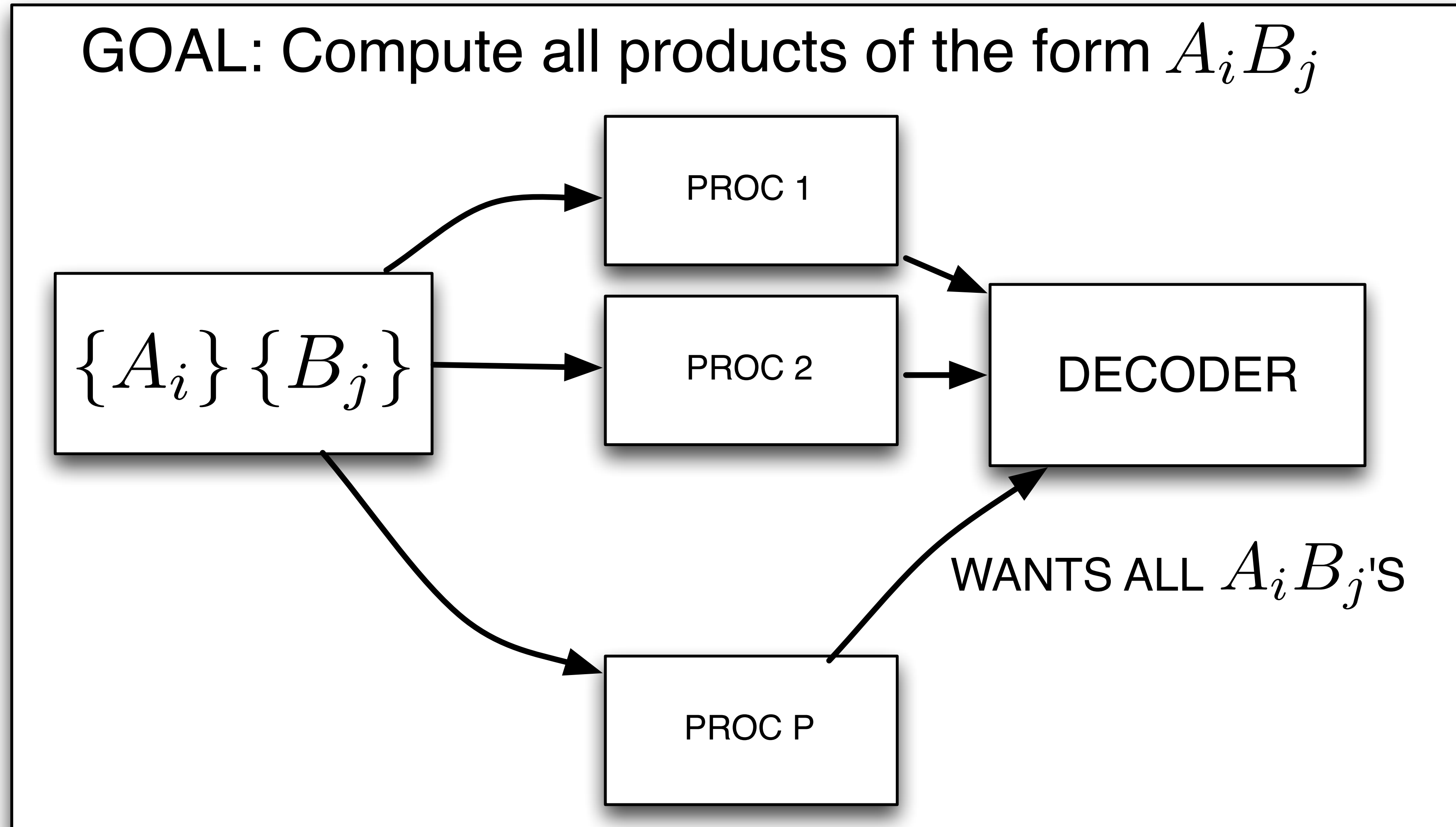


Constraints:
1) Can only send information of *size of* one A_{i} and one B_{j}
2) Processor can only compute a product of its inputs

Solution:
Send $\sum_i \gamma_{ip} A_i$ and $\sum_i \delta_{ip} B_i$

Polynomial codes [Yu, Maddah-Ali, Avestimehr '17]

Intuition: forget matrices for this slide



Constraints:

1) Can only send information of *size of* one A_i and one B_j
2) Processor can only compute a product of its inputs

Solution:

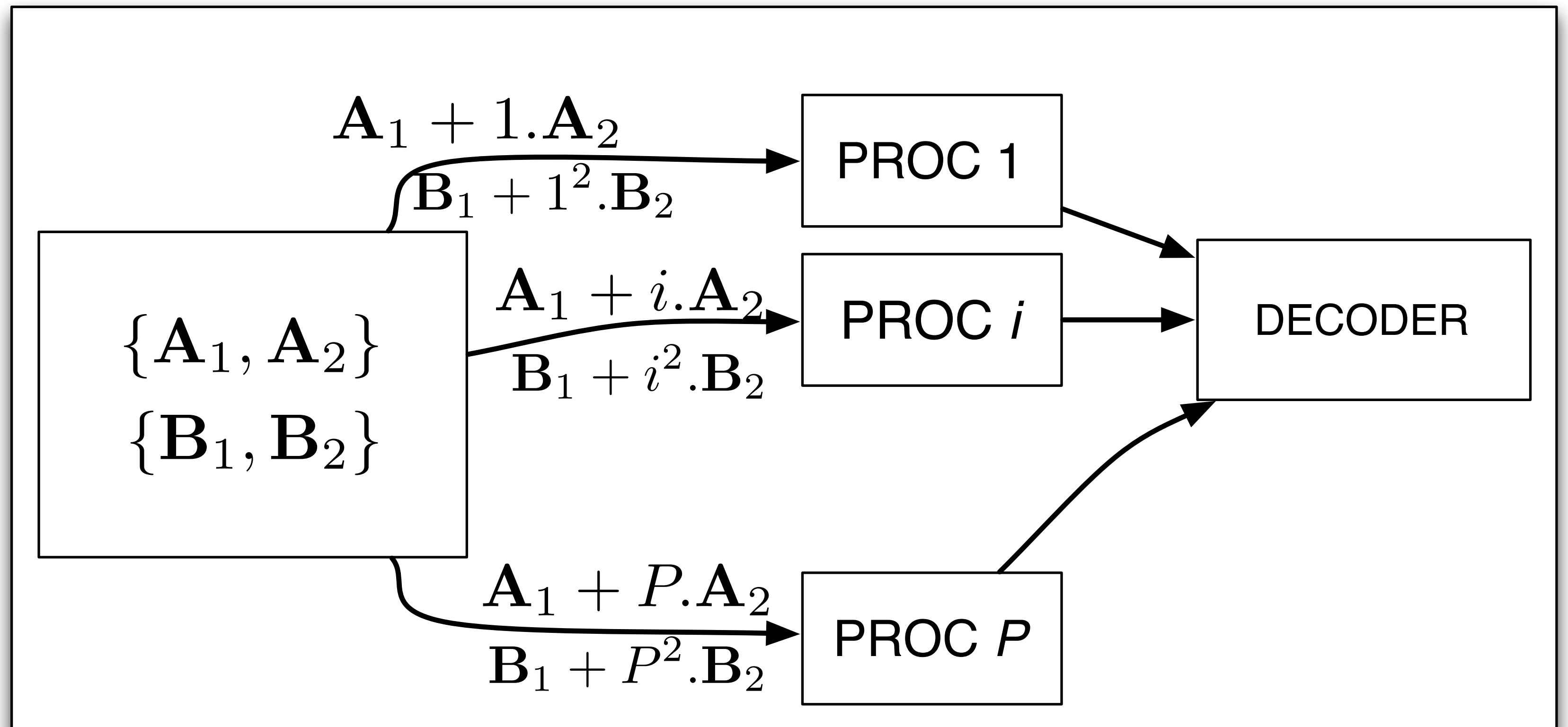
Send $\sum_i \gamma_{ip} A_i$ and $\sum_i \delta_{ip} B_i$

$$\{A_i\}_{i=1}^{m}\ \{B_j\}_{i=1}^{n}$$

Achievability

You *can* use random codes.
But "polynomial codes" get you there with lower enc/dec complexity

Example:
m=2, n=2

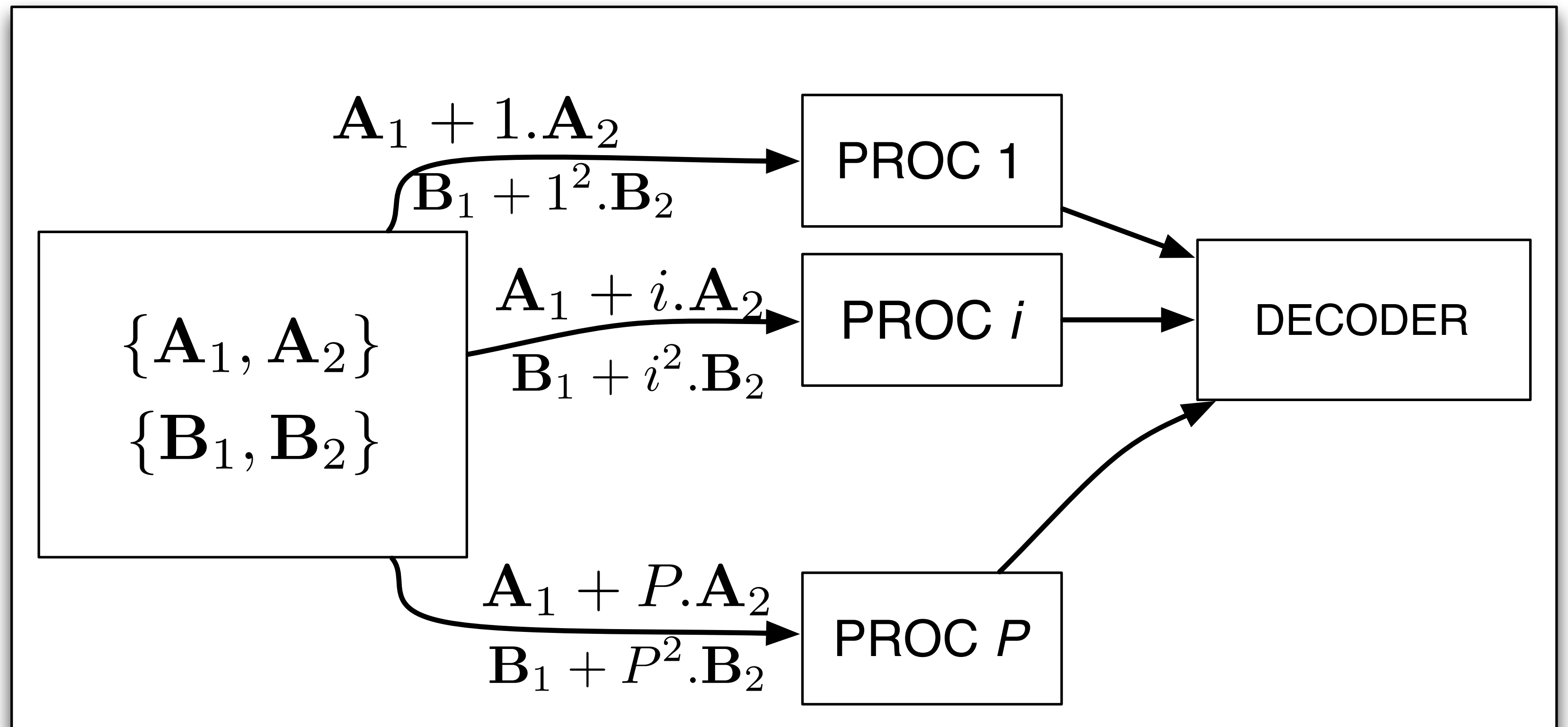


Proc i computes $\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i \tilde{\mathbf{B}}_i = \mathbf{A}_1\mathbf{B}_1 + i\mathbf{A}_2\mathbf{B}_1 + i^2\mathbf{A}_1\mathbf{B}_2 + i^3\mathbf{A}_2\mathbf{B}_2$

Achievability

You *can* use random codes.
But "polynomial codes" get you there with lower enc/dec complexity

Example:
m=2, n=2



Proc i computes $\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i \tilde{\mathbf{B}}_i = \mathbf{A}_1\mathbf{B}_1 + i\mathbf{A}_2\mathbf{B}_1 + i^2\mathbf{A}_1\mathbf{B}_2 + i^3\mathbf{A}_2\mathbf{B}_2$

Fusion center needs outputs from only 4 such processors! e.g. from 1,2,3,4:

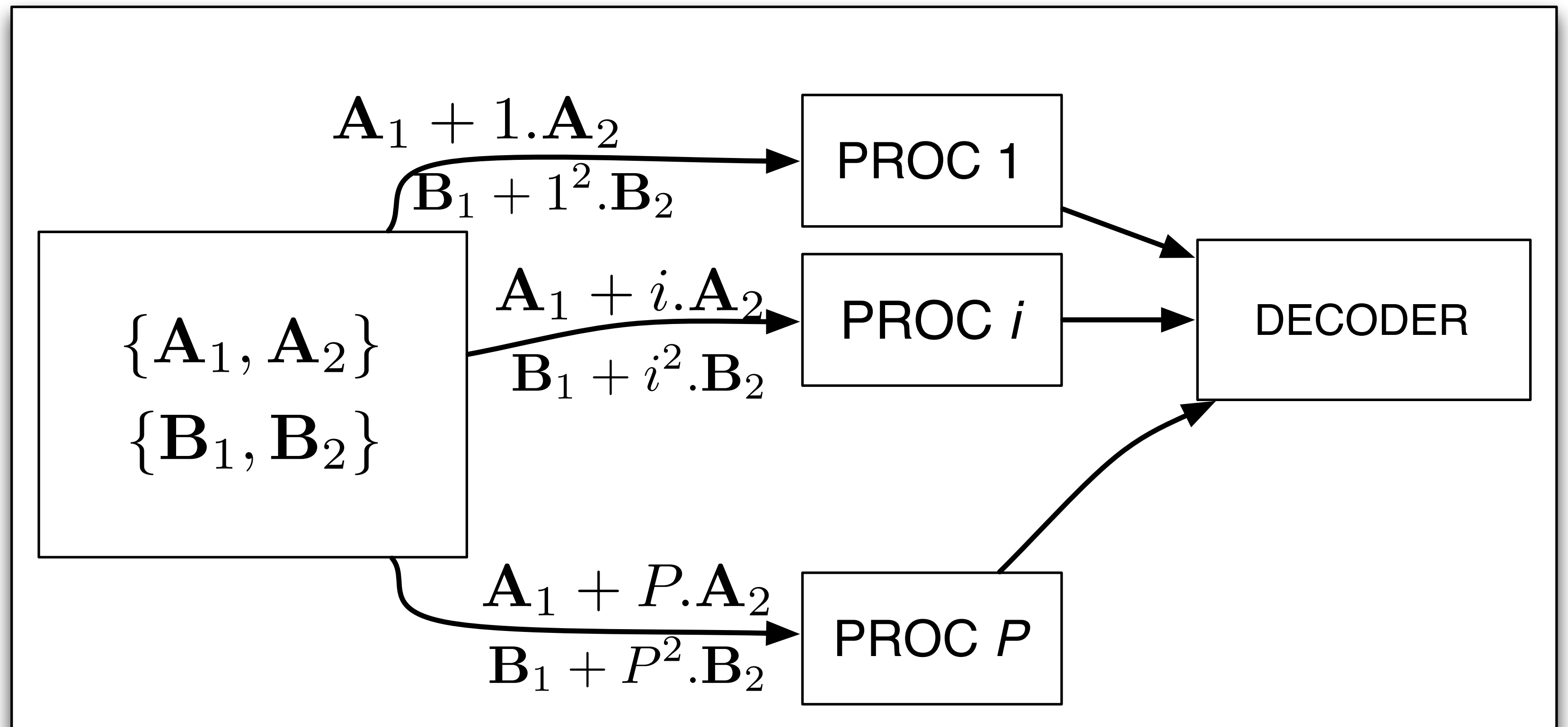
$$\begin{bmatrix} \tilde{\mathbf{C}}_1 \\ \tilde{\mathbf{C}}_2 \\ \tilde{\mathbf{C}}_3 \\ \tilde{\mathbb{C}}_4 \end{bmatrix} = \begin{bmatrix} 1^0 & 1^1 & 1^2 & 1^3 \\ 2^0 & 2^1 & 2^2 & 2^3 \\ 3^0 & 3^1 & 3^2 & 3^3 \\ 4^0 & 4^1 & 4^2 & 4^3 \end{bmatrix} \begin{bmatrix} \mathbf{A}_1\mathbf{B}_1 \\ \mathbf{A}_2\mathbf{B}_1 \\ \mathbf{A}_1\mathbf{B}_2 \\ \mathbf{A}_2\mathbf{B}_2 \end{bmatrix}$$

Invert a Vandermonde matrix

Achievability

You *can* use random codes.
But "polynomial codes" get you there with lower enc/dec complexity

Example:
m=2, n=2



Proc i computes $\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i \tilde{\mathbf{B}}_i = \mathbf{A}_1\mathbf{B}_1 + i\mathbf{A}_2\mathbf{B}_1 + i^2\mathbf{A}_1\mathbf{B}_2 + i^3\mathbf{A}_2\mathbf{B}_2$

Fusion center needs outputs from only 4 such processors! e.g. from 1,2,3,4:

$$\begin{bmatrix} \tilde{\mathbf{C}}_1 \\ \tilde{\mathbf{C}}_2 \\ \tilde{\mathbf{C}}_3 \\ \tilde{\mathbb{C}}_4 \end{bmatrix} = \begin{bmatrix} 1^0 & 1^1 & 1^2 & 1^3 \\ 2^0 & 2^1 & 2^2 & 2^3 \\ 3^0 & 3^1 & 3^2 & 3^3 \\ 4^0 & 4^1 & 4^2 & 4^3 \end{bmatrix} \begin{bmatrix} \mathbf{A}_1\mathbf{B}_1 \\ \mathbf{A}_2\mathbf{B}_1 \\ \mathbf{A}_1\mathbf{B}_2 \\ \mathbf{A}_2\mathbf{B}_2 \end{bmatrix}$$

Invert a Vandermonde matrix

In general, Recovery Threshold = mn (attained using RS-code-type construction)

Summary so far…

- V x V : Coding offers little advantage over replication
- M x V: Short-Dot codes provide arbitrary gains over replication, MDS coding,
- M x M: polynomial coding provides arbitrary gains over M x V codes

What additional costs come with coding?

- encoding and decoding complexity (skipped here for simplicity)
- Next: degradation is not graceful as you pull deadline earlier

To see this, let’s look a problem with repeated M x V, and slow convergence to solution

Understanding a limitation of coding: Coding for linear iterative solutions

MxV computation input

$$\mathbf{x}^{(l+1)} = (1-d)\mathbf{A}\mathbf{x}^{(l)} + d\mathbf{r}.$$

Converges to $\mathbf{x}^*$ satisfying $\mathbf{x}^* = (1-d)\mathbf{A}\mathbf{x}^* + d\mathbf{r}$.

Subtracting, $\mathbf{e}^{(l+1)} = (1-d)\mathbf{A}\mathbf{e}^{(l)}$, where $\mathbf{e}^{(l)} = \mathbf{x}^{(l)} - \mathbf{x}^*$.

“Coding Method for Parallel Iterative Linear Solver,” Y Yang, P Grover, S Kar, Submitted

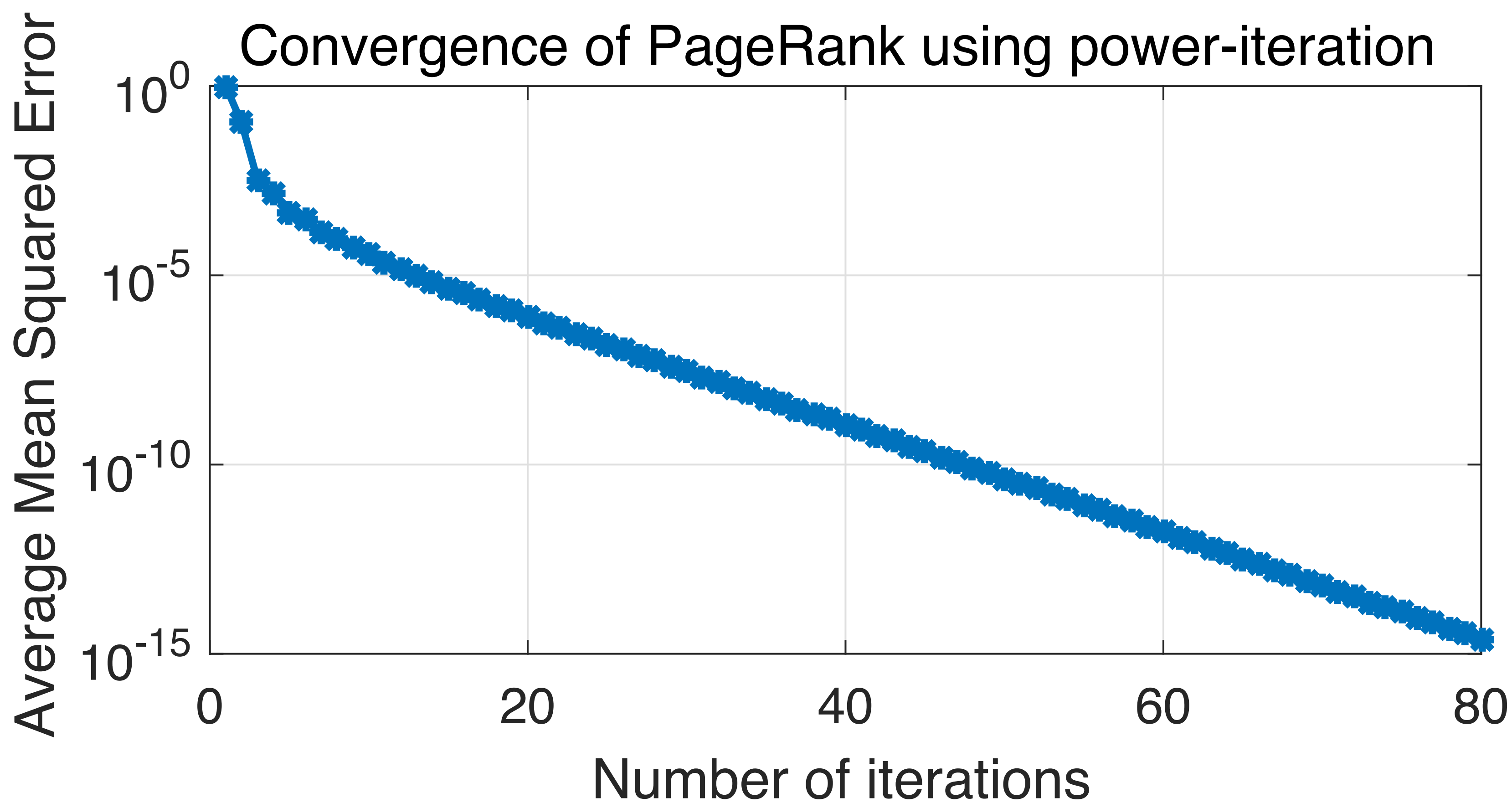
Understanding a limitation of coding: Coding for linear iterative solutions

MxV computation input

$$\mathbf{x}^{(l+1)} = (1-d)\mathbf{A}\mathbf{x}^{(l)} + d\mathbf{r}.$$

Converges to $\mathbf{x}^*$ satisfying $\mathbf{x}^* = (1-d)\mathbf{A}\mathbf{x}^* + d\mathbf{r}.$

Subtracting, $\mathbf{e}^{(l+1)} = (1-d)\mathbf{A}\mathbf{e}^{(l)}$, where $\mathbf{e}^{(l)} = \mathbf{x}^{(l)} - \mathbf{x}^*.$



“Coding Method for Parallel Iterative Linear Solver,” Y Yang, P Grover, S Kar, Submitted

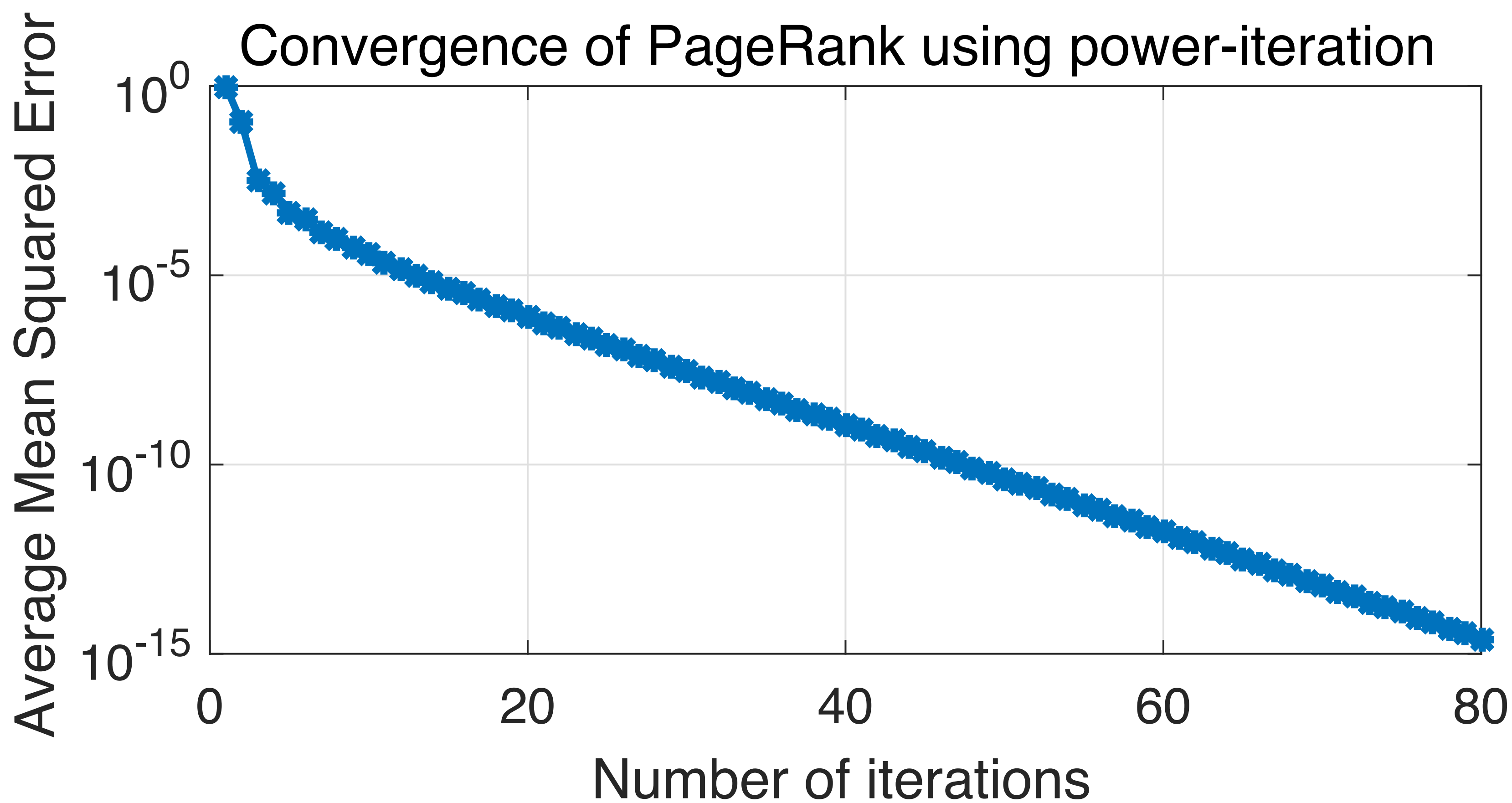
Understanding a limitation of coding: Coding for linear iterative solutions

MxV computation input

$$\mathbf{x}^{(l+1)} = (1-d)\mathbf{A}\mathbf{x}^{(l)} + d\mathbf{r}.$$

Converges to $\mathbf{x}^*$ satisfying $\mathbf{x}^* = (1-d)\mathbf{A}\mathbf{x}^* + d\mathbf{r}$.

