3D Coded SUMMA

Communication-Efficient and Robust Parallel Matrix Multiplication