Subtracting, $\mathbf{e}^{(l+1)} = (1-d)\mathbf{A}\mathbf{e}^{(l)}$, where $\mathbf{e}^{(l)} = \mathbf{x}^{(l)} - \mathbf{x}^*$.



Next: how to code multiple linear iterative problems in parallel

"Coding Method for Parallel Iterative Linear Solver," Y Yang, P Grover, S Kar, Submitted

Understanding a limitation of coding: Coding for linear iterative solutions

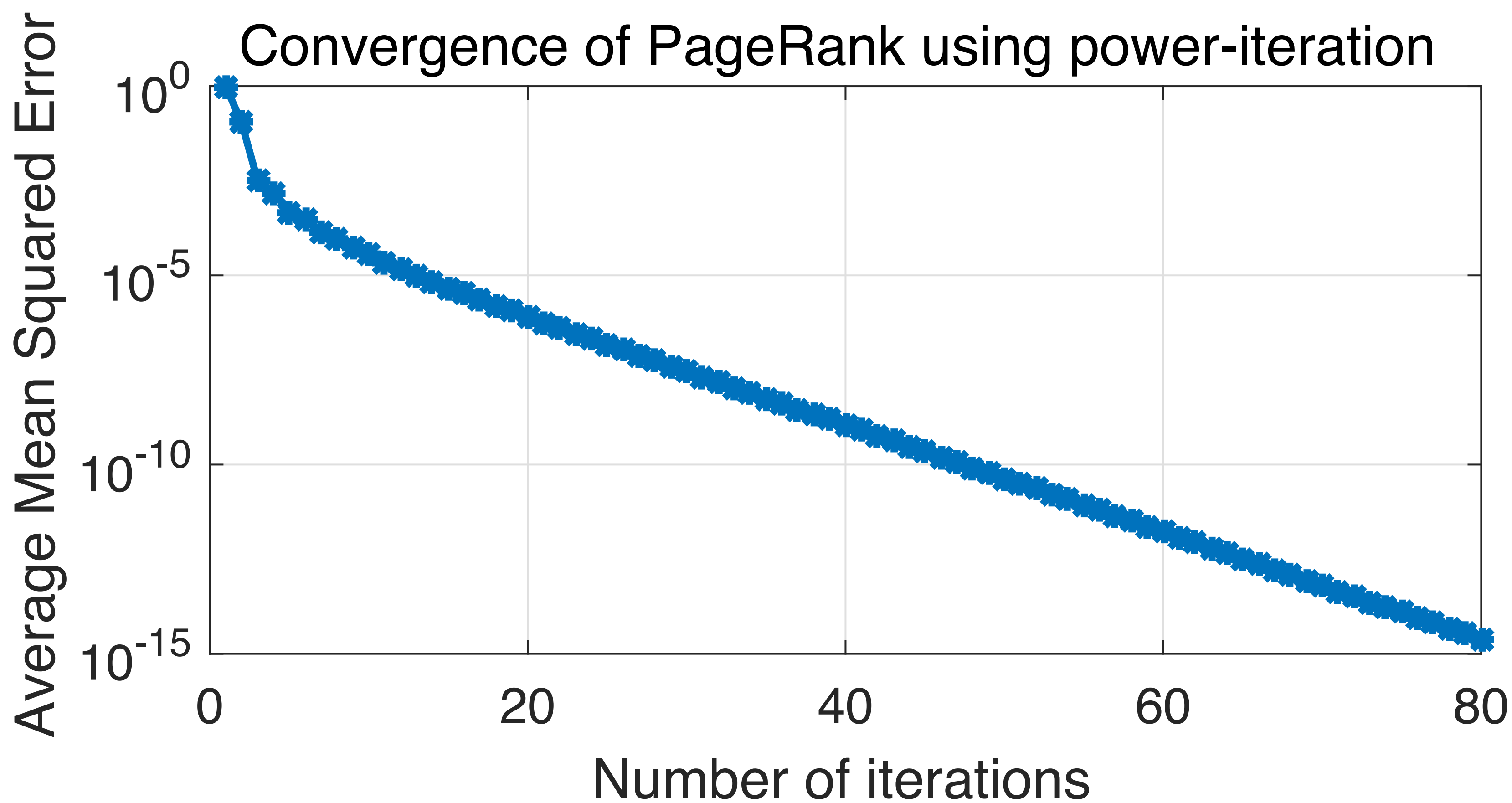
MxV computation input

$$\mathbf{x}^{(l+1)} = (1-d)\mathbf{A}\mathbf{x}^{(l)} + d\mathbf{r}.$$

Converges to $\mathbf{x}^*$ satisfying $\mathbf{x}^* = (1-d)\mathbf{A}\mathbf{x}^* + d\mathbf{r}$.

$\mathbf{x}^*$ linear in $\mathbf{r}$

Subtracting, $\mathbf{e}^{(l+1)} = (1-d)\mathbf{A}\mathbf{e}^{(l)}$, where $\mathbf{e}^{(l)} = \mathbf{x}^{(l)} - \mathbf{x}^*$.

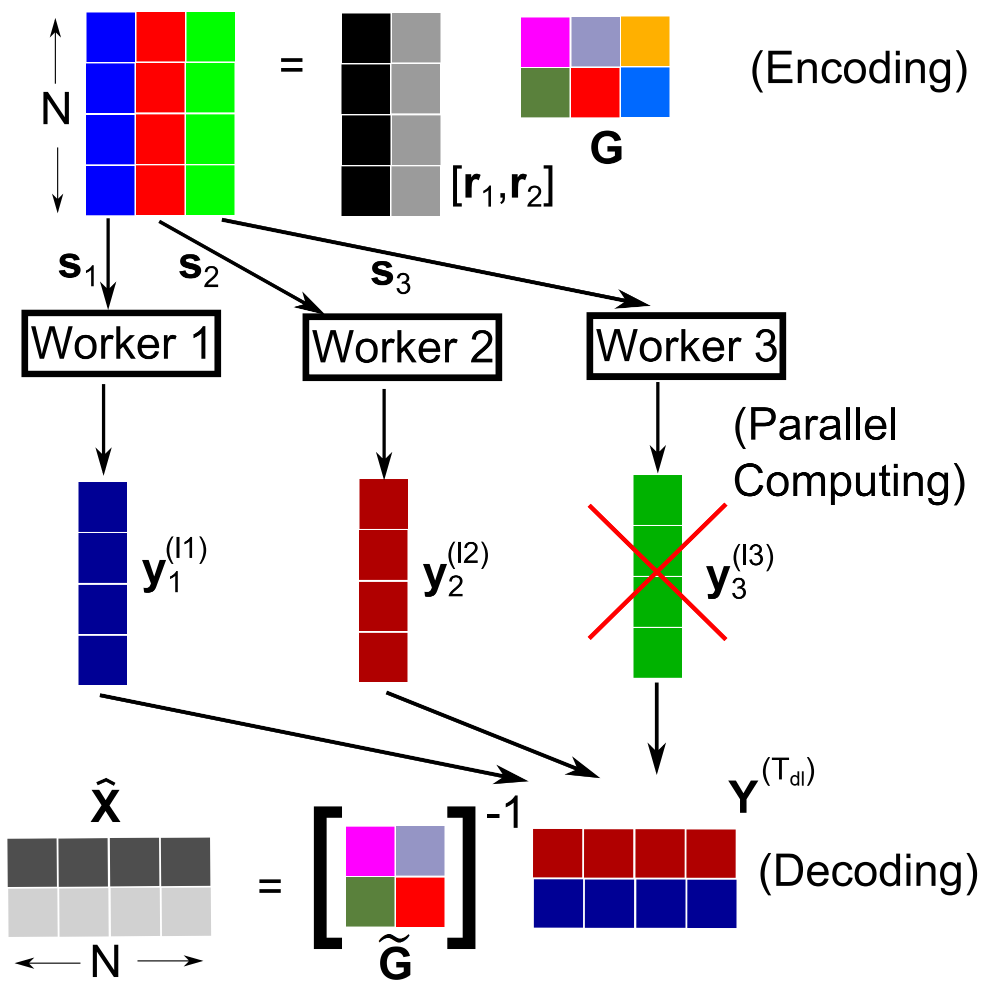


Next: how to code multiple linear iterative problems in parallel

"Coding Method for Parallel Iterative Linear Solver," Y Yang, P Grover, S Kar, Submitted

Solving multiple iterative problems in parallel

Classical coded computation applied to linear iterative problems



- Initialize (Encoding)

$$[\mathbf{s}_1, \ldots, \mathbf{s}_P] = [\mathbf{r}_1, \ldots, \mathbf{r}_k] \cdot \mathbf{G}_{k \times P}.$$

- Parallel Computing:
  l_i power iterations at the i-th worker with input $\mathbf{s}_i$

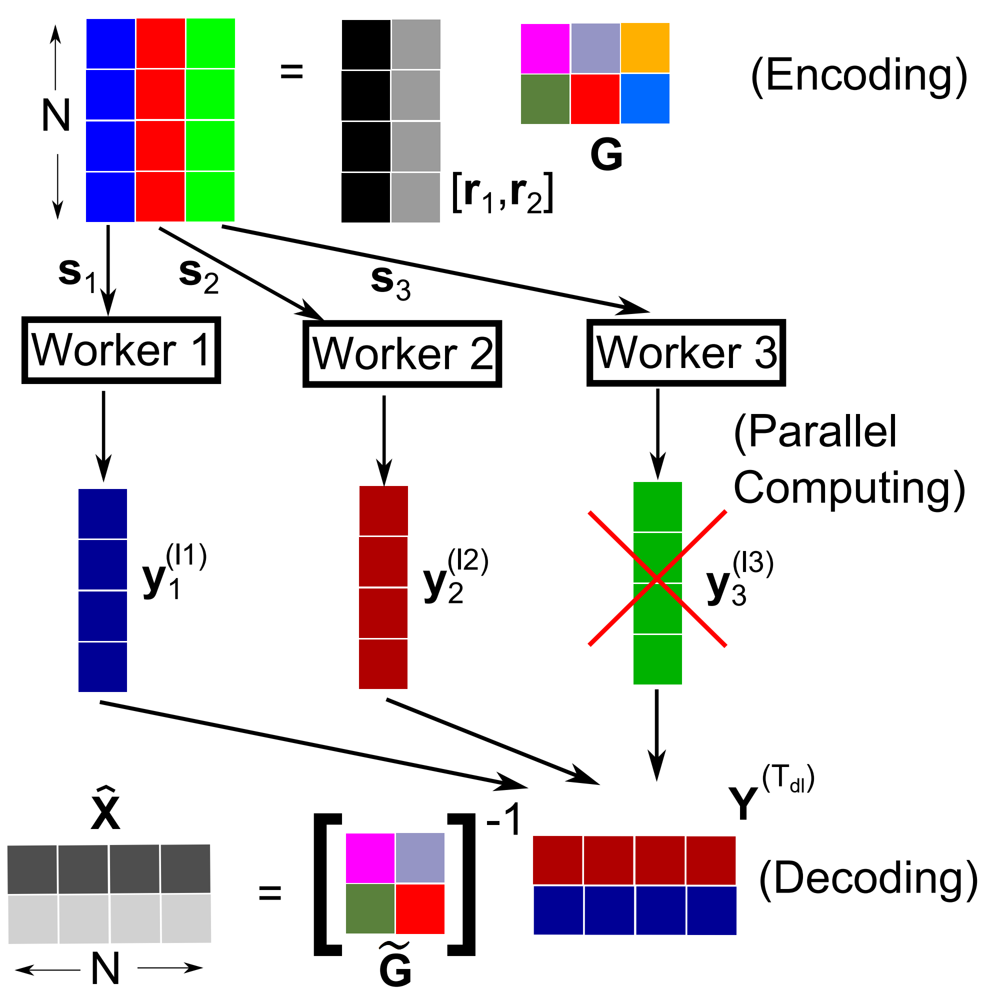
$$\mathbf{Y}_{N \times P}^{(T_{\text{dl}})} = [\mathbf{y}_1^{(l_1)}, \ldots, \mathbf{y}_P^{(l_P)}].$$

- Post Processing (Decoding) Matrix inversion on fastest k processors.

$$\widehat{\mathbf{X}}^\top = \tilde{\mathbf{G}}^{-1}(\mathbf{Y}^{(T_{\text{dl}})})^\top.$$

Solving multiple iterative problems in parallel

Classical coded computation applied to linear iterative problems



- Initialize (Encoding)

$$[\mathbf{s}_1, \ldots, \mathbf{s}_P] = [\mathbf{r}_1, \ldots, \mathbf{r}_k] \cdot \mathbf{G}_{k \times P}.$$

- Parallel Computing: l_i power iterations at the i-th worker with input $\mathbf{s}_i$

$$\mathbf{Y}^{(T_{\mathsf{dl}})}_{N \times P} = [\mathbf{y}_1^{(l_1)}, \ldots, \mathbf{y}_P^{(l_P)}].$$

- Post Processing (Decoding) Matrix inversion on fastest k processors.

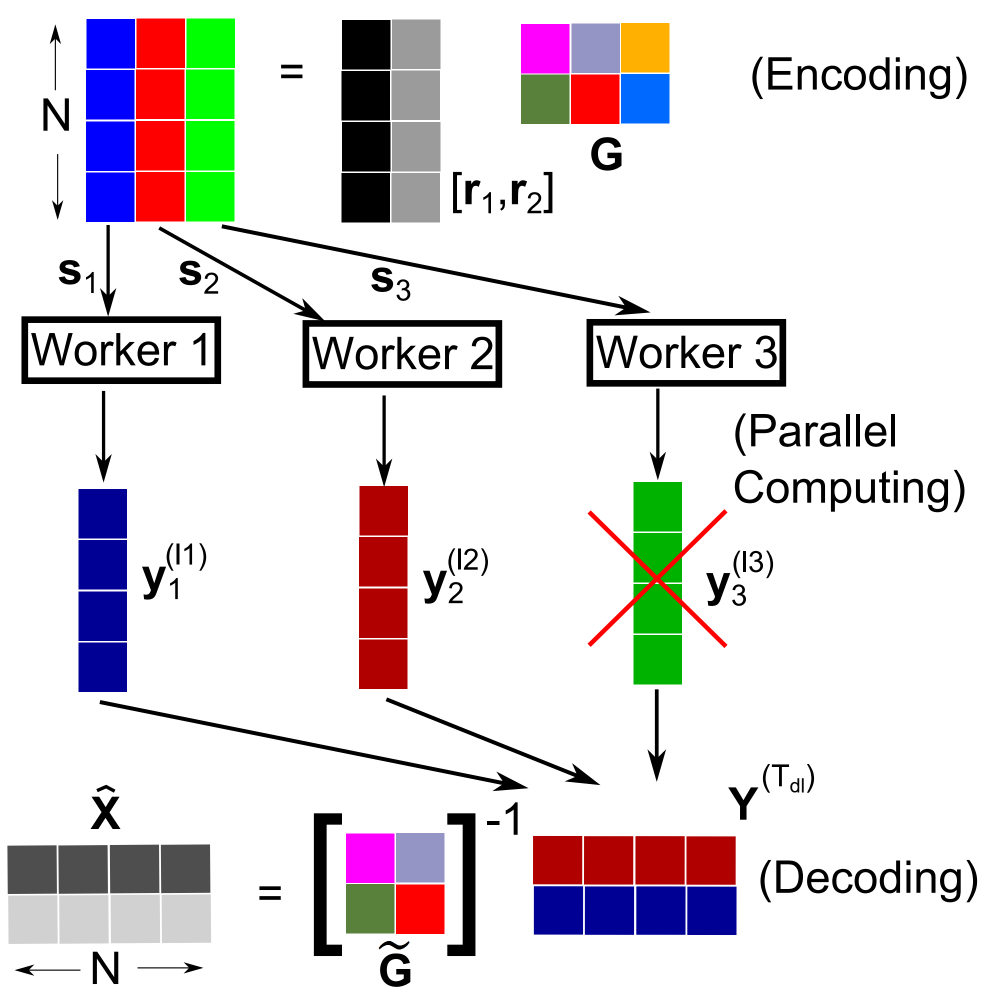
$$\widehat{\mathbf{X}}^\top = \tilde{\mathbf{G}}^{-1} (\mathbf{Y}^{(T_{\mathsf{dl}})})^\top.$$

Is this invertible?

Is this well conditioned?

Solving multiple iterative problems in parallel

Classical coded computation applied to linear iterative problems



- Initialize (Encoding)

$$[\mathbf{s}_1, \ldots, \mathbf{s}_P] = [\mathbf{r}_1, \ldots, \mathbf{r}_k] \cdot \mathbf{G}_{k\times P}.$$

- Parallel Computing:
  l_i power iterations at the i-th worker with input $\mathbf{s}_i$

$$\mathbf{Y}_{N\times P}^{(T_{\text{dl}})} = [\mathbf{y}_1^{(l_1)}, \ldots, \mathbf{y}_P^{(l_P)}].$$

- Post Processing (Decoding) Matrix inversion on fastest k processors.

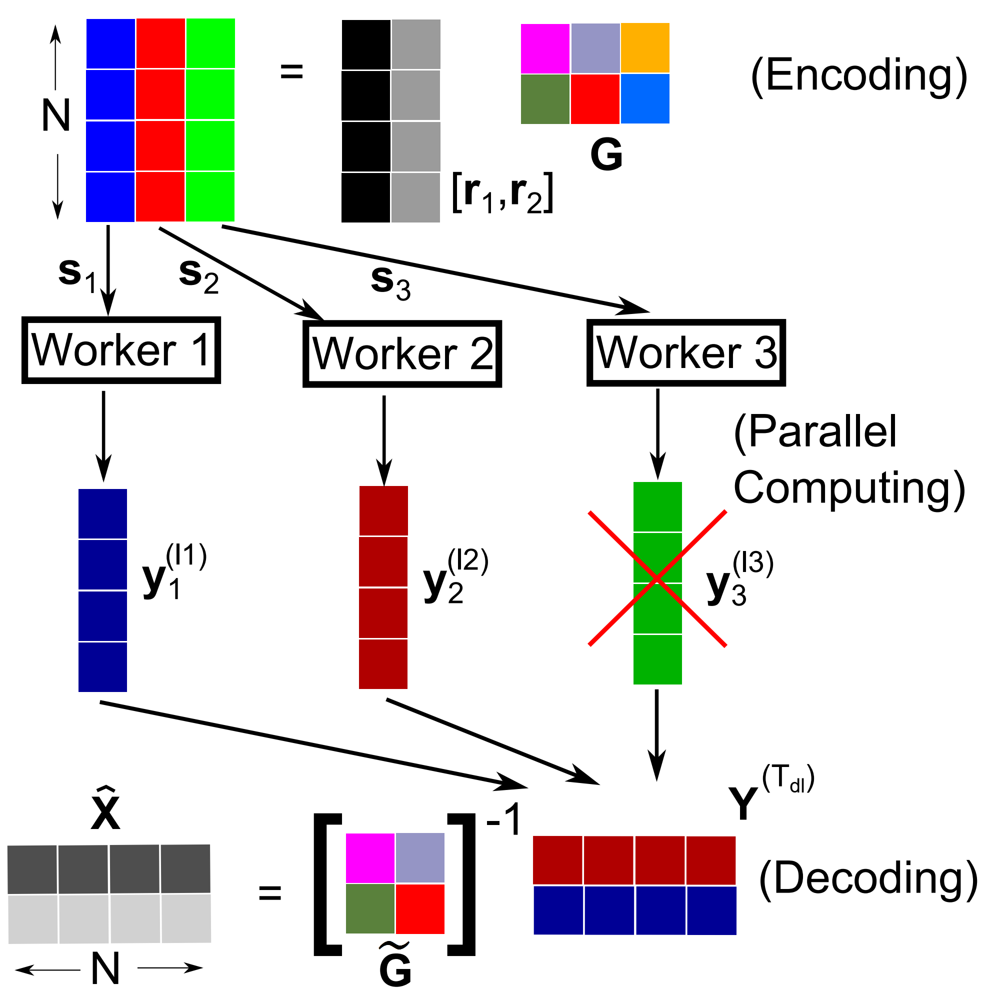
$$\widehat{\mathbf{X}}^\top = \tilde{\mathbf{G}}^{-1} (\mathbf{Y}^{(T_{\text{dl}})})^\top.$$

Is this invertible? Yes!

Is this well conditioned?

Solving multiple iterative problems in parallel

Classical coded computation applied to linear iterative problems



- Initialize (Encoding)

$$[\mathbf{s}_1, \ldots, \mathbf{s}_P] = [\mathbf{r}_1, \ldots, \mathbf{r}_k] \cdot \mathbf{G}_{k \times P}.$$

- Parallel Computing:
  l_i power iterations at the i-th worker with input $\mathbf{s}_i$

$$\mathbf{Y}_{N \times P}^{(T_{\text{dl}})} = [\mathbf{y}_1^{(l_1)}, \ldots, \mathbf{y}_P^{(l_P)}].$$

- Post Processing (Decoding) Matrix inversion on fastest k processors.

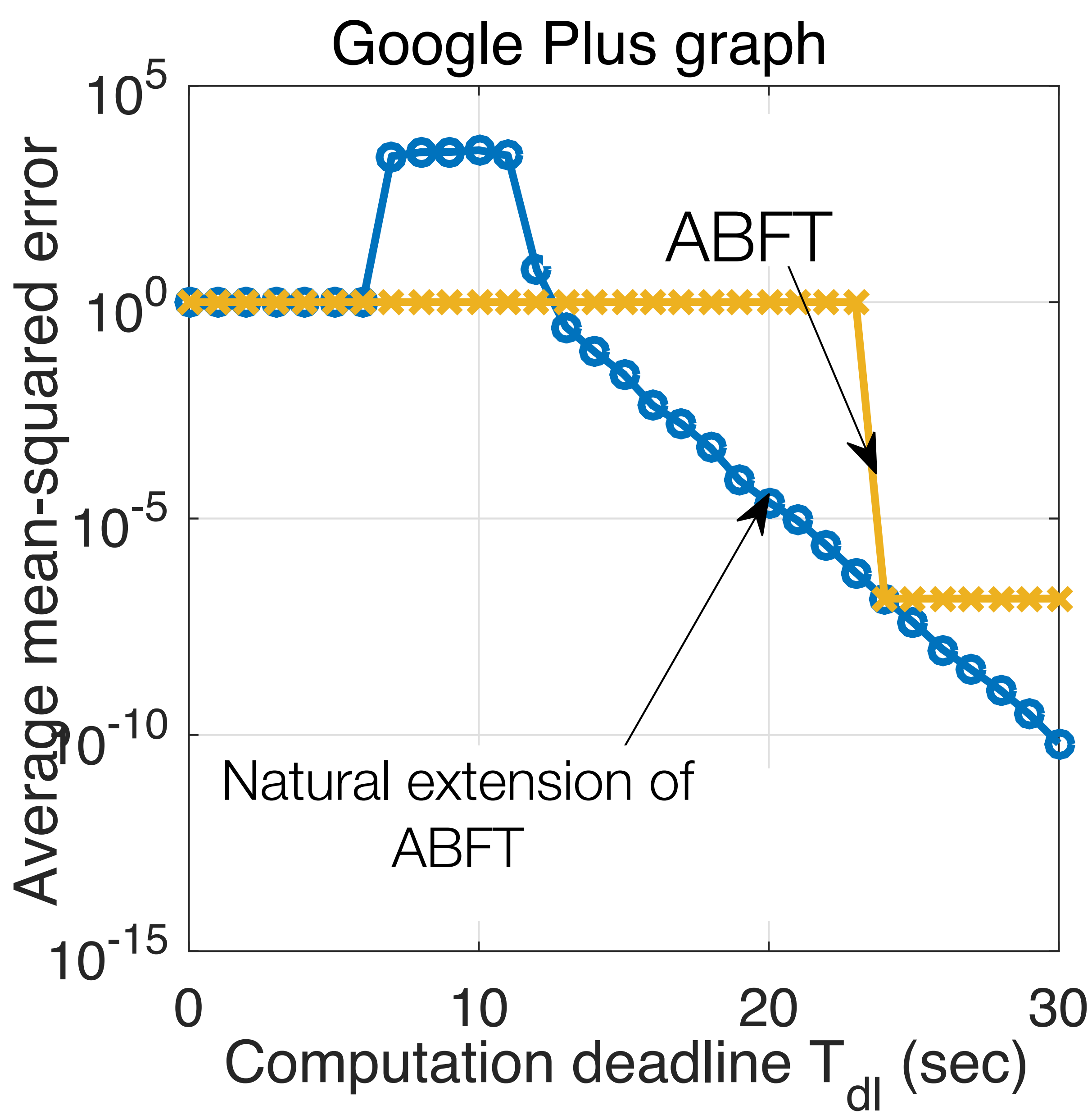
$$\widehat{\mathbf{X}}^\top = \tilde{\mathbf{G}}^{-1} (\mathbf{Y}^{(T_{\text{dl}})})^\top.$$

Is this invertible? Yes!

Is this well conditioned? No!

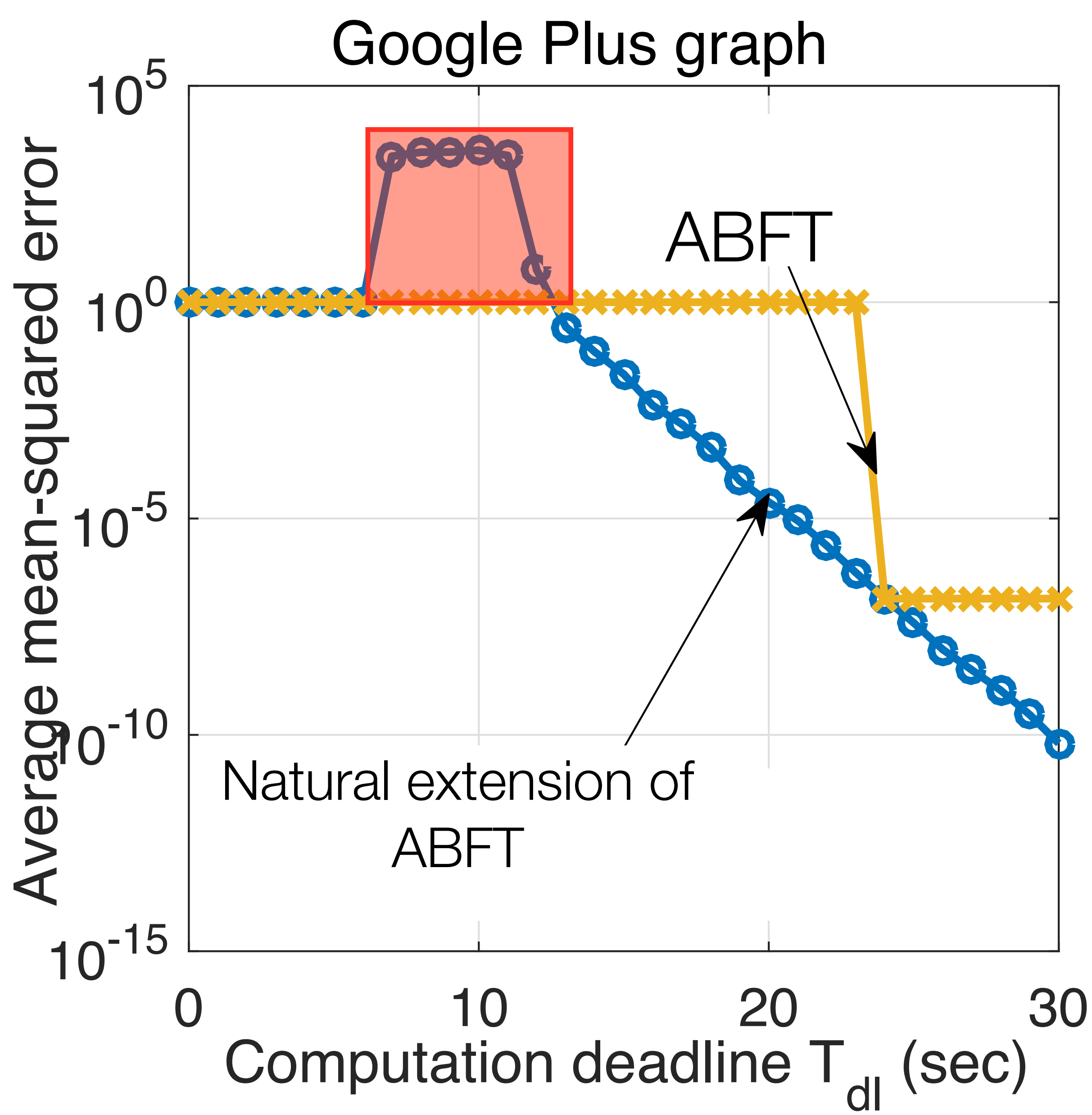
What is the effect of a poor conditioning number?
Error blows up!

Experiments on CMU clusters:



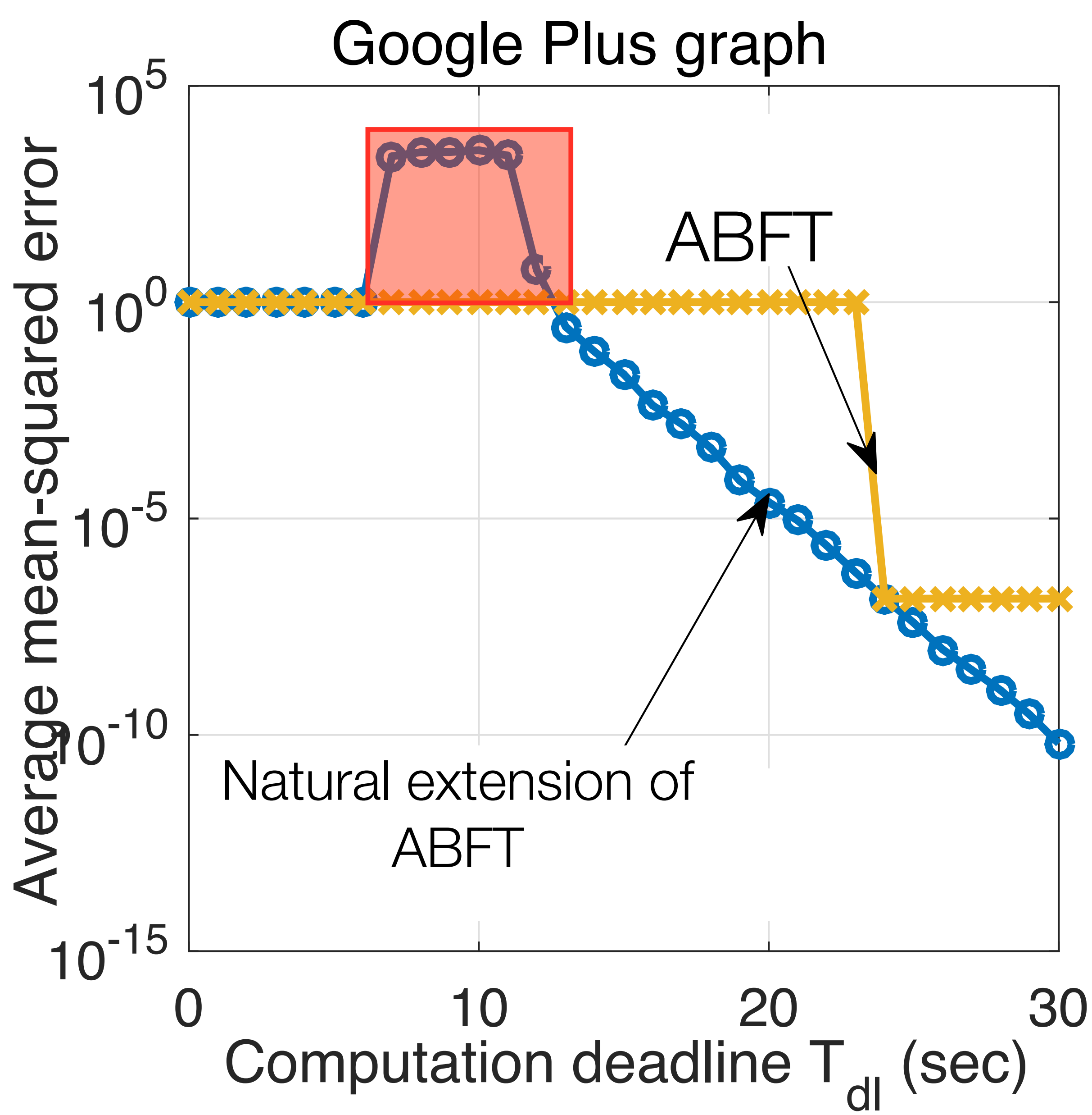
What is the effect of a poor conditioning number? Error blows up!

Experiments on CMU clusters:



What is the effect of a poor conditioning number? Error blows up!

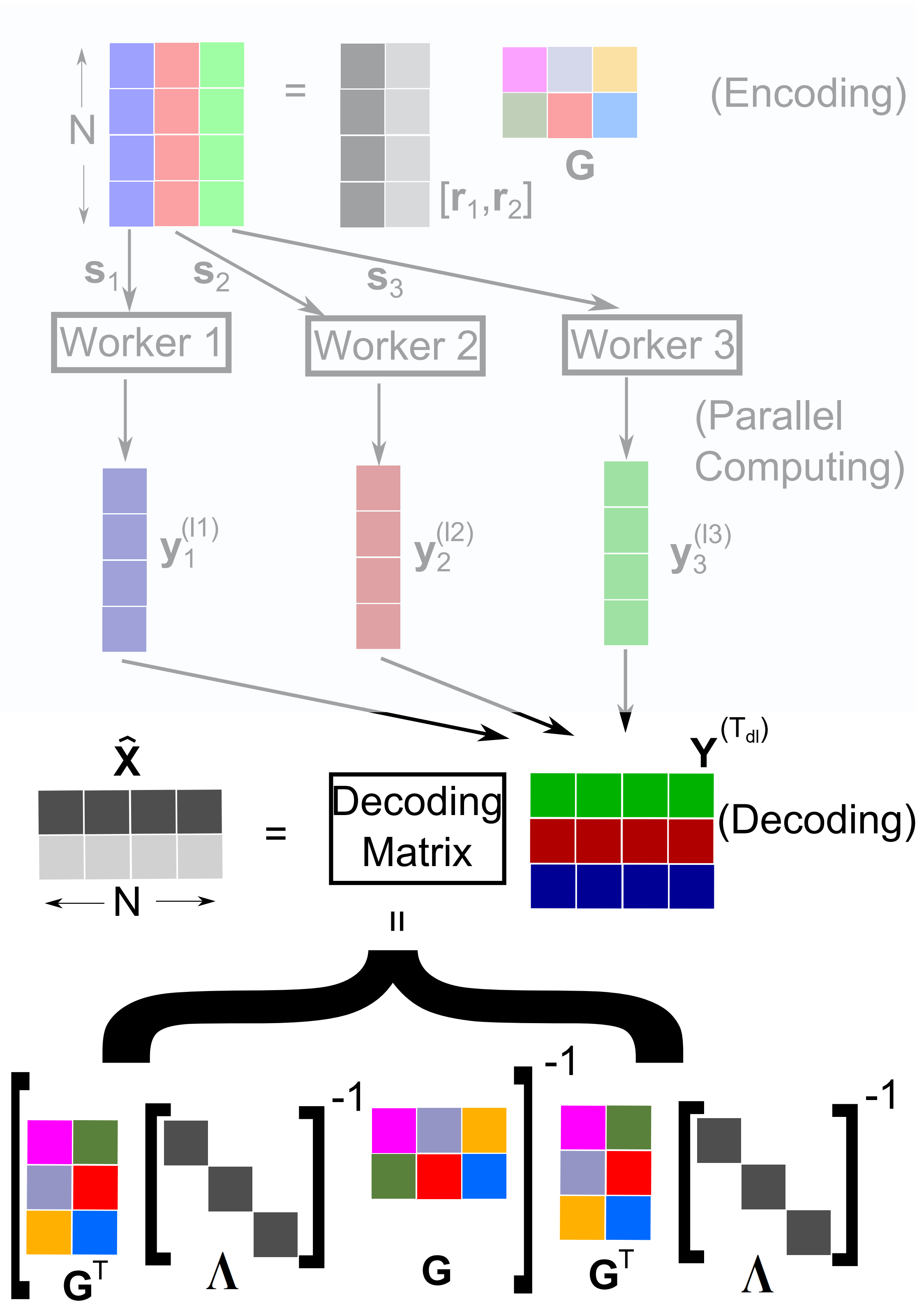
Experiments on CMU clusters:



Similar issues arise in designing good “analog coding with erasures”

[Haikin, Zamir ISIT’16][Haikin, Zamir, Gavish ‘17]

A graceful degradation with time: Coded computing with weighted least squares



- Initialize (Encoding)

$$[\mathbf{s}_1,\ldots,\mathbf{s}_P] = [\mathbf{r}_1,\ldots,\mathbf{r}_k]\cdot\mathbf{G}.$$

- Parallel Computing:
  l_i power iterations at the i-th worker with input $\mathbf{s}_i$

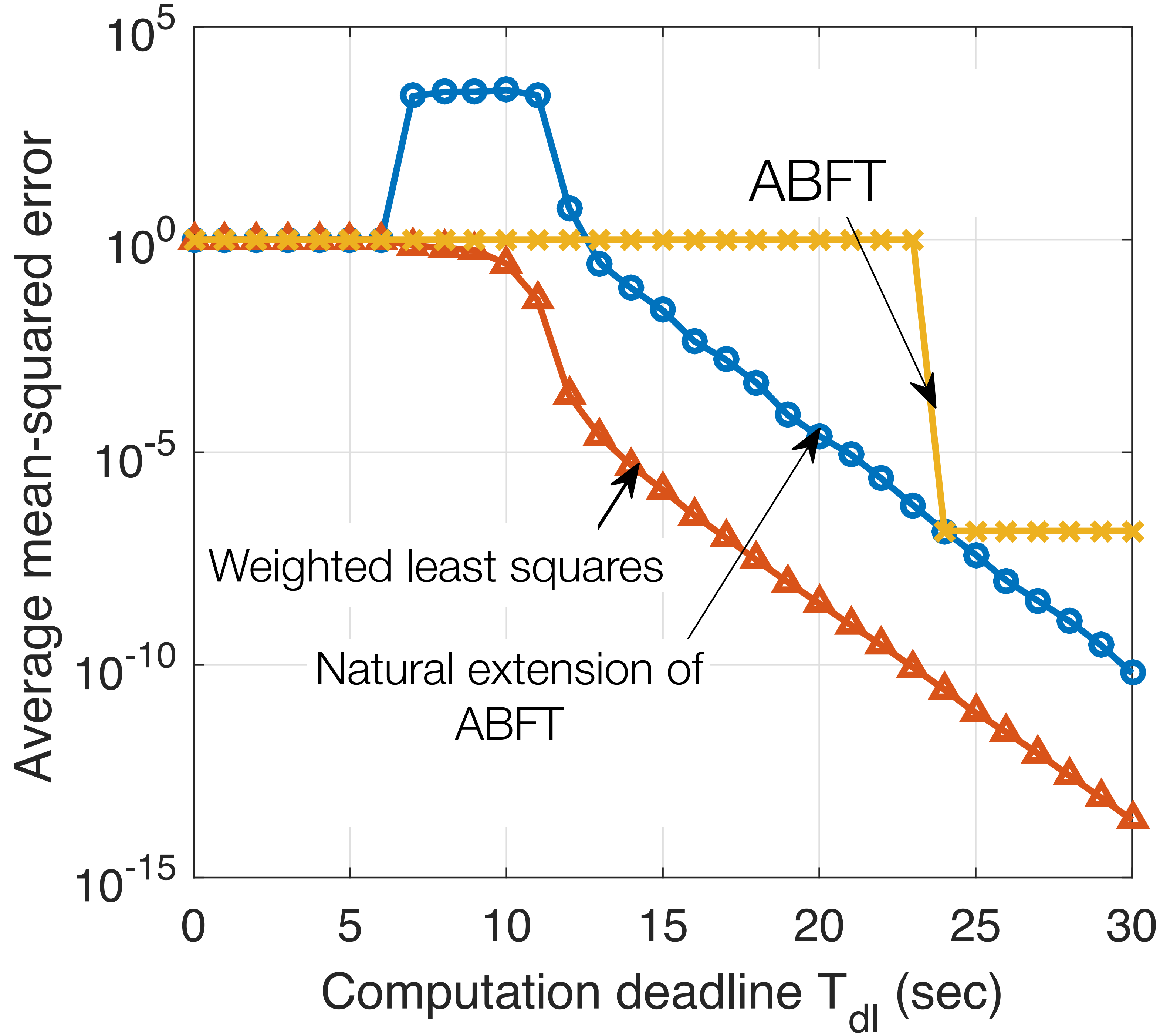
$$\mathbf{Y}_{N\times P}^{(T_{\text{dl}})} = [\mathbf{y}_1^{(l_1)},\ldots,\mathbf{y}_P^{(l_P)}].$$

- Post Processing (Decoding)

$$\widehat{\mathbf{X}}^\top = (\mathbf{G}\boldsymbol{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\boldsymbol{\Lambda}^{-1}(\mathbf{Y}^{(T_{\text{dl}})})^\top$$

Similar to the "weighted least-square" solution.

Weighted least squares outperforms competition; Degrades gracefully with early deadline



Summary thus far…

ABFT $\subsetneq$ Coded computation

New codes, new problems, new analyses, converses

But, we need to be careful in lit-searching ABFT literature

Next: small processors

Break!

Questions/comments?
Your favorite computation problem?

Preview of Part II: Small Processors

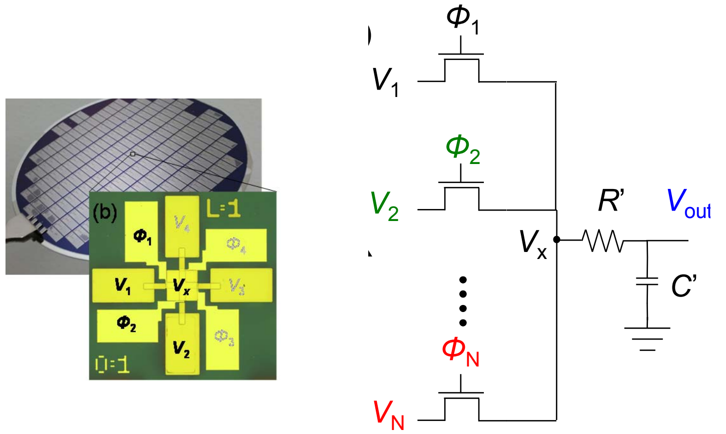
Controlling error propagation with small processors/gates
- No central processor to distribute/aggregate

Encoding/decoding also have errors

Part II: “Small processors”

has so far received relatively less attention

What are small processors?



1) Logic gates

e.g. Dot product “nanofunction” in graphene
[Pop, Shanbhag, Blaauw labs ’15-’16]

2) Analog “Nanofunctions” and beyond CMOS devices

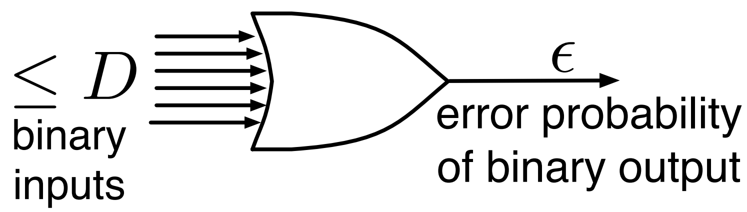
3) Processors with limited memory (i.e., ALL processors are small!)
- can’t assume that processor memory increases with problem size

Synthesize large reliable computations using small processors?

What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

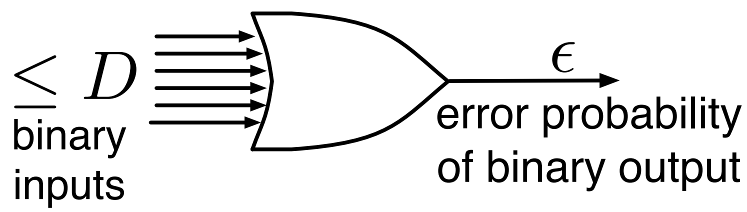
a) Info-dissipation in noisy circuits:



What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

a) Info-dissipation in noisy circuits:

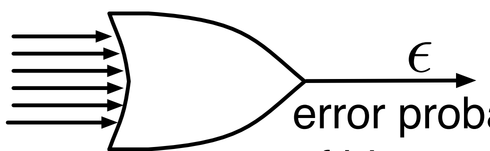


Noisy circuits
built with noisy gates

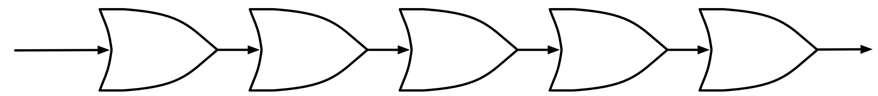
What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

<u>a) Info-dissipation in noisy circuits</u>:



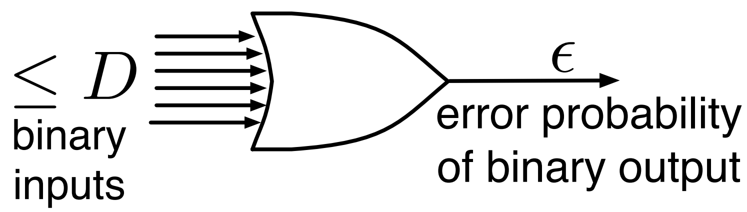
Noisy circuits
built with noisy gates



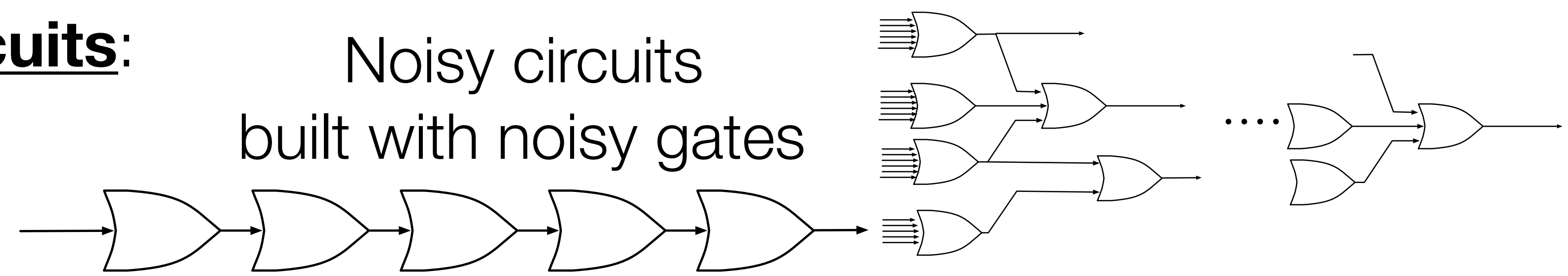
What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

a) Info-dissipation in noisy circuits:



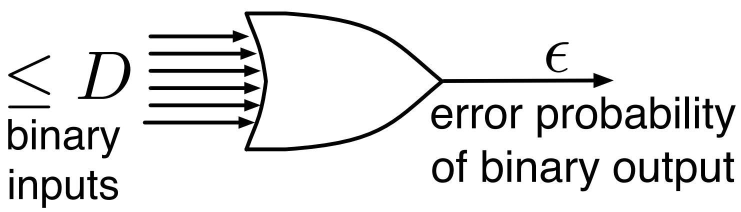
Noisy circuits
built with noisy gates



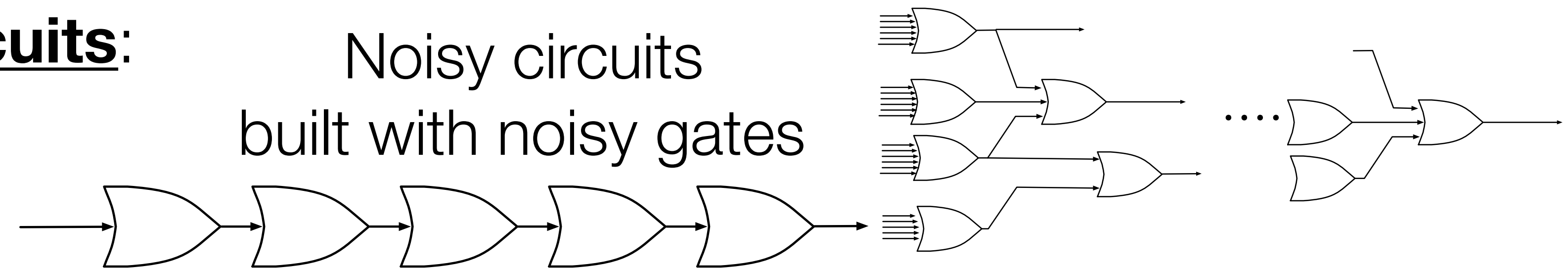
What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

<u>a) Info-dissipation in noisy circuits</u>:



Noisy circuits
built with noisy gates

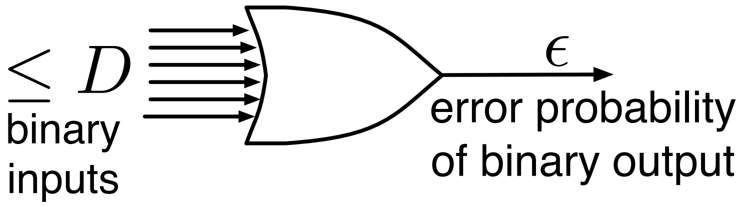


$$X \;-\; Y \overset{BSC(\epsilon)}{-} Z$$

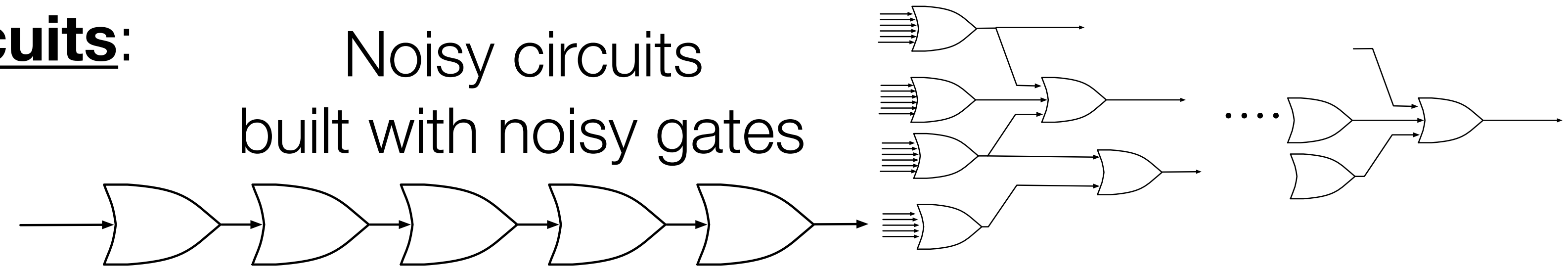
What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

a) Info-dissipation in noisy circuits:



Noisy circuits
built with noisy gates



$$X - Y \overset{BSC(\epsilon)}{-} Z$$

Classical Data-Processing Inequality

$$\frac{I(X;Z)}{I(X;Y)} \leq 1$$

"Strong" Data-Processing Inequality

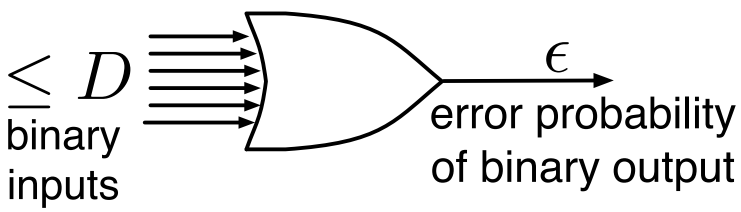
$$\frac{I(X;Z)}{I(X;Y)} \leq f(\epsilon) < 1$$

[Pippenger '88]
[Evans, Schulman '99][Erkip, Cover '98]
[Polayanskiy, Wu '14]
[Anantharam, Gohari, Nair, Kamath '14]
[Raginsky '14]

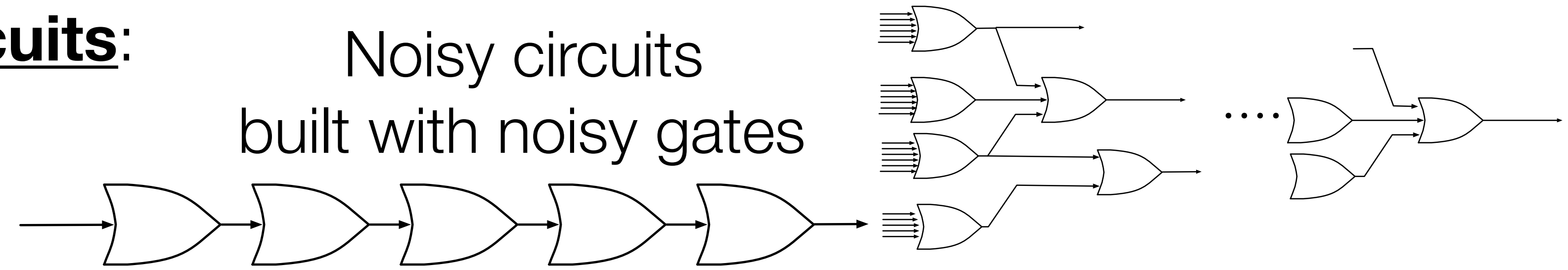
What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

<u>a) Info-dissipation in noisy circuits</u>:



Noisy circuits
built with noisy gates



$X - Y \overset{BSC(\epsilon)}{-} Z$

<u>Classical Data-Processing Inequality</u>

$$\frac{I(X;Z)}{I(X;Y)} \leq 1$$

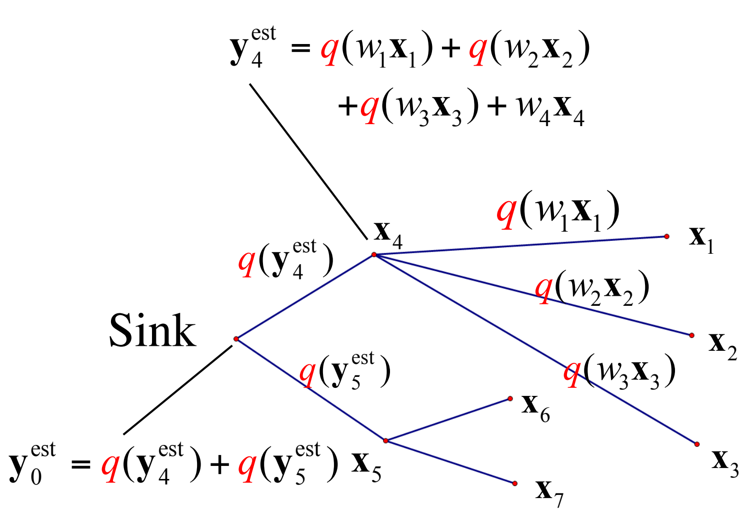
<u>"Strong" Data-Processing Inequality</u>

$$\frac{I(X;Z)}{I(X;Y)} \leq f(\epsilon) < 1$$

[Pippenger '88]
[Evans, Schulman '99][Erkip, Cover '98]
[Polayanskiy, Wu '14]
[Anantharam, Gohari, Nair, Kamath '14]
[Raginsky '14]

<u>b) Distortion accumulation with quantization noise</u>

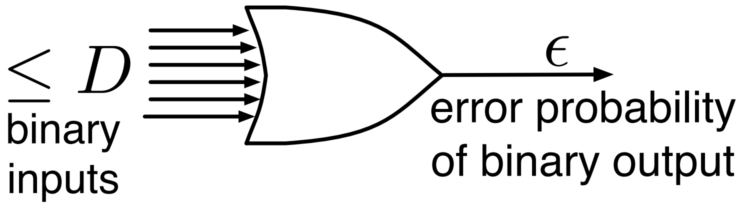
(e.g. in "data summarization", consensus, etc.)

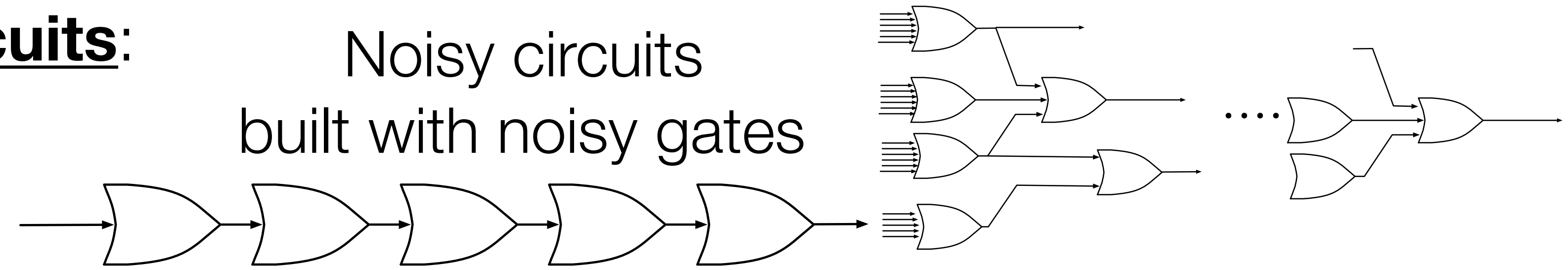


What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

a) Info-dissipation in noisy circuits:





$$X - Y \xrightarrow{BSC(\epsilon)} Z$$

Classical Data-Processing Inequality

$$\frac{I(X;Z)}{I(X;Y)} \leq 1$$

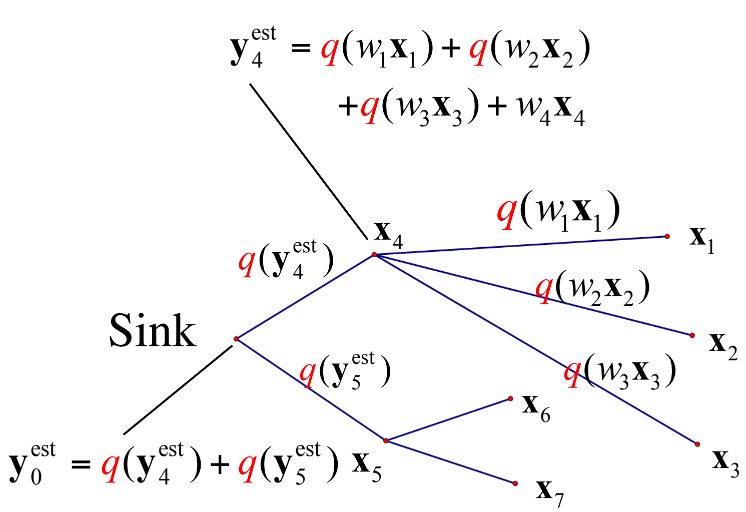
"Strong" Data-Processing Inequality

$$\frac{I(X;Z)}{I(X;Y)} \leq f(\epsilon) < 1$$

[Pippenger '88]
[Evans, Schulman '99][Erkip, Cover '98]
[Polayanskiy, Wu '14]
[Anantharam, Gohari, Nair, Kamath '14]
[Raginsky '14]

b) Distortion accumulation with quantization noise

(e.g. in "data summarization", consensus, etc.)



An application of cut-set bound:
[Cuff, Su, El Gamal '09]

$$R_{i \to PN(i)} \geq \frac{1}{2}\log_2 \frac{\sigma_{S_i}^2}{D_i}$$

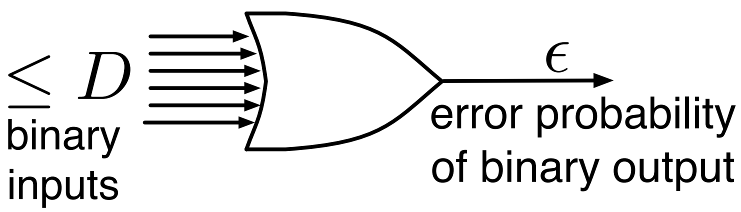
Incremental-distortion bound:
[Yang, Grover, Kar IEEE Trans IT'17]

$$R_{i \to PN(i)} \geq \frac{1}{2}\log_2 \frac{\sigma_{S_i}^2}{\Delta D_i} - O(D_i^{1/2})$$

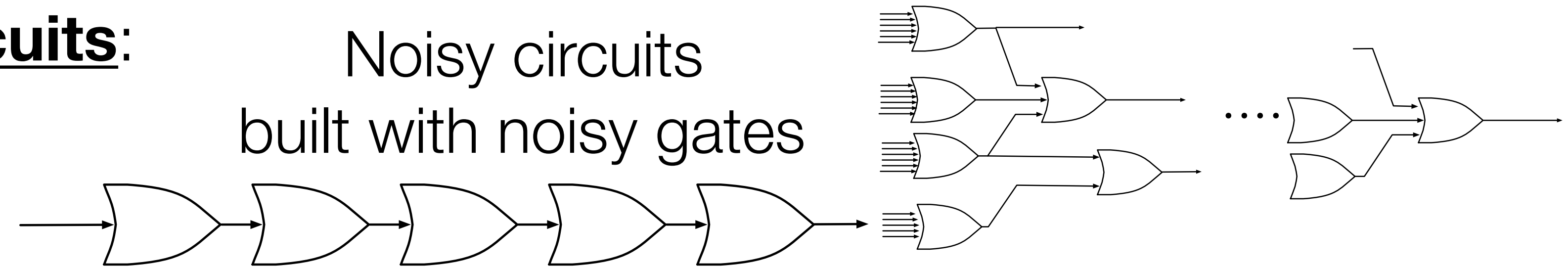
What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

<u>a) Info-dissipation in noisy circuits</u>:



Noisy circuits
built with noisy gates



$X \quad - \quad Y \; \overset{BSC(\epsilon)}{-} \; Z$

<u>Classical Data-Processing Inequality</u>

$$\frac{I(X;Z)}{I(X;Y)} \leq 1$$

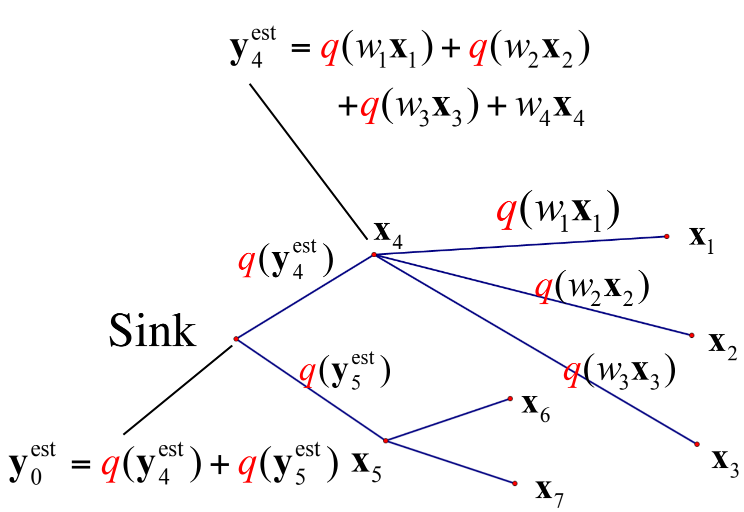
<u>“Strong” Data-Processing Inequality</u>

$$\frac{I(X;Z)}{I(X;Y)} \leq f(\epsilon) < 1$$

[Pippenger ’88]
[Evans, Schulman ’99][Erkip, Cover ’98]
[Polayanskiy, Wu ’14]
[Anantharam, Gohari, Nair, Kamath ’14]
[Raginsky ’14]

<u>b) Distortion accumulation with quantization noise</u>

(e.g. in “data summarization”, consensus, etc.)



An application of cut-set bound:
[Cuff, Su, El Gamal ’09]

$$R_{i\to PN(i)} \geq \frac{1}{2}\log_2 \frac{\sigma_{S_i}^2}{D_i}$$

Incremental-distortion bound:
[Yang, Grover, Kar IEEE Trans IT’17]

$$R_{i\to PN(i)} \geq \frac{1}{2}\log_2 \frac{\sigma_{S_i}^2}{\Delta D_i} - O(D_i^{1/2})$$

tighter by an unbounded factor

What is fundamentally new in small processor computing?

1) Errors accumulate; information dissipates

2) Decoding, and possibly encoding, also error prone

Essential to analyze decoding/encoding costs in noisy computation:
there may be no conceptual analog of Shannon capacity in computing problems
[Grover et al.'07-'15][Grover ISIT'14][Blake, Kschischang '15,'16]

Error-prone *decoding* (often message-passing for LDPCs)
[Taylor '67][Hadjicostis, Verghese '05][Vasic et al. '07-'13][Varshney '11][Grover, Palaiyanur, Sahai '10]
[Huang, Yao, Dolecek '14][Gross et al. '13]**[Vasic et al.'16]**

Error-prone *encoding* [Yang, Grover, Kar '14][Dupraz et al. '15]
- see also erasure version [Hachem, Wang, Fragouli, Diggavi '13]

Can we compute M x V reliably using error-prone gates? Is it even possible?

We'll next discuss this for 1) Gates; 2) Processors

M x V on noisy gates: the basics

$$[r_1, r_2, \ldots, r_K] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} \quad A \quad \\ \text{Linear transform} \end{bmatrix}_{L \times K}$$

Output — Input — Linear transform

M x V on noisy gates: the basics

$$[r_1, r_2, \ldots, r_K] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} \quad & A & \quad \\ & \text{Linear transform} & \end{bmatrix}_{L \times K}$$

Output

Input

$$[x_1, x_2, \ldots, x_N] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} \quad & A & \quad \end{bmatrix}_{L \times K} \begin{bmatrix} \mathbb{I}_{K \times K} | \mathbb{P} \\ \mathbb{G} \end{bmatrix}_{K \times N}$$

Coded output

Input

Systematic generator matrix

M x V on noisy gates: the basics

$$[r_1, r_2, \ldots, r_K] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} A \end{bmatrix}_{L \times K}$$

Output — Input — Linear transform

$$[x_1, x_2, \ldots, x_N] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} A \end{bmatrix}_{L \times K} \begin{bmatrix} \mathbb{I}_{K \times K} | \mathbb{P} \\ \mathbb{G} \end{bmatrix}_{K \times N}$$

Coded output — Input — Systematic generator matrix

$\widetilde{\mathbb{G}}$: coded generator matrix

M x V on noisy gates: the basics

$$[r_1, r_2, \ldots, r_K] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} & A & \\ & \text{Linear transform} & \end{bmatrix}_{L\times K}$$

Output   Input

$$[x_1, x_2, \ldots, x_N] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} & A & \\ & & \end{bmatrix}_{L\times K} \begin{bmatrix} \mathbb{I}_{K\times K} \vert \mathbb{P} \\ \mathbb{G} \end{bmatrix}_{K\times N}$$

Coded output   Input   Systematic generator matrix

$\widetilde{\mathbb{G}}$: coded generator matrix

Note: rows of $\widetilde{\mathbb{G}}$ are also codewords of $\mathbb{G}$!

M x V on noisy gates: the basics

$$[r_1, r_2, \ldots, r_K] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} \quad A \quad \end{bmatrix}_{L \times K}$$

Output — Input — Linear transform

$$[x_1, x_2, \ldots, x_N] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} \quad A \quad \end{bmatrix}_{L \times K} \begin{bmatrix} \mathbb{I}_{K \times K} | \mathbb{P} \\ \mathbb{G} \end{bmatrix}_{K \times N}$$

Coded output — Input — Systematic generator matrix

$\widetilde{\mathbb{G}}$: coded generator matrix

Note: rows of $\widetilde{\mathbb{G}}$ are also codewords of $\mathbb{G}$!

Encoded computation: multiply s with $\widetilde{\mathbb{G}}$

Decoding: use parity-check matrix H for $\mathbb{G}$

M x V on noisy gates: the basics

$$[r_1, r_2, \ldots, r_K] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} & A & \\ & \text{Linear transform} & \end{bmatrix}_{L\times K}$$

Output　　Input

$$[x_1, x_2, \ldots, x_N] = [s_1, s_2, \ldots, s_L] \begin{bmatrix} & A & \end{bmatrix}_{L\times K} \begin{bmatrix} \mathbb{I}_{K\times K} | \mathbb{P} \\ \mathbb{G} \end{bmatrix}_{K\times N}$$

Coded output　　Input　　Systematic generator matrix

$\widetilde{\mathbb{G}}$: coded generator matrix

Note: rows of $\widetilde{\mathbb{G}}$ are also codewords of $\mathbb{G}$!

PRECOMPUTED NOISELESSLY

Encoded computation: multiply s with $\widetilde{\mathbb{G}}$

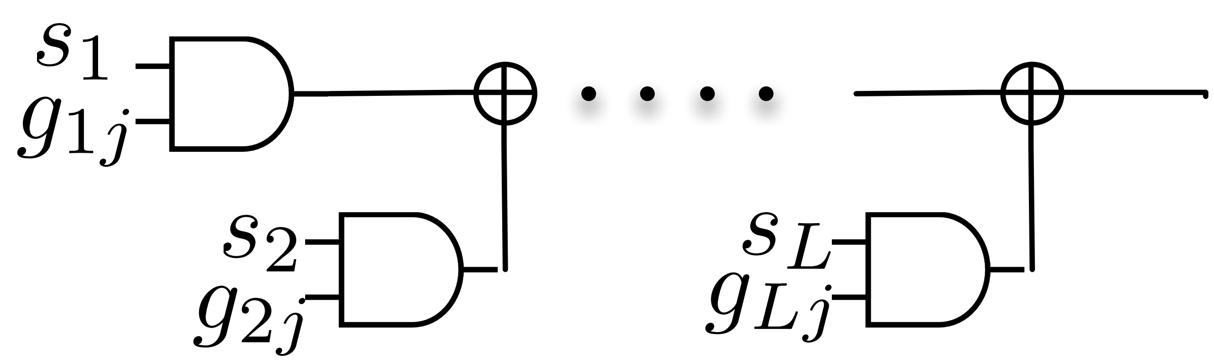
Decoding: use parity-check matrix H for $\mathbb{G}$

A difficulty with this approach: error propagation

Naive computation of $\mathbf{x} = \mathbf{s}\widetilde{\mathbf{G}}$ requires computing $x_i = \sum_j s_j g_{ji}$

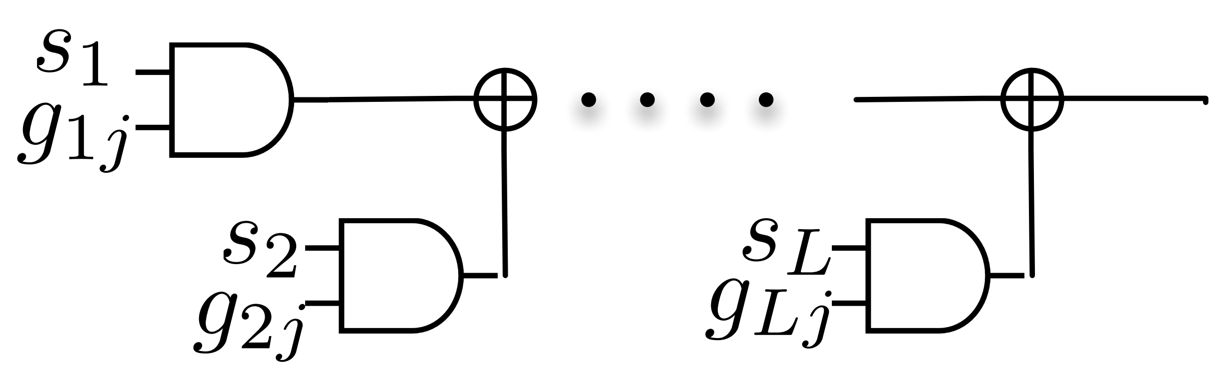
A difficulty with this approach: error propagation

Naive computation of $\mathbf{x} = \mathbf{s}\widetilde{\mathbf{G}}$ requires computing $x_i = \sum_j s_j g_{ji}$



A difficulty with this approach: error propagation

Naive computation of $\mathbf{x} = \mathbf{s}\widetilde{\mathbf{G}}$ requires computing $x_i = \sum_j s_j g_{ji}$



Requiring L AND gates, L-1 XOR gates

Error accumulates! As $L \rightarrow \infty$, each x_i approaches a random coin flip

Addressing error accumulation: a simple observation

source sequence

generator matrix

$$\mathbf{x} = \mathbf{s}\widetilde{\mathbf{G}} = [s_1, s_2, \ldots, s_k] \begin{bmatrix} - - -\widetilde{\mathbf{g}}_1 - - - \\ - - -\widetilde{\mathbf{g}}_2 - - - \\ \cdot \\ \cdot \\ \cdot \\ - - -\widetilde{\mathbf{g}}_k - - - \end{bmatrix}$$

Codeword

Addressing error accumulation: a simple observation

source sequence

generator matrix

$$\mathbf{x} = \mathbf{s}\widetilde{\mathbf{G}} = [s_1, s_2, \ldots, s_k] \begin{bmatrix} ---\widetilde{\mathbf{g}}_1 --- \\ ---\widetilde{\mathbf{g}}_2 --- \\ \cdot \\ \cdot \\ \cdot \\ ---\widetilde{\mathbf{g}}_k --- \end{bmatrix}$$

Codeword

$$= s_1\widetilde{\mathbf{g}}_1 + s_2\widetilde{\mathbf{g}}_2 + \ldots + s_k\widetilde{\mathbf{g}}_k$$

Addressing error accumulation: a simple observation

source sequence

generator matrix

$$\mathbf{x} = \mathbf{s}\widetilde{\mathbf{G}} = [s_1, s_2, \ldots, s_k] \begin{bmatrix} ---\widetilde{\mathbf{g}}_1--- \\ ---\widetilde{\mathbf{g}}_2--- \\ \cdot \\ \cdot \\ \cdot \\ ---\widetilde{\mathbf{g}}_k--- \end{bmatrix}$$

Codeword

$$= \boxed{s_1\widetilde{\mathbf{g}}_1 + s_2\widetilde{\mathbf{g}}_2} + \ldots + s_k\widetilde{\mathbf{g}}_k$$

A valid codeword.
Can be corrected for errors

Addressing error accumulation: a simple observation

source sequence

generator matrix

$$\mathbf{x} = \mathbf{s}\widetilde{\mathbf{G}} = [s_1, s_2, \ldots, s_k] \begin{bmatrix} ---\widetilde{\mathbf{g}}_1--- \\ ---\widetilde{\mathbf{g}}_2--- \\ \cdot \\ \cdot \\ \cdot \\ ---\widetilde{\mathbf{g}}_k--- \end{bmatrix}$$

Codeword

$$= s_1\widetilde{\mathbf{g}}_1 + s_2\widetilde{\mathbf{g}}_2 + \ldots + s_k\widetilde{\mathbf{g}}_k$$

A valid codeword.
Can be corrected for errors

Any correctly computed partial sum is a valid codeword

Addressing error accumulation: a simple observation

source sequence

generator matrix

$$\mathbf{x} = \mathbf{s}\widetilde{\mathbf{G}} = [s_1, s_2, \ldots, s_k] \begin{bmatrix} - - -\tilde{\mathbf{g}}_1 - - - \\ - - -\tilde{\mathbf{g}}_2 - - - \\ \cdot \\ \cdot \\ \cdot \\ - - -\tilde{\mathbf{g}}_k - - - \end{bmatrix}$$

Codeword

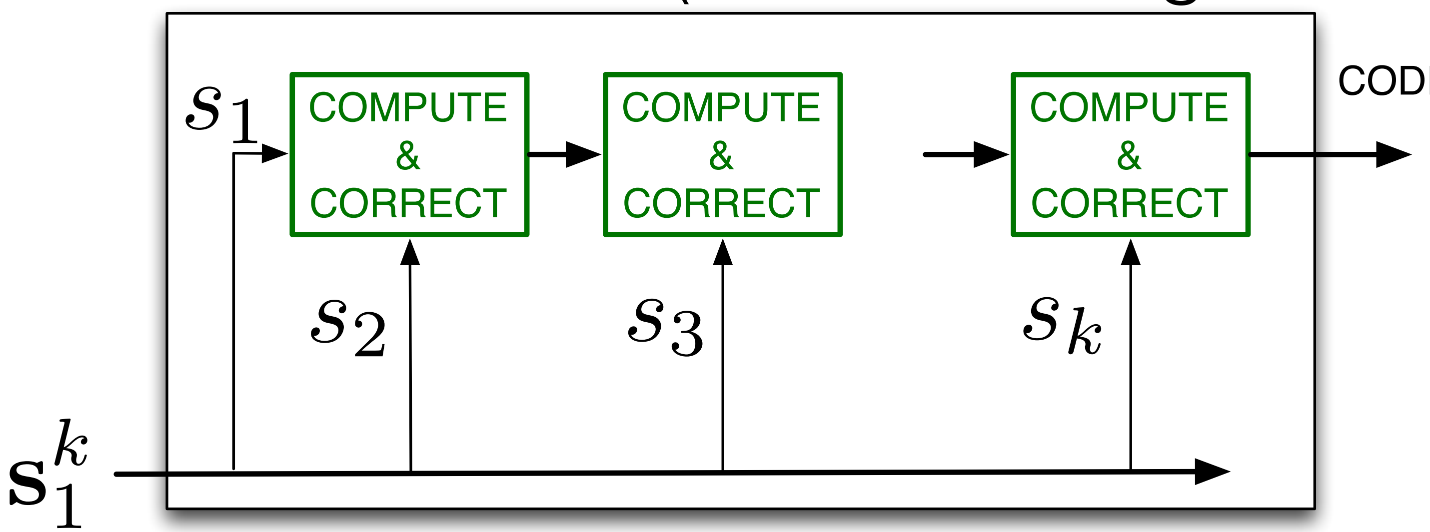
$$= s_1\tilde{\mathbf{g}}_1 + s_2\tilde{\mathbf{g}}_2 + \ldots + s_k\tilde{\mathbf{g}}_k$$

A valid codeword.
Can be corrected for errors

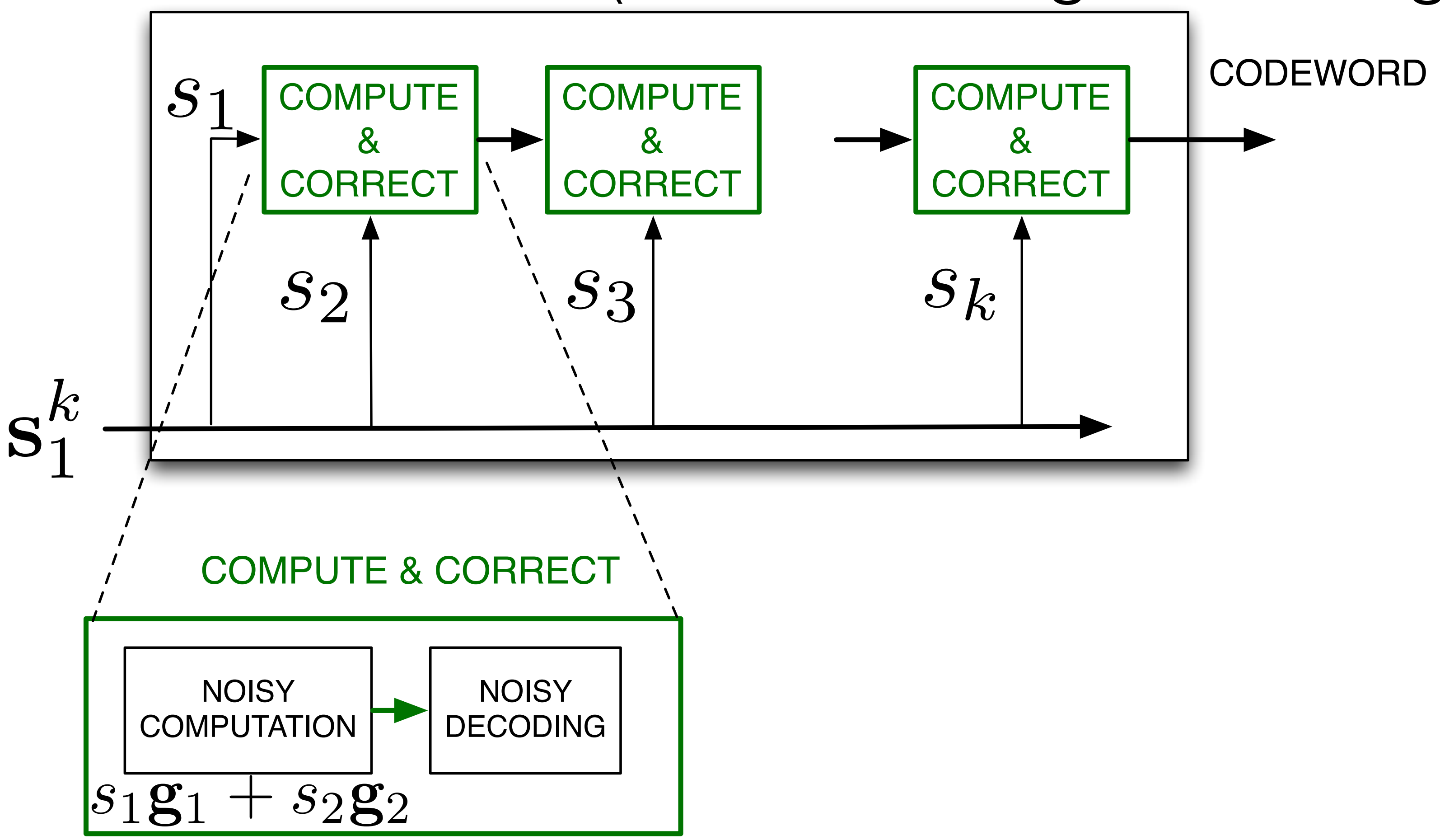
Any correctly computed partial sum is a valid codeword

- possibly correct compute errors by embedding decoders inside encoder
- Use LDPC codes: utilize results on noisy decoding
  (we used [Tabatabaei, Cho, Dolecek ’14])

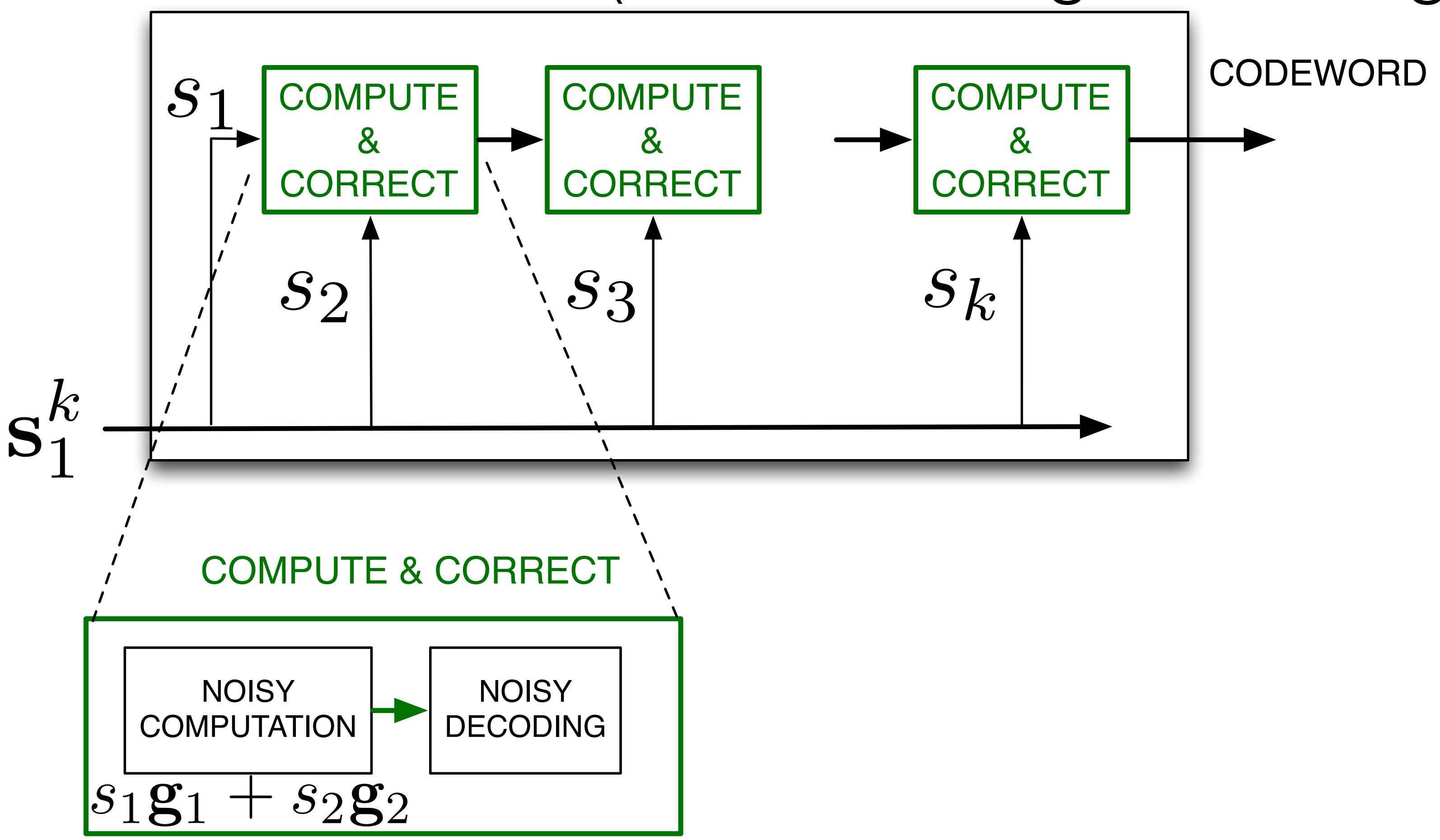
"ENCODED": ENcoded COmputation with Decoders EmbeddeD (with decoding also being noisy)



"ENCODED": ENcoded COmputation with Decoders EmbeddeD (with decoding also being noisy)

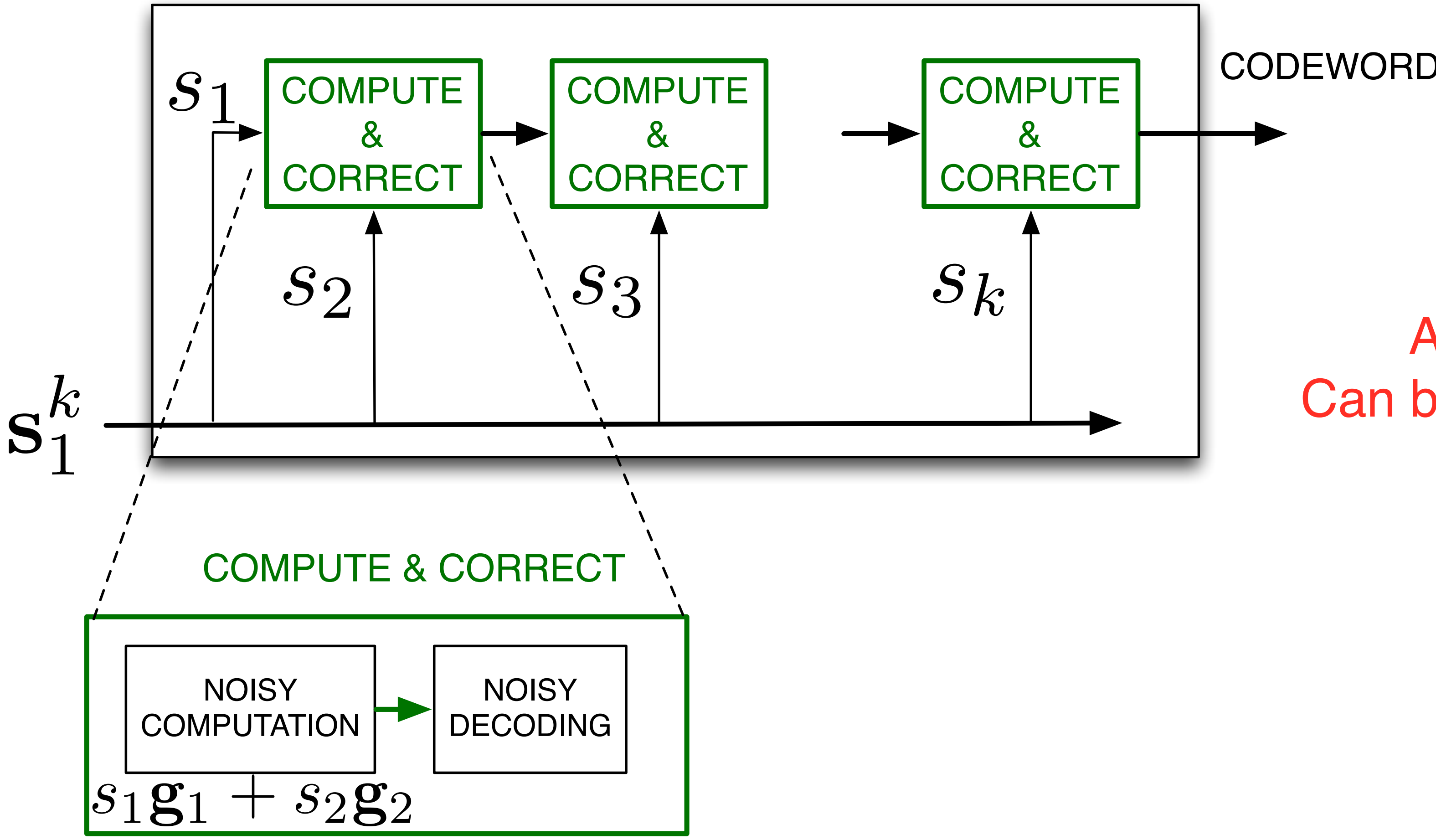


"ENCODED": ENcoded COmputation with Decoders EmbeddeD (with decoding also being noisy)



$$= s_1\widetilde{\mathbf{g}}_1 + s_2\widetilde{\mathbf{g}}_2 + \ldots + s_k\widetilde{\mathbf{g}}_k$$

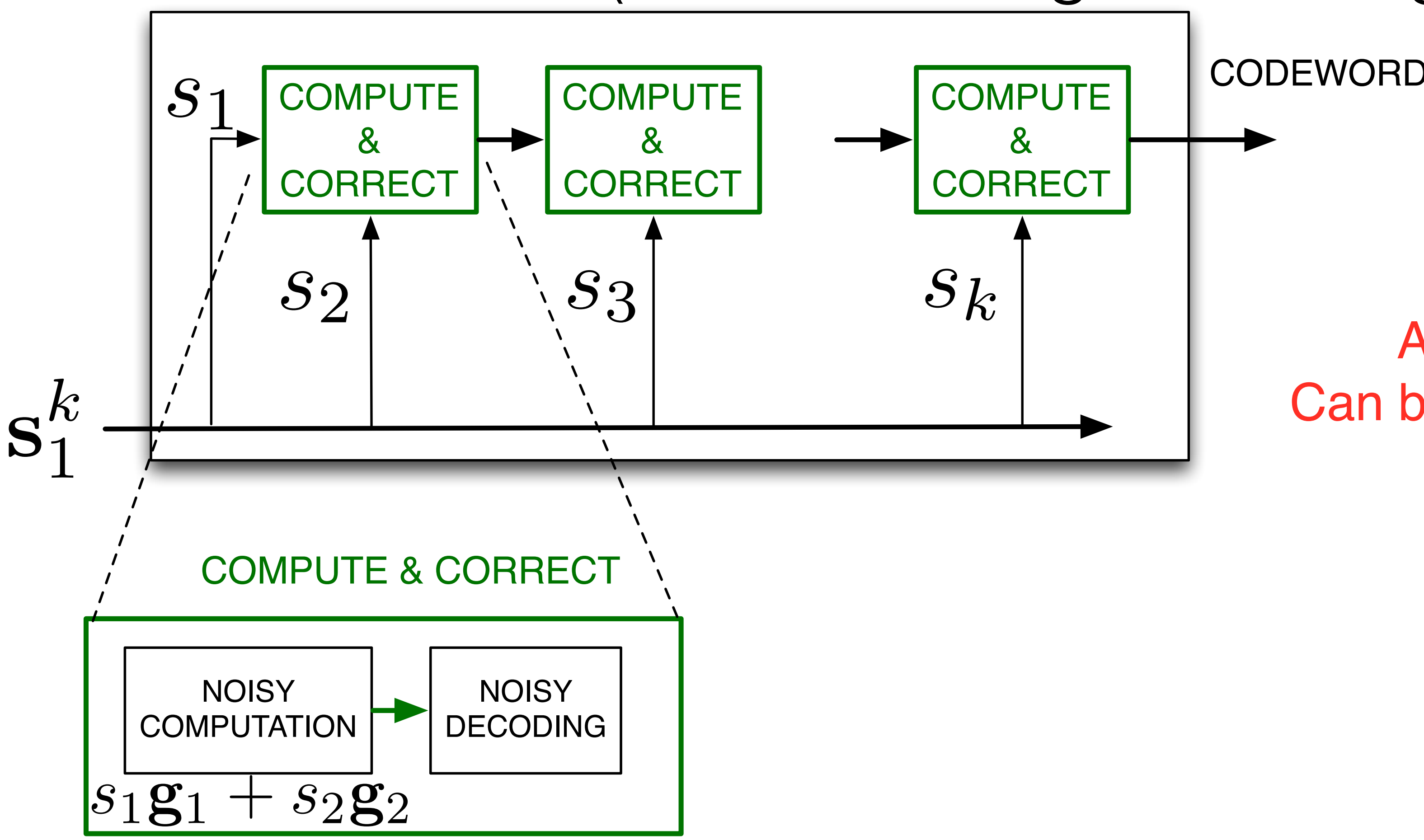
"ENCODED": ENcoded COmputation with Decoders EmbeddeD
(with decoding also being noisy)



$$= s_1\widetilde{\mathbf{g}}_1 + s_2\widetilde{\mathbf{g}}_2 + \ldots + s_k\widetilde{\mathbf{g}}_k$$

A valid codeword.
Can be corrected for errors

“ENCODED”: ENcoded COmputation with Decoders EmbeddeD (with decoding also being noisy)

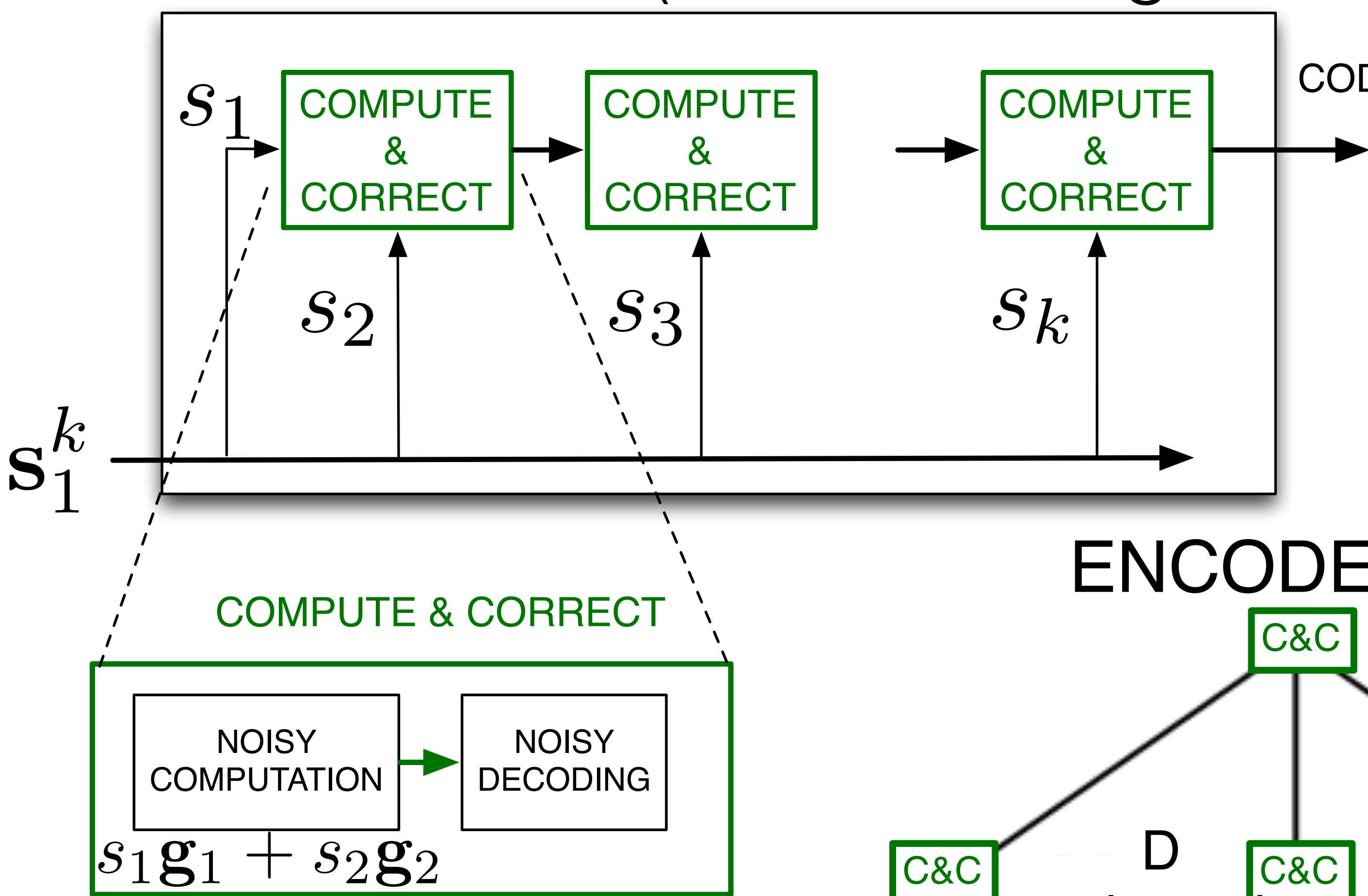


$$= s_1\tilde{\mathbf{g}}_1 + s_2\tilde{\mathbf{g}}_2 + \ldots + s_k\tilde{\mathbf{g}}_k$$

A valid codeword.
Can be corrected for errors

Better yet: ENCODED-Tree

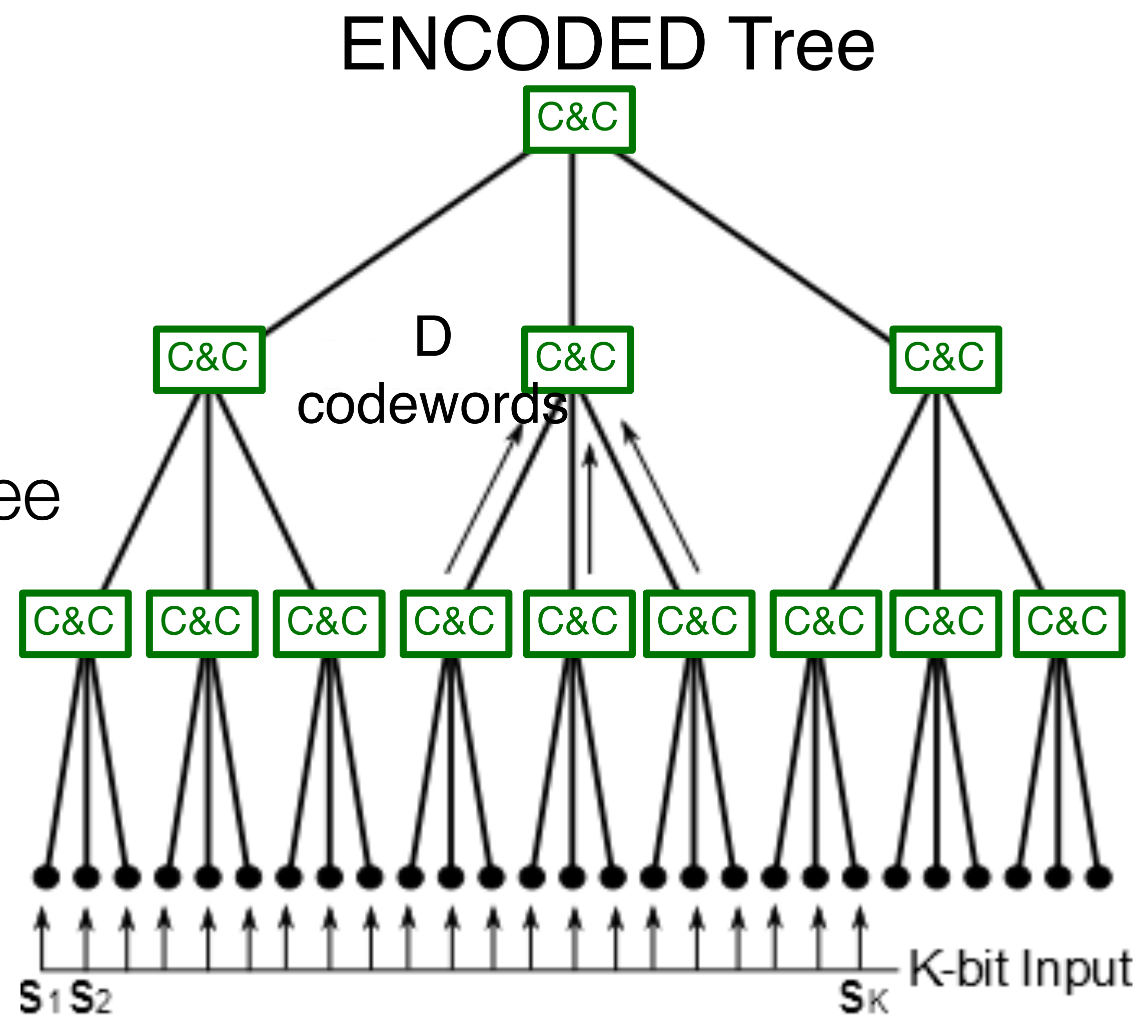
"ENCODED": ENcoded COmputation with Decoders EmbeddeD (with decoding also being noisy)

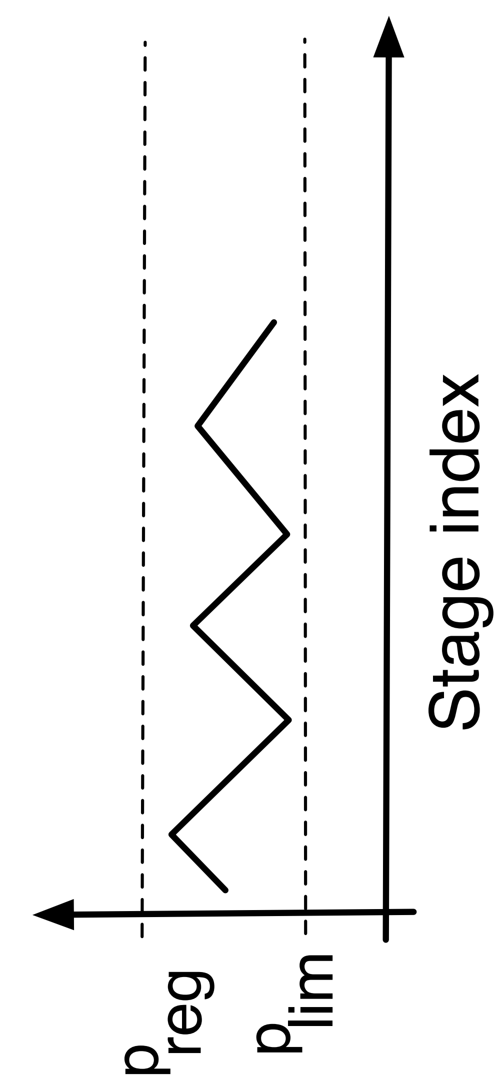


$$= s_1\tilde{\mathbf{g}}_1 + s_2\tilde{\mathbf{g}}_2 + \ldots + s_k\tilde{\mathbf{g}}_k$$

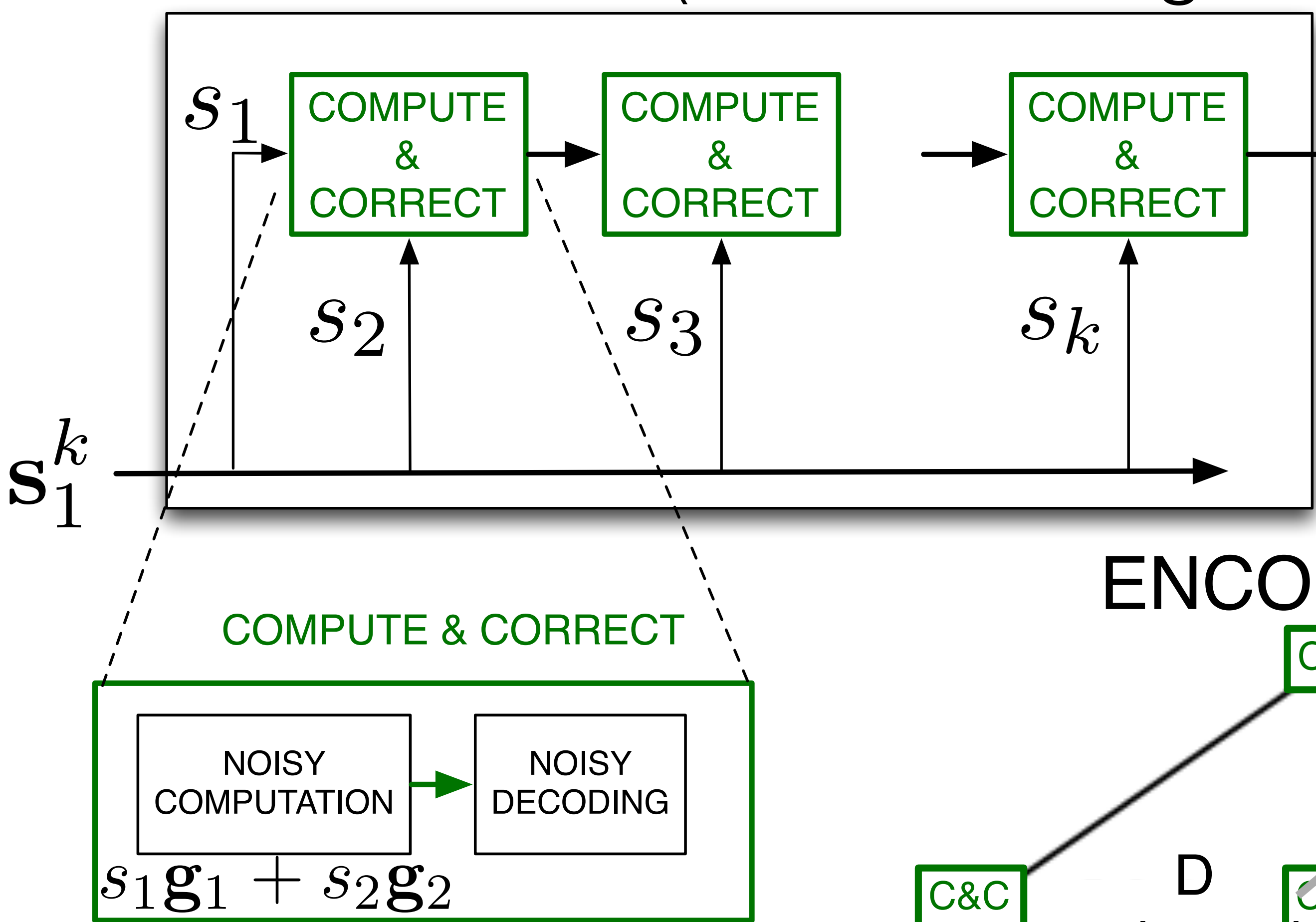
A valid codeword.
Can be corrected for errors

Better yet: ENCODED-Tree





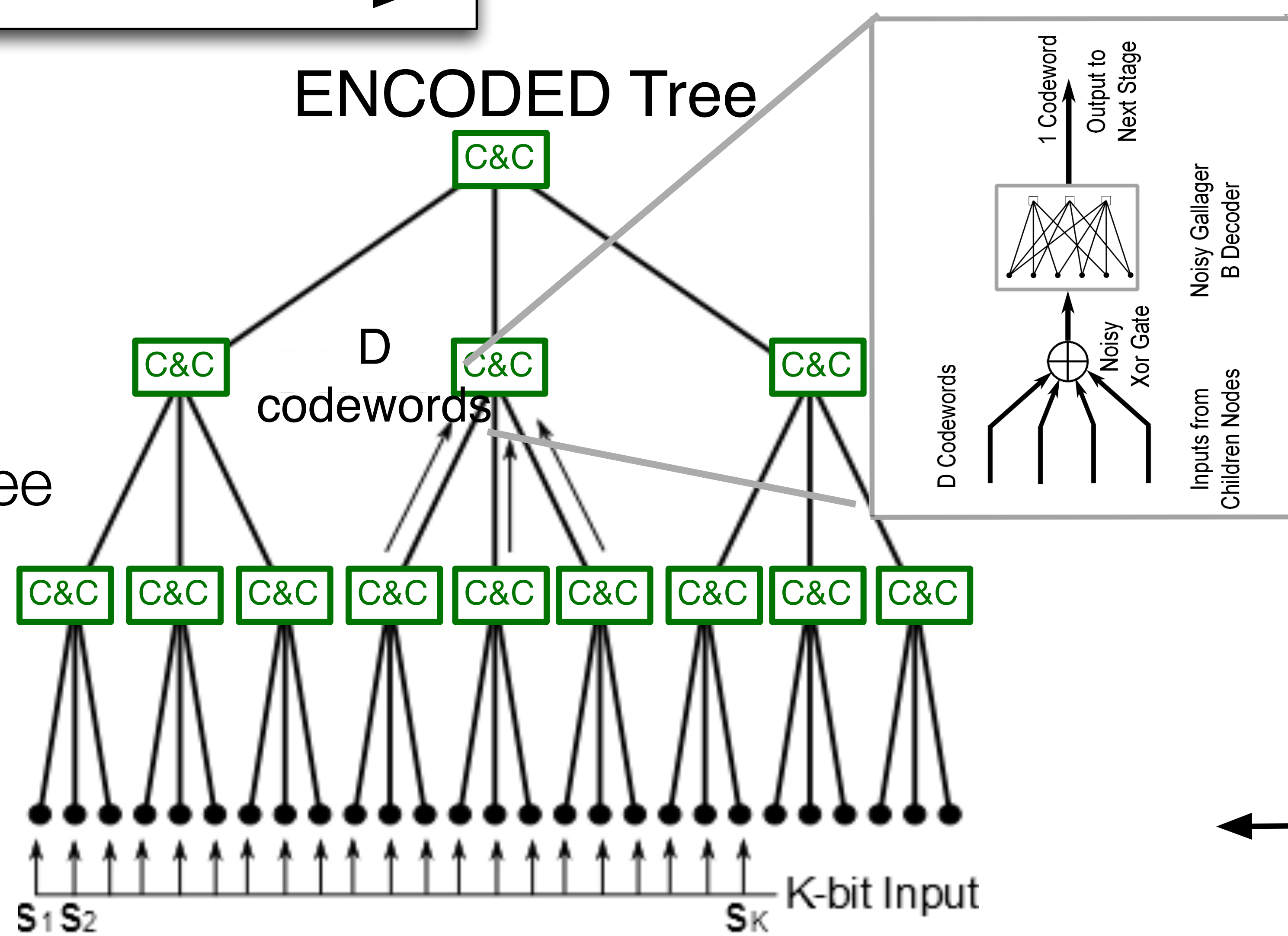
“ENCODED”: ENcoded COmputation with Decoders EmbeddeD (with decoding also being noisy)

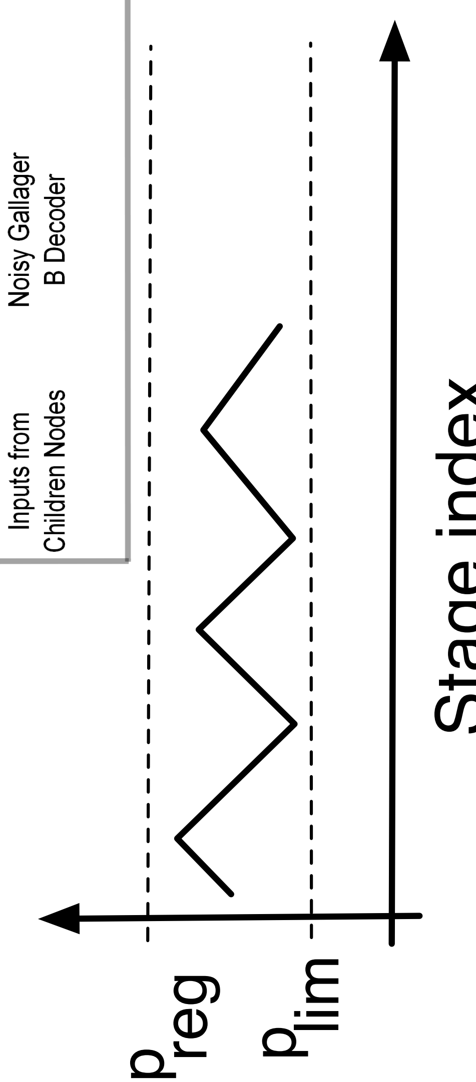


$$= s_1\tilde{\mathbf{g}}_1 + s_2\tilde{\mathbf{g}}_2 + \ldots + s_k\tilde{\mathbf{g}}_k$$

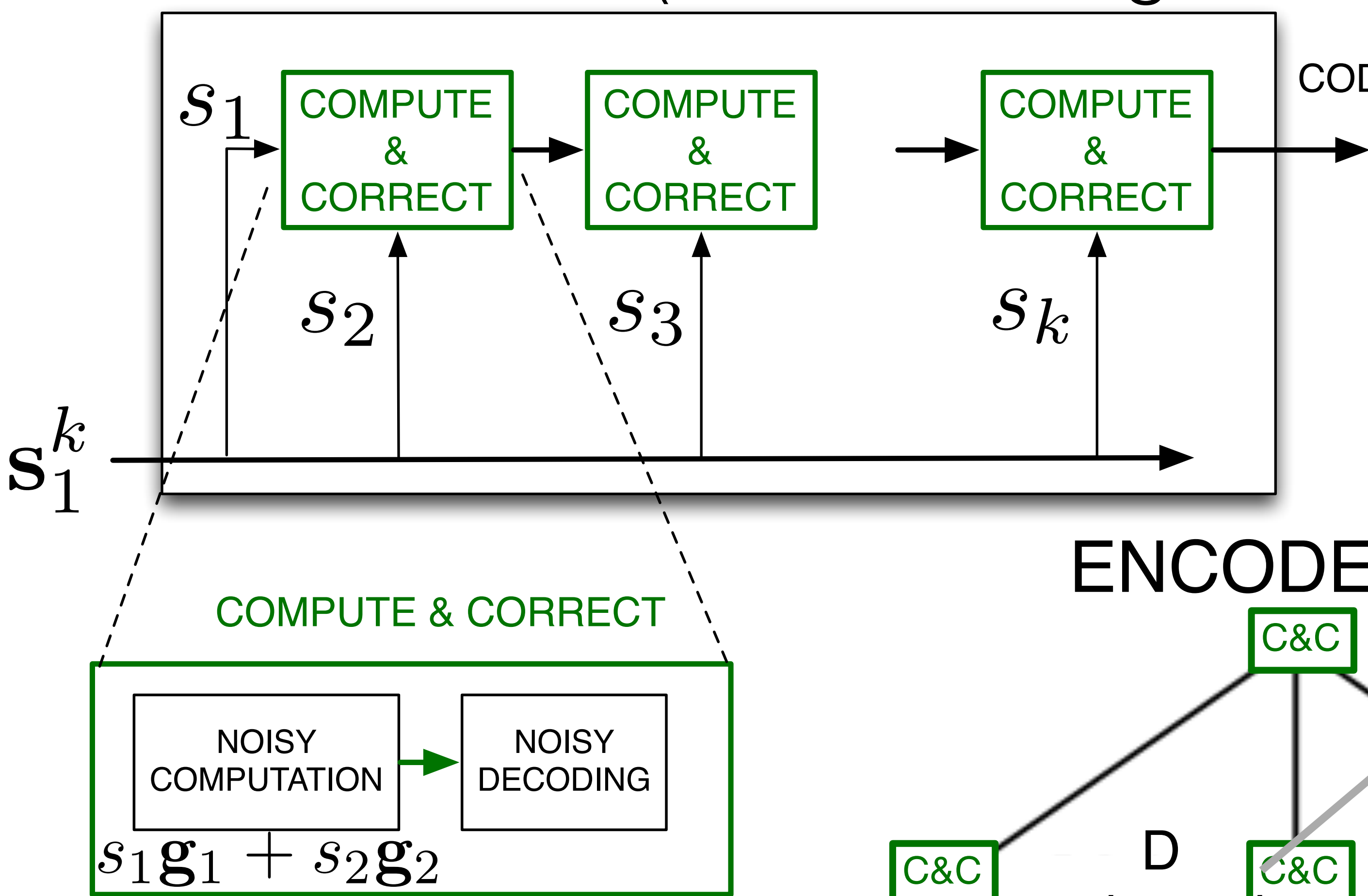
A valid codeword.
Can be corrected for errors

Better yet: ENCODED-Tree





"ENCODED": ENcoded COmputation with Decoders EmbeddeD (with decoding also being noisy)

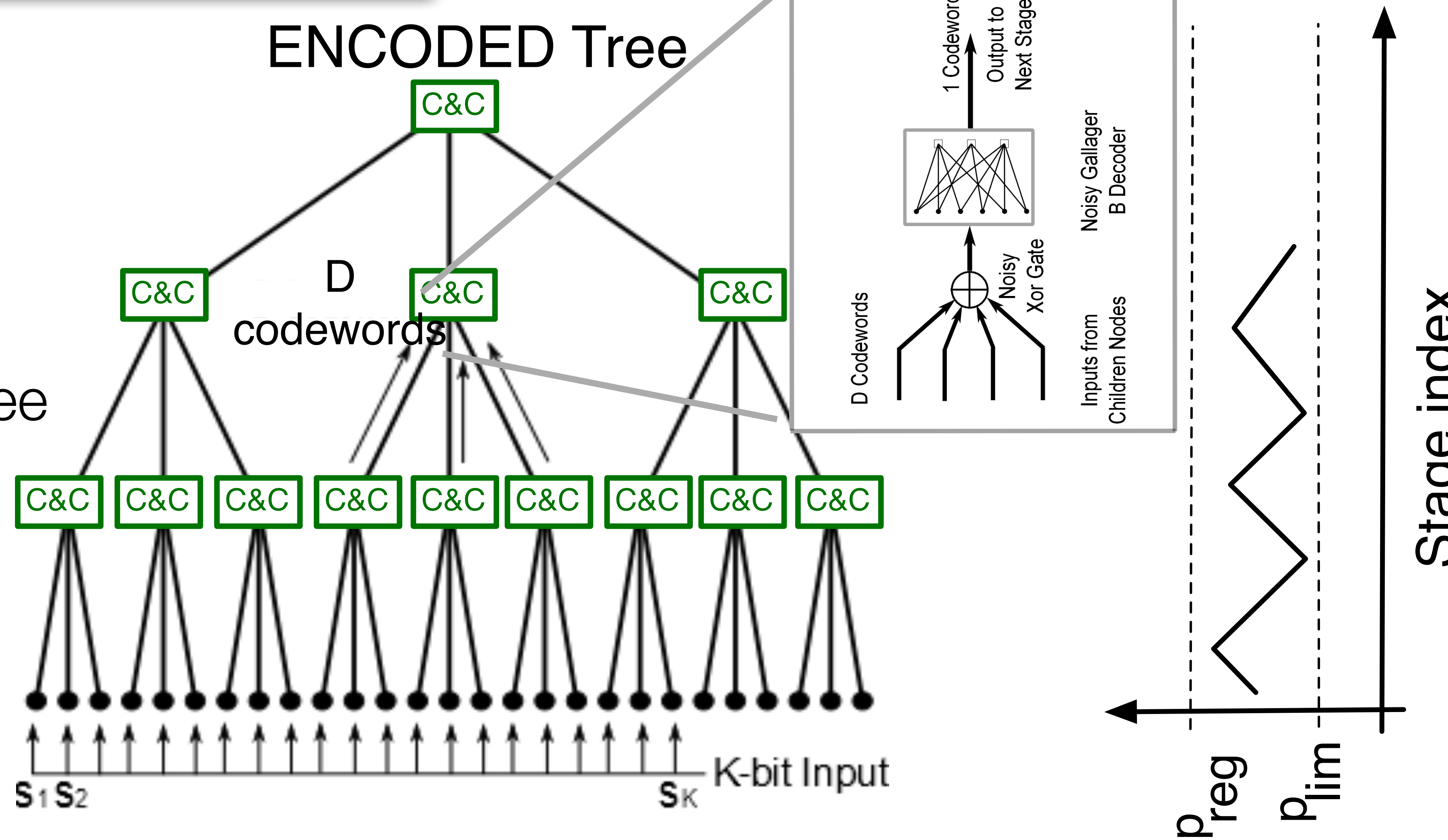


CODEWORD

$$= s_1\tilde{\mathbf{g}}_1 + s_2\tilde{\mathbf{g}}_2 + \ldots + s_k\tilde{\mathbf{g}}_k$$

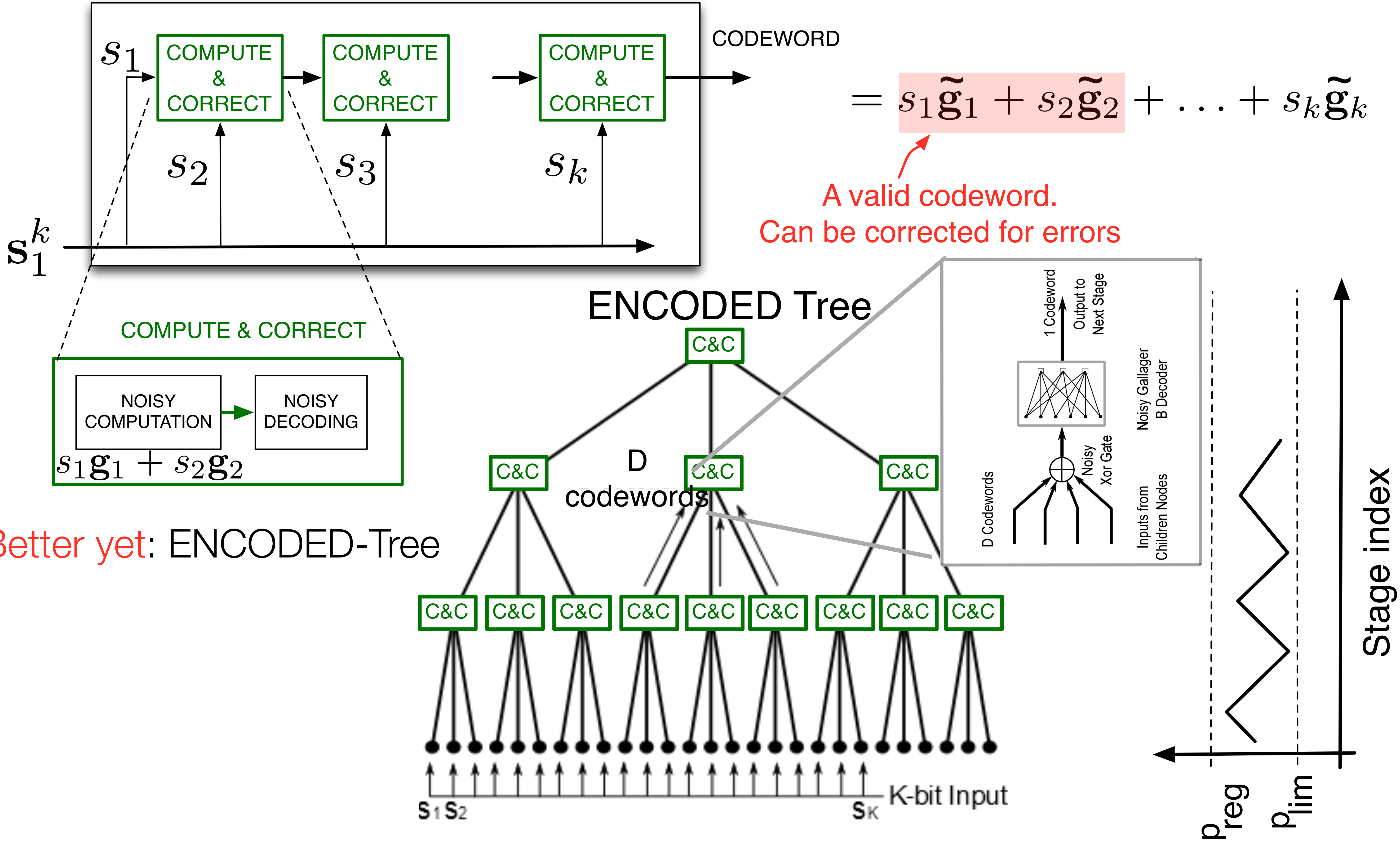
A valid codeword.
Can be corrected for errors

Better yet: ENCODED-Tree



Moral: can overcome info loss on each link by collecting info over many links

“ENCODED”: ENcoded COmputation with Decoders EmbeddeD (with decoding also being noisy)



Better yet: ENCODED-Tree

Moral: can overcome info loss on each link by collecting info over many links

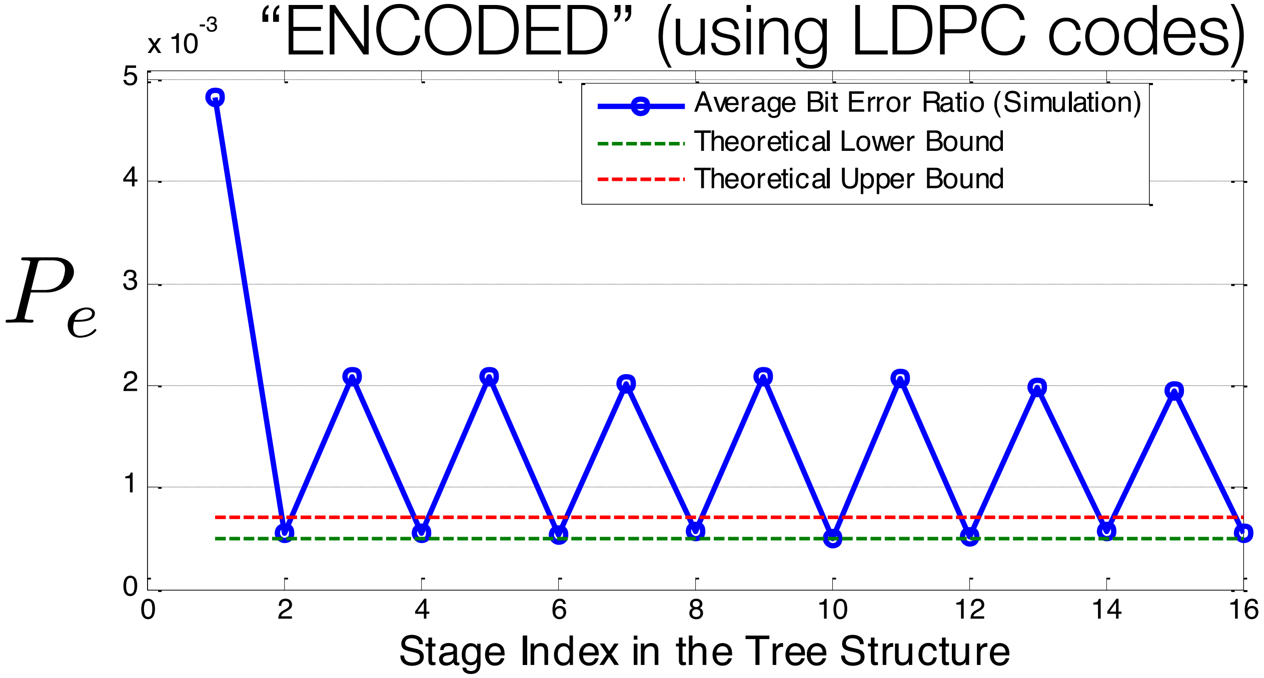
Reflections of a converse [Evans, Schulman ’99] in our achievability

ENCODED vs Uncoded and Repetition

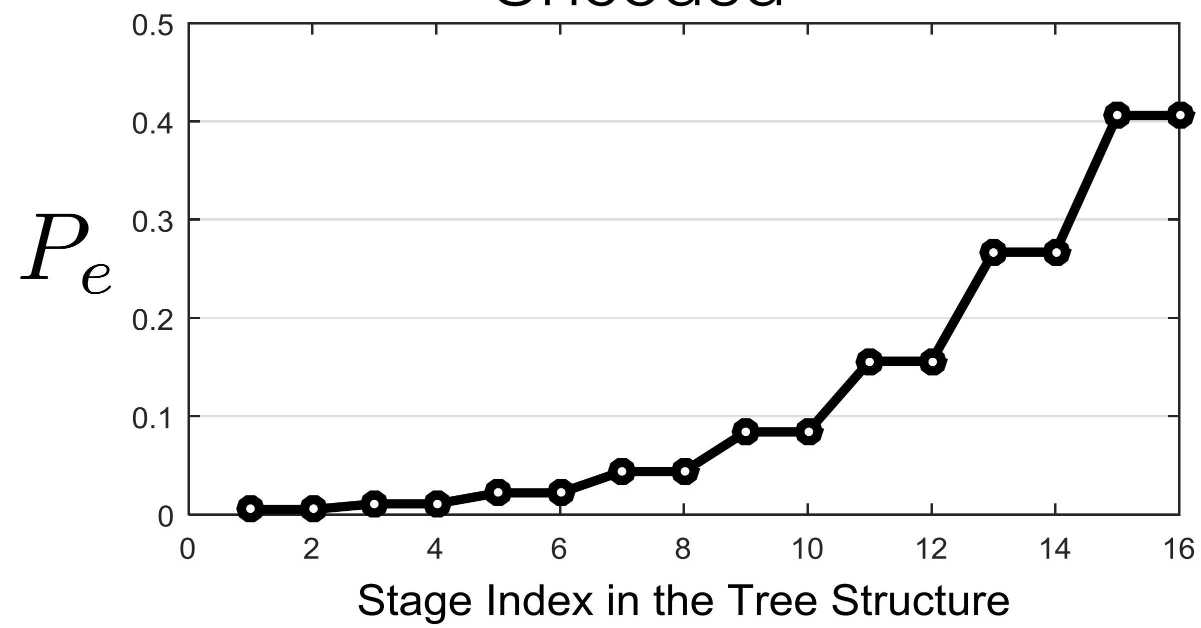
Theorem Error correction with ENCODED-Tree [Yang, Grover, Kar Allerton '14]

LDPC codes of sufficiently large girth can keep errors contained through repeated error suppression

"ENCODED" (using LDPC codes)



"Uncoded"



ENCODED provably requires fewer gates, and less energy than repetition *in scaling sense* [Yang, Grover, Kar *IEEE Trans. Info Theory* '17]

Using general device models, focusing specifically on spintronics

Moral: repeated error-correction can fight information dissipation

Next: How do these insights apply to processors of limited memory (but > 1 gate)?

M x V on small (but reliable) processors

Let's first understand M x V on *reliable* processors

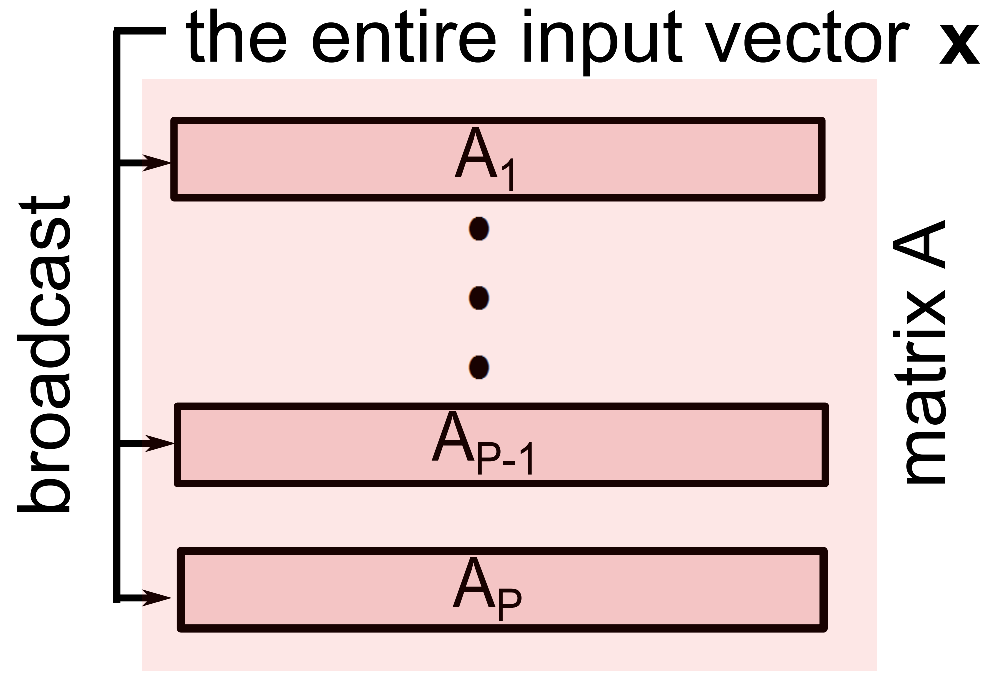
"SUMMA": Scalable Universal Matrix Multiplication Algorithm
- a widely used algorithm [van de Geijn and Watts '95]

M x V on small (but reliable) processors

Let’s first understand M x V on *reliable* processors

“SUMMA”: Scalable Universal Matrix Multiplication Algorithm
- a widely used algorithm [van de Geijn and Watts ’95]

Naive M x V computation (**Ax**)

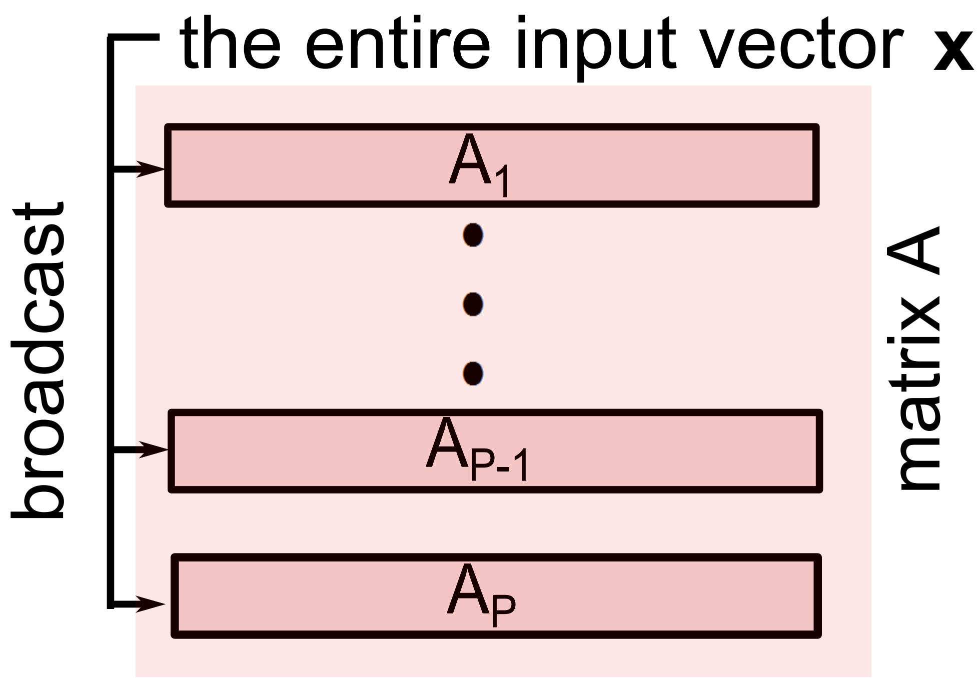


M x V on small (but reliable) processors

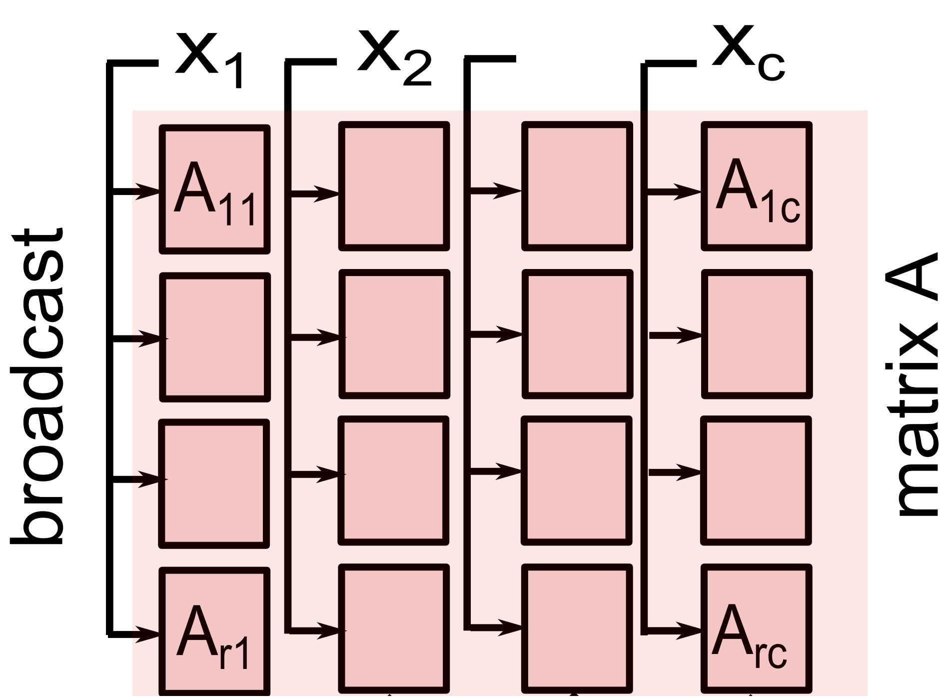
Let's first understand M x V on *reliable* processors

"SUMMA": Scalable Universal Matrix Multiplication Algorithm
- a widely used algorithm [van de Geijn and Watts '95]

Naive M x V computation (**Ax**)



SUMMA **x**=[x_1, x_2, ..., x_c]



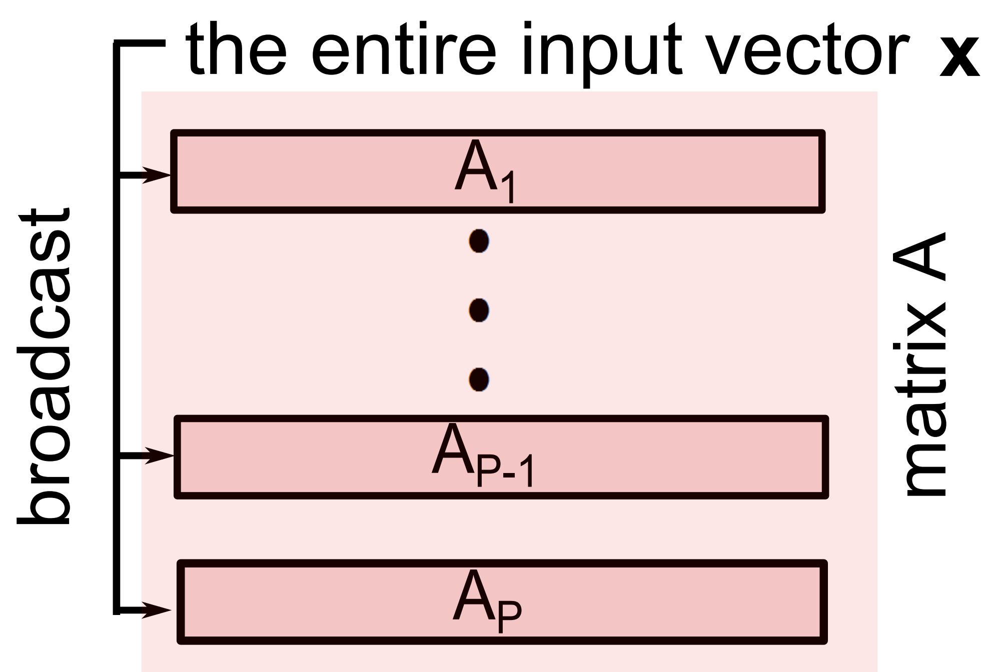
M x V on small (but reliable) processors

Let's first understand M x V on *reliable* processors

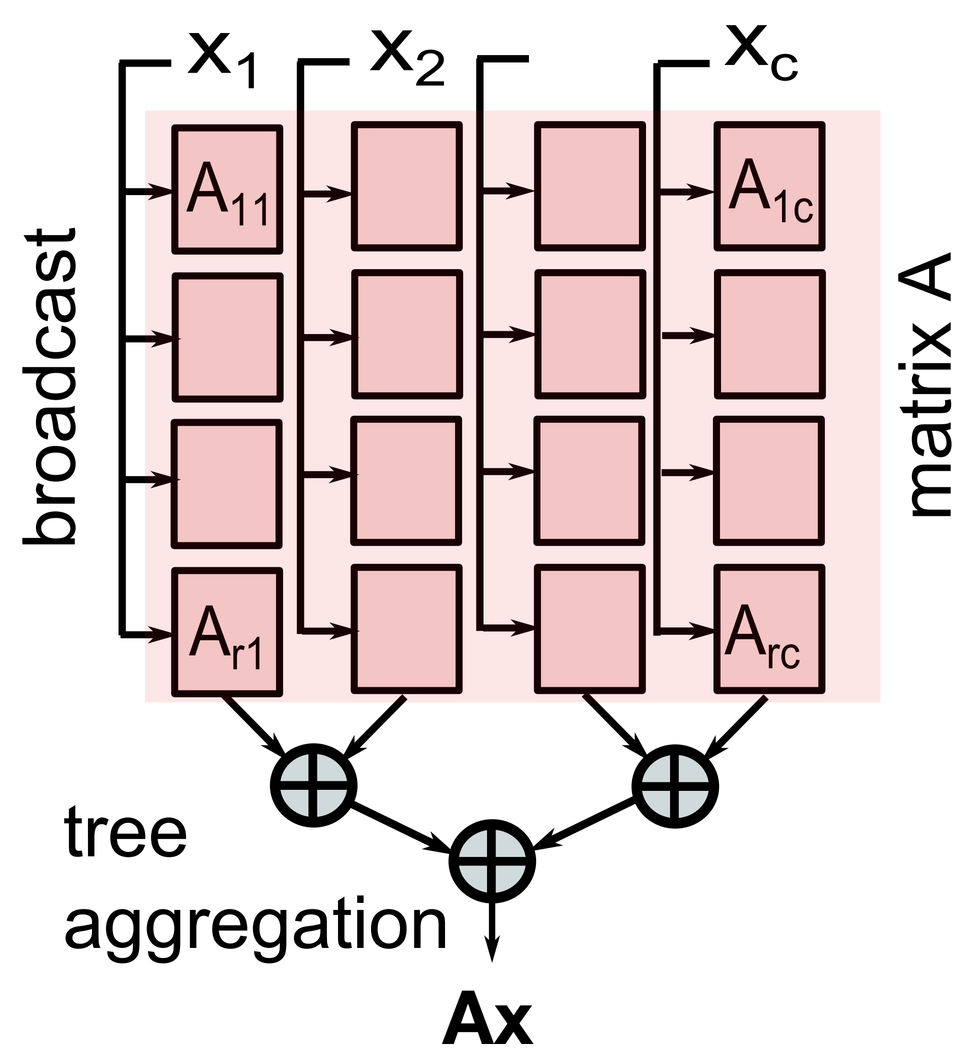
"SUMMA": Scalable Universal Matrix Multiplication Algorithm

- a widely used algorithm [van de Geijn and Watts '95]

Naive M x V computation (**Ax**)

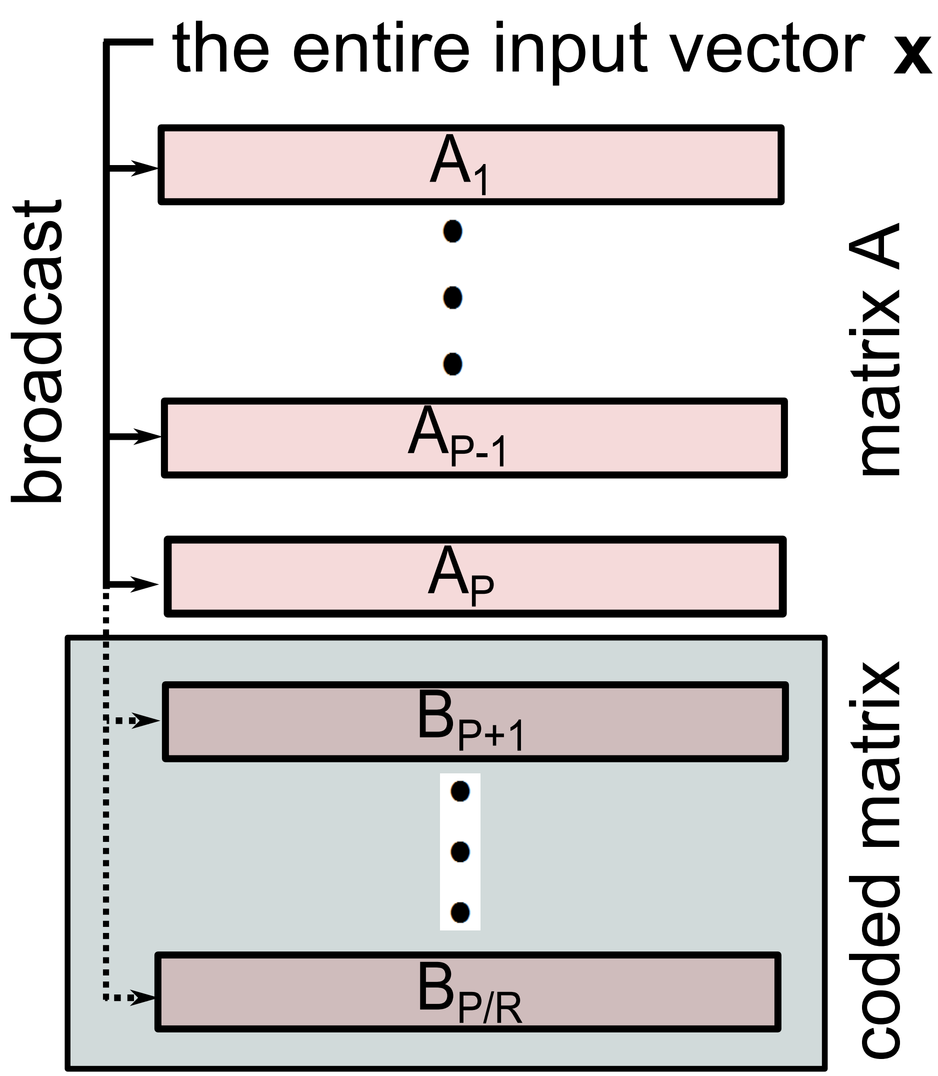


SUMMA **x**=[x_1, x_2, ..., x_c]



Coded SUMMA for M x V on error-prone processors

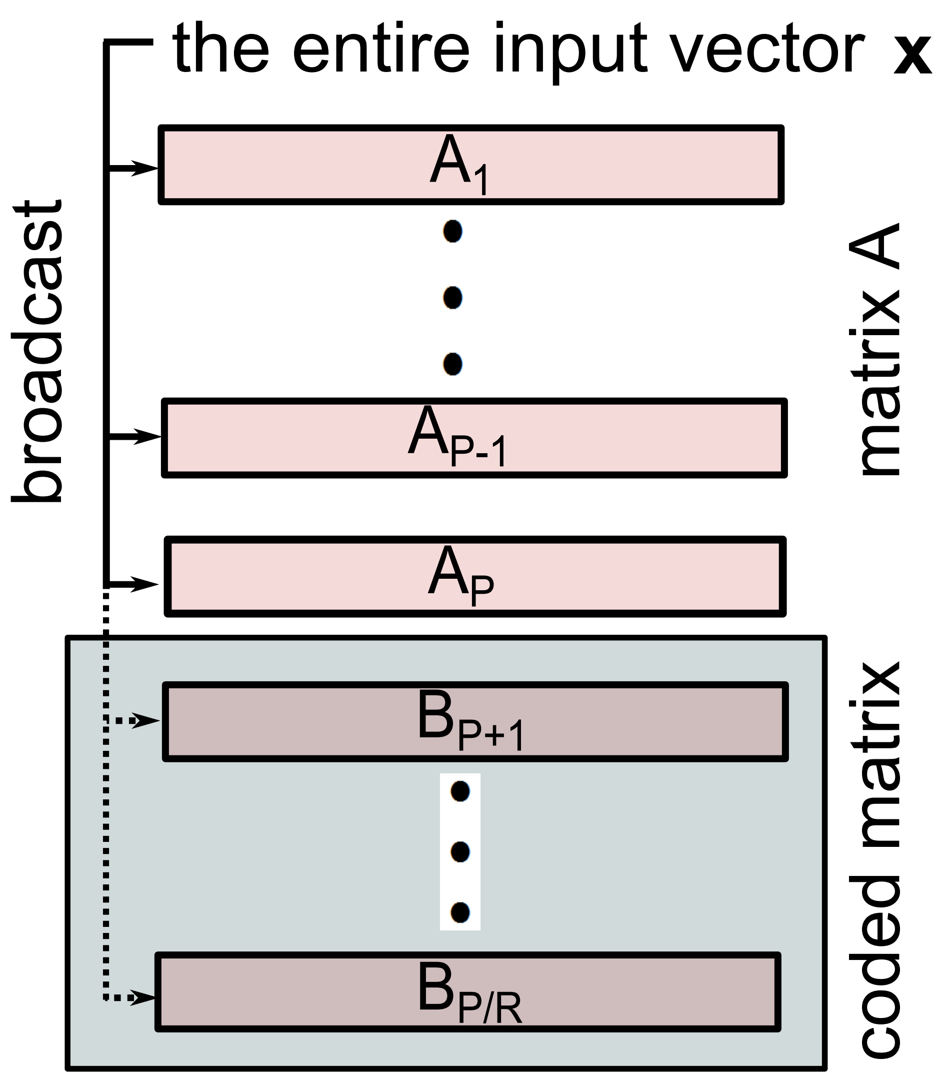
ABFT/MDS coding



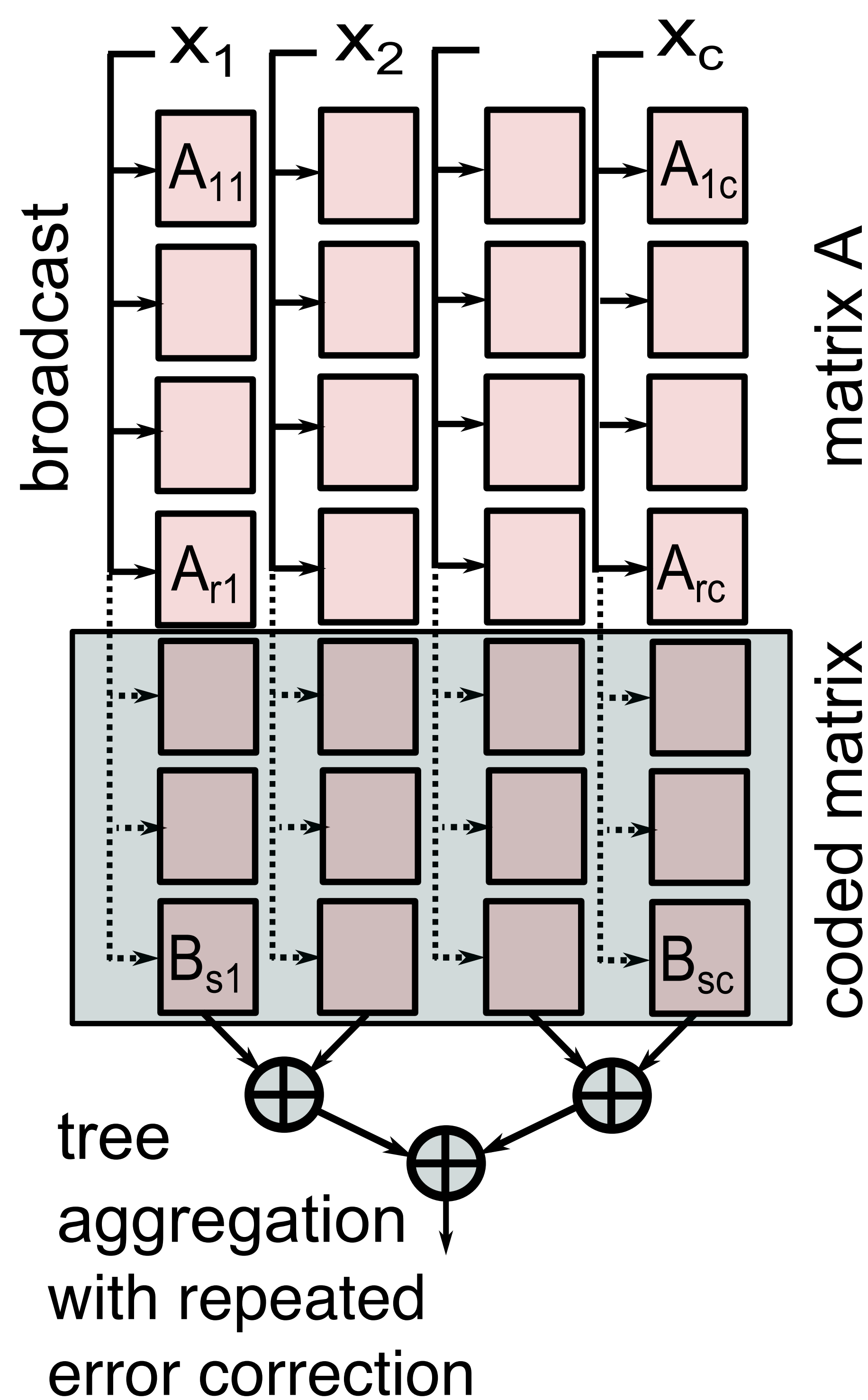
[*in prep.*]

Coded SUMMA for M x V on error-prone processors

ABFT/MDS coding



ENCODED (using LDPC)



[*in prep.*]

Summary of Part II.2

What is fundamentally new in small vs large processors?

0) Memory limitations: necessitate algorithms like SUMMA

1) Errors accumulate; information dissipates

2) Decoding also error prone

Embed (noisy) decoders to repeatedly suppress errors, limiting info dissipation

Coded Map-reduce

Not covered in detail here, but belongs thematically

[Li-Avestimehr-Maddah-Ali 2015]

Map-reduce: A widely used framework for parallelizing a variety of tasks

- Simple to learn, very scalable

Coded Map-reduce

Not covered in detail here, but belongs thematically

[Li-Avestimehr-Maddah-Ali 2015]

Map-reduce: A widely used framework for parallelizing a variety of tasks

- Simple to learn, very scalable

Three phases

Map( )	Data exchange	Reduce( )
First phase	Second phase (usually called *shuffle*)	Third phase

Coded Map-reduce

Not covered in detail here, but belongs thematically

[Li-Avestimehr-Maddah-Ali 2015]

<u>Map-reduce:</u> A widely used framework for parallelizing a variety of tasks

- Simple to learn, very scalable

<u>Three phases</u>

Map( )	Data exchange	Reduce( )
First phase	Second phase (usually called *shuffle*)	Third phase

Idea of coded map reduce

- Introduce redundancy in the map phase
- Exploit information theory ideas (a la coded caching) to minimize communication cost in data exchange
- Save on overall time-to-completion by tuning correctly

Lots of follow up work, exciting area of research!

Broader view of coded distributed computing

<u>Conventional "division of labor" approach:</u>

- design a "good" algorithm with low Turing complexity
- engineer deals with real world costs and imperfections

<u>This tutorial: an information-theoretic approach</u>:

- model system costs and imperfections and,
- derive fundamental information-theoretic limits,
- obtain optimal strategies for these models

Our thanks to…

Collaborators:

- Soummya Kar
- Kishori Konwar
- Nancy Lynch
- Muriel Medard
- Prakash N Moorthy
- Peter Musial
- Zhiying Wang

Student collaborators:
- Rami Ali
- Jeremy Bai
- Malhar Chaudhari
- Sanghamitra Dutta
- Mohammad Fahim
- Farzin Haddadpour
- Haewon Jeong
- Yaoqing Yang

Help with talk and slides:

- Mohammad Ali Maddah Ali
- Salman Avestimehr
- Alex Dimakis
- Gauri Joshi
- Kangwook Lee
- Ramtin Pedarsani

Funding sources:

National Science Foundation (NSF)

SONIC center of the Semiconductor Research Corporation

Appendices/Backup slides

Weak scaling:
Number of processors scales with problem size
- constant computational workload per processor

Strong scaling:
Problem size fixed!
- finding the “sweet-spot” in number of processors
- too many processors => high comm overhead
- too few => not enough parallelization

Related: gate-level errors
- error/fault-tolerant computing

Related problem:
Minimizing total power in communication systems

New goal: Design a P_{total} -efficient code

(errors only in the channel;
encoding/decoding noiseless)

$$P_{total} = P_T + P_{enc} + P_{dec}$$

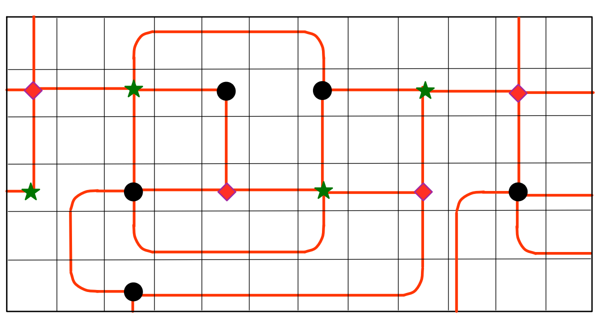
Related problem:
Minimizing total power in communication systems

New goal: Design a P_{total}-efficient code

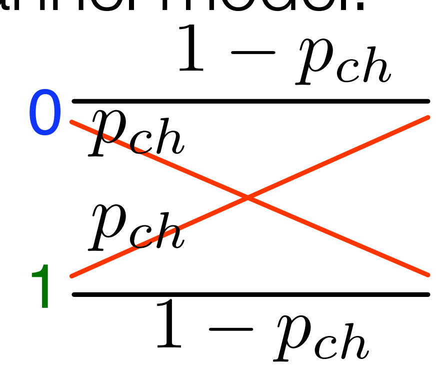
$$P_{total} = P_T + P_{enc} + P_{dec}$$

(errors only in the channel; encoding/decoding noiseless)

Circuit implementation model:



Channel model:



$$p_{ch} = \mathbb{Q}\left(\sqrt{\frac{\eta P_T}{\frac{N_0}{2}}}\right)$$

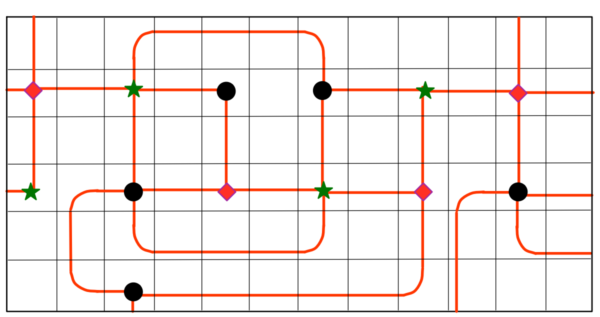
Related problem:
Minimizing total power in communication systems

New goal: Design a P_{total} -efficient code

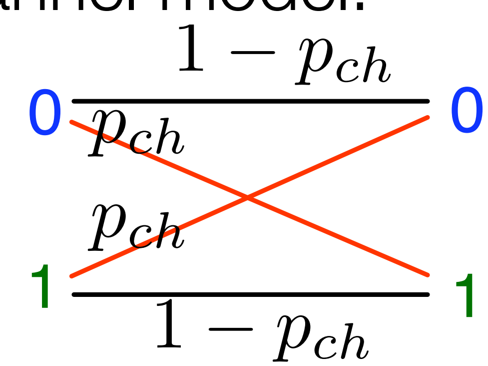
$$P_{total} = P_T + P_{enc} + P_{dec}$$

(errors only in the channel; encoding/decoding noiseless)

Circuit implementation model:

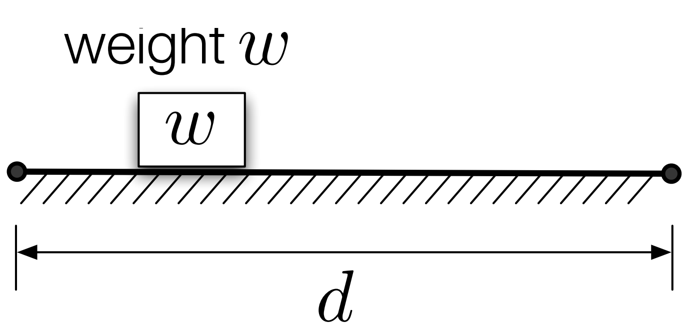


Channel model:

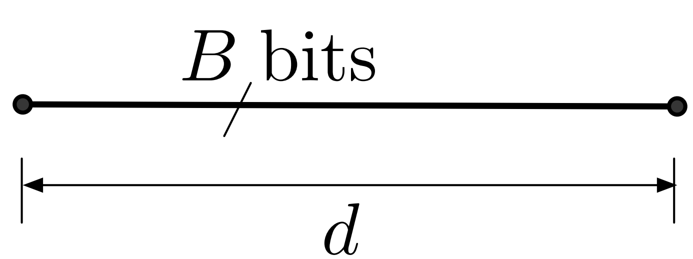


$$p_{ch} = \mathbb{Q}\left(\sqrt{\frac{\eta P_T}{\frac{N_0}{2}}}\right)$$

Circuit energy model: “Information-Friction” [Grover, *IEEE Trans IT 2015*]
[Blake, Ph.D. thesis UToronto, 2017]



$$E_{\text{friction}} = \mu\ w\ d$$



$$E_{\text{info-friction}} = \mu\ B\ d$$

Fundamental limits on total communication energy

Theorem [Grover, IEEE Trans. Info Theory ’15]

$$E_{enc,dec\ per\text{-}bit} \geq \Omega\left(\sqrt{\frac{\log \frac{1}{P_e}}{P_T}}\right)$$

for any code, and any encoding & decoding algorithm implemented in the circuit model

builds on
[El Gamal, Greene, Peng ’84]
[Grover, Woyach, Sahai ’11]
[Grover, Goldsmith, Sahai ’12]
[Grover et al. ’07-15]
[Thompson ’80]

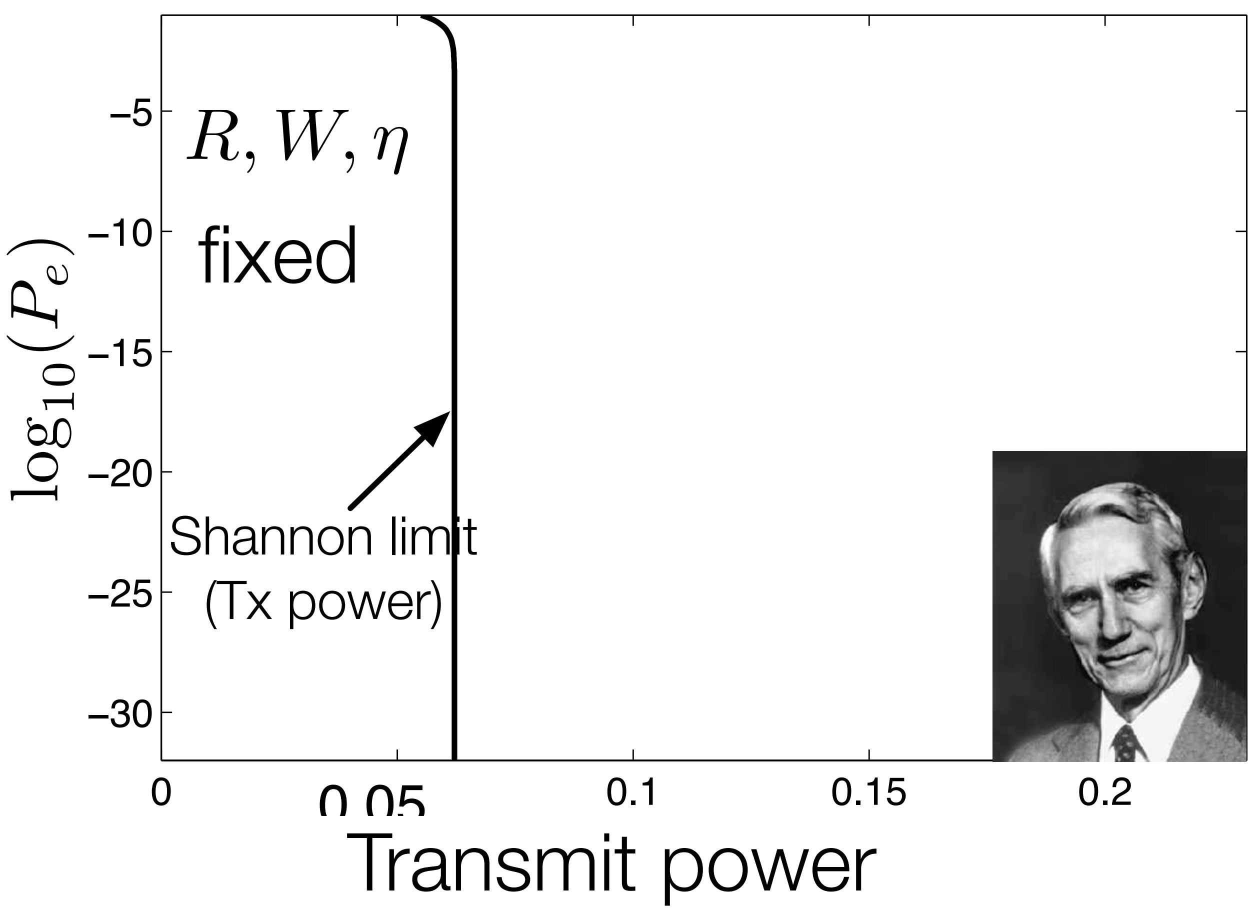
Fundamental limits on total communication energy

Theorem [Grover, IEEE Trans. Info Theory '15]

$$E_{enc,dec\ per\text{-}bit} \geq \Omega\left(\sqrt{\frac{\log \frac{1}{P_e}}{P_T}}\right)$$

for any code, and any encoding & decoding algorithm implemented in the circuit model

builds on
[El Gamal, Greene, Peng '84]
[Grover, Woyach, Sahai '11]
[Grover, Goldsmith, Sahai '12]
[Grover et al. '07-15]
[Thompson '80]

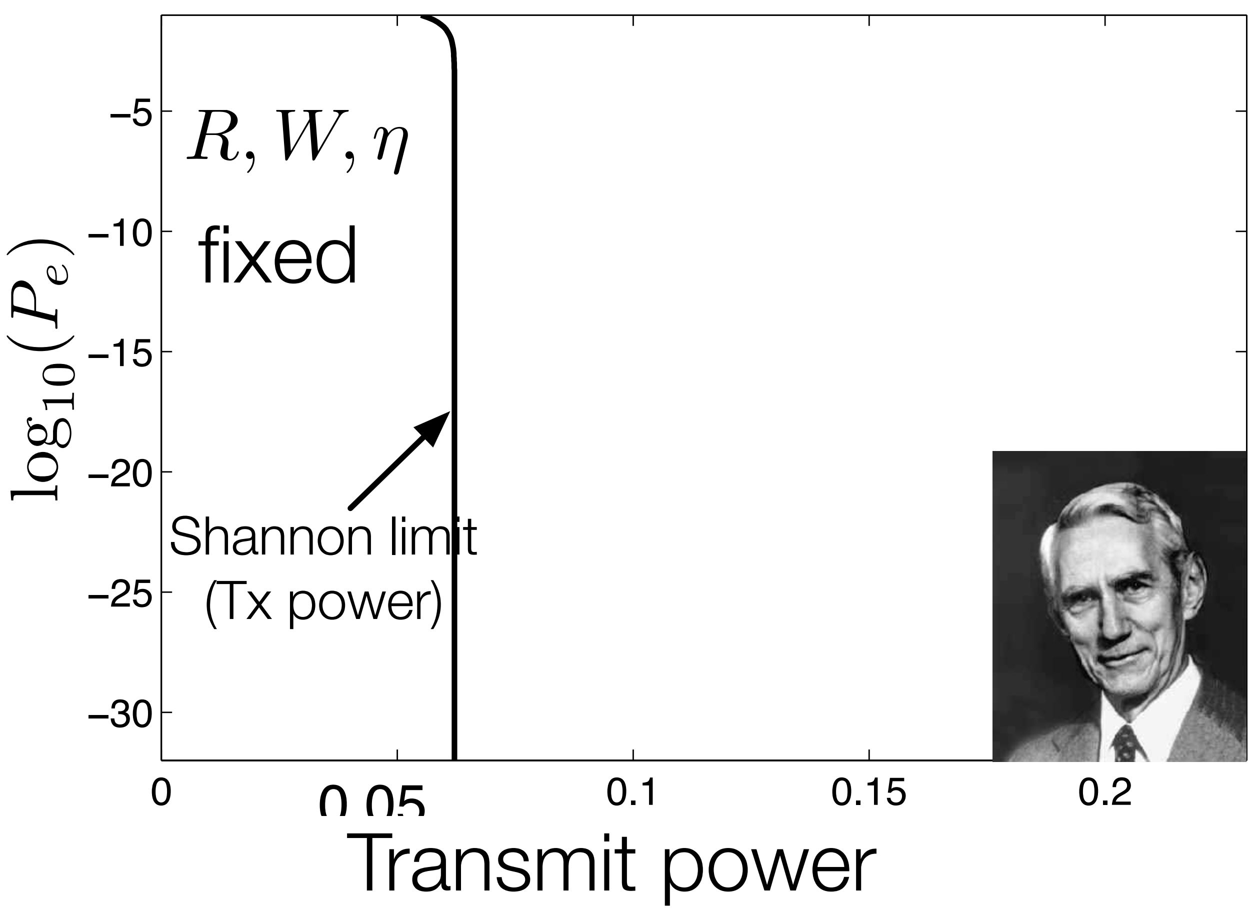


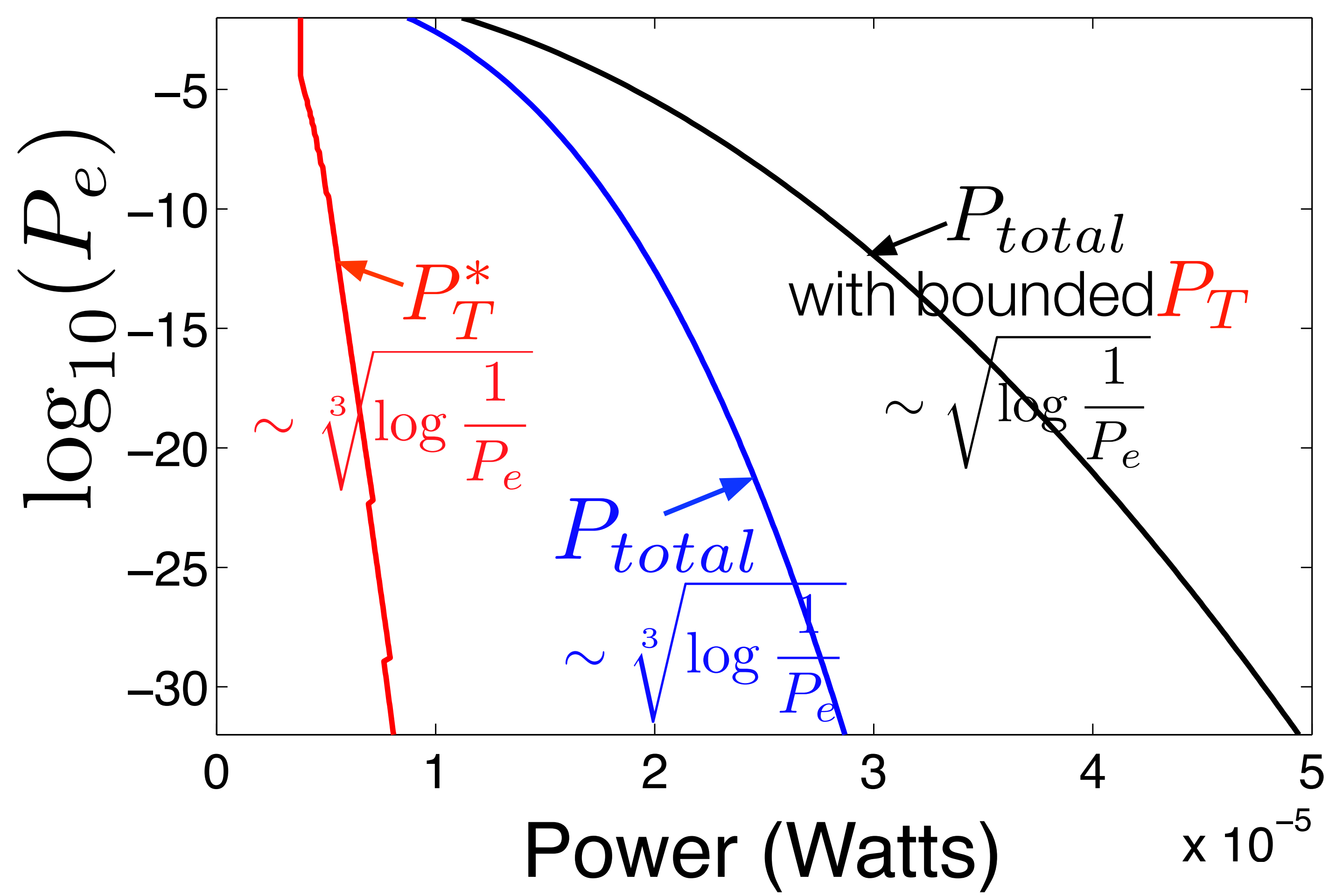
Fundamental limits on total communication energy

Theorem [Grover, IEEE Trans. Info Theory '15]

$$E_{enc,dec\ per\text{-}bit} \geq \Omega\left(\sqrt{\frac{\log\frac{1}{P_e}}{P_T}}\right)$$ for any code, and any encoding & decoding algorithm implemented in the circuit model

builds on
[El Gamal, Greene, Peng '84]
[Grover, Woyach, Sahai '11]
[Grover, Goldsmith, Sahai '12]
[Grover et al. '07-15]
[Thompson '80]





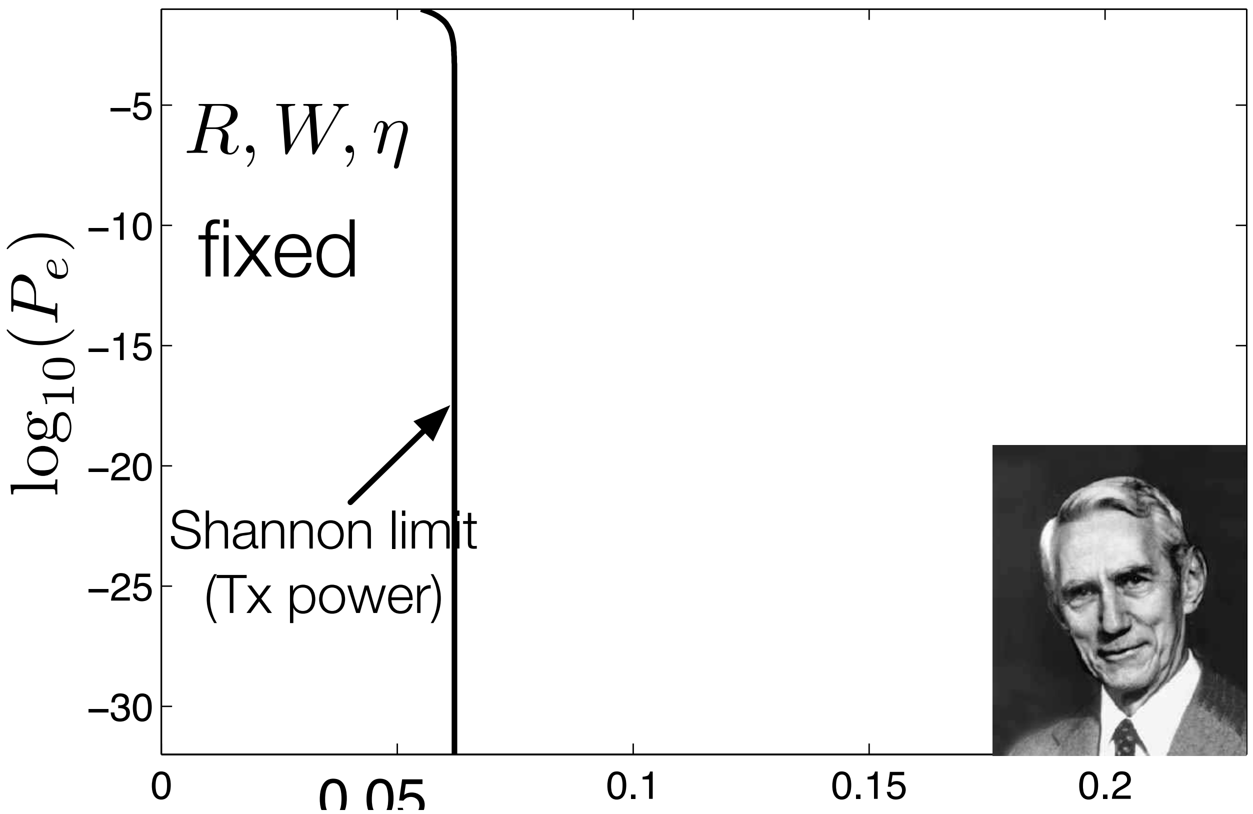
Fundamental limits on total communication energy

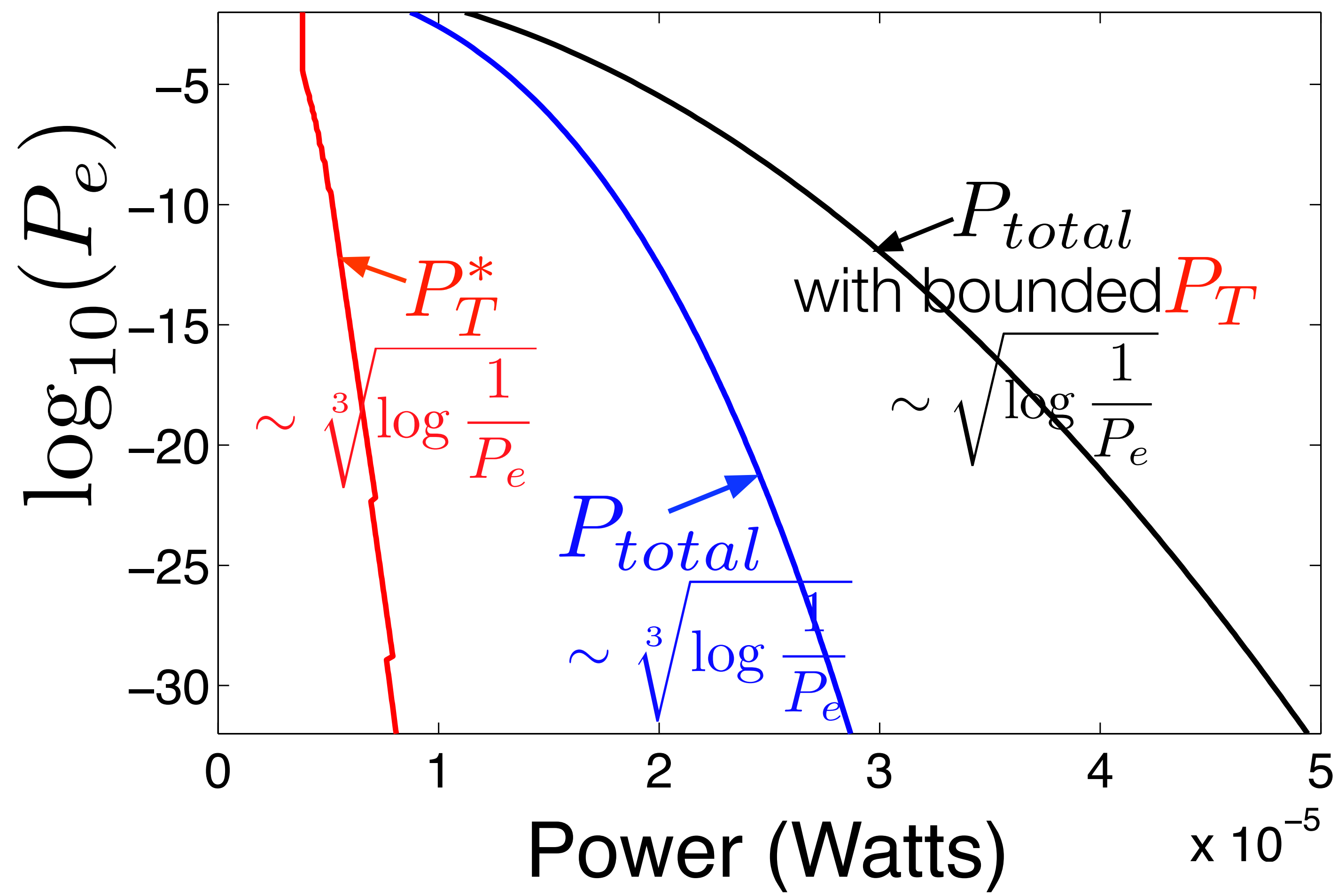
Theorem [Grover, IEEE Trans. Info Theory '15]

$$E_{enc,dec\ per\text{-}bit} \geq \Omega\left(\sqrt{\frac{\log \frac{1}{P_e}}{P_T}}\right)$$

for any code, and any encoding & decoding algorithm implemented in the circuit model

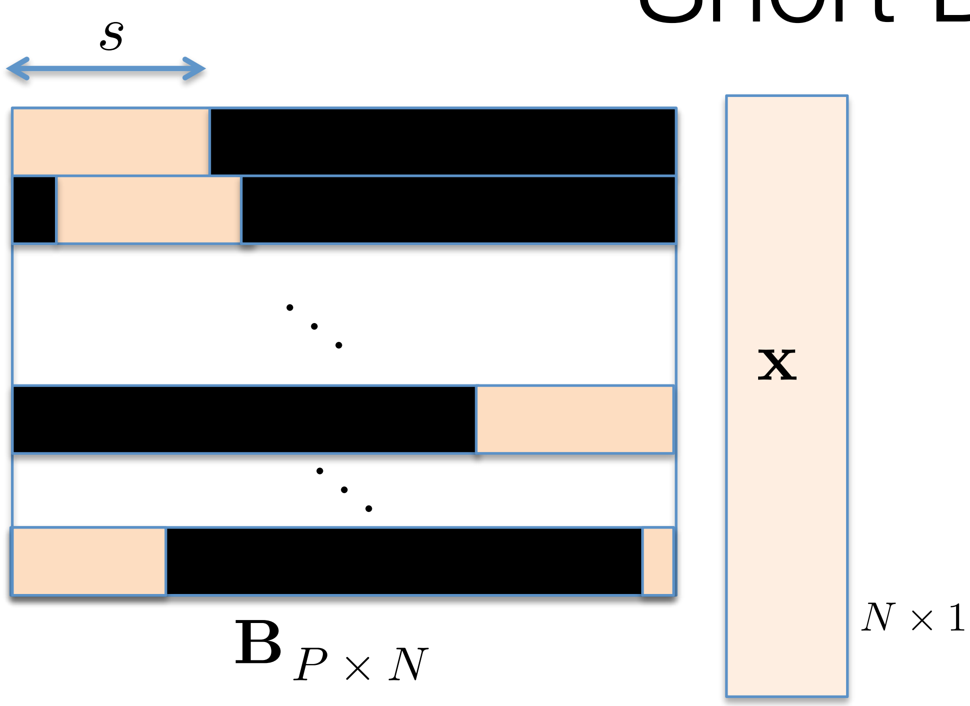
builds on
[El Gamal, Greene, Peng '84]
[Grover, Woyach, Sahai '11]
[Grover, Goldsmith, Sahai '12]
[Grover et al. '07-15]
[Thompson '80]

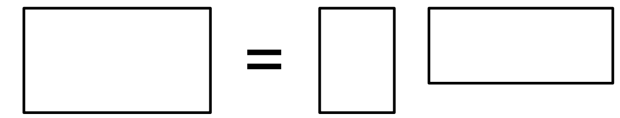




Straightforward extension to noisy computing of invertible linear transforms [Grover, ISIT'14]: don't aim for "Shannon capacity of noisy computing"!

Short Dot Achievability





$$\underset{P \times N}{\mathbf{B}} = \underset{P \times K}{\mathbf{R}} \underset{K \times N}{\begin{bmatrix} \mathbf{A} \\ \mathbf{Z} \end{bmatrix}}$$

$\mathbf{A}_{M \times N}$

$\mathbf{Z}_{(K-M) \times N}$

any square submatrix invertible
(e.g. gen matrix of MDS code; transposed)

$$K = P - r + 1$$

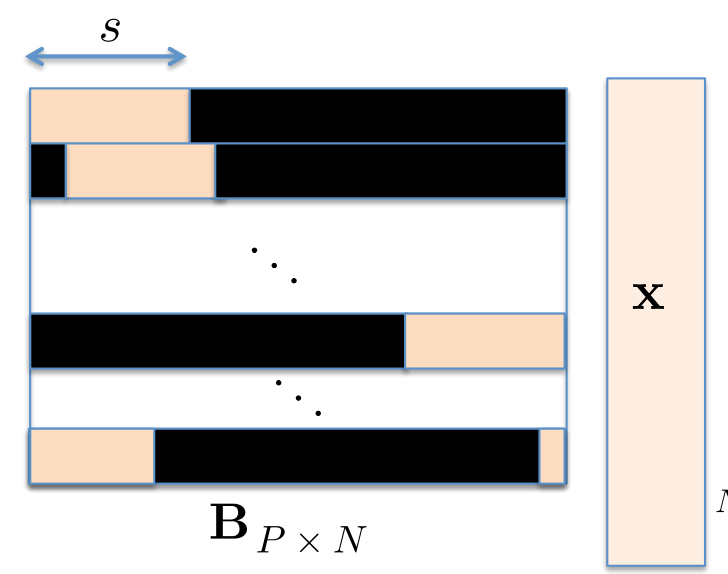
Rows of **A** lie in the span of any K rows of **B**

i-th column of **Z** chosen to set zeroes in the i-th column of **B**

Equation/variable counting gives $s \leq \frac{N}{P}(P - K + M)$

Short Dot outer bound intuition

Intuition: no column can be too sparse:
can't have > K zeros
 - since **A** has to be recoverable from any K rows



This argument yields a looser converse:

Converse: Any Short-Dot code satisfies:

$$s \geq \frac{N}{P}(P - K + 1)$$

Tighten by rank arguments (messy; happy to discuss offline)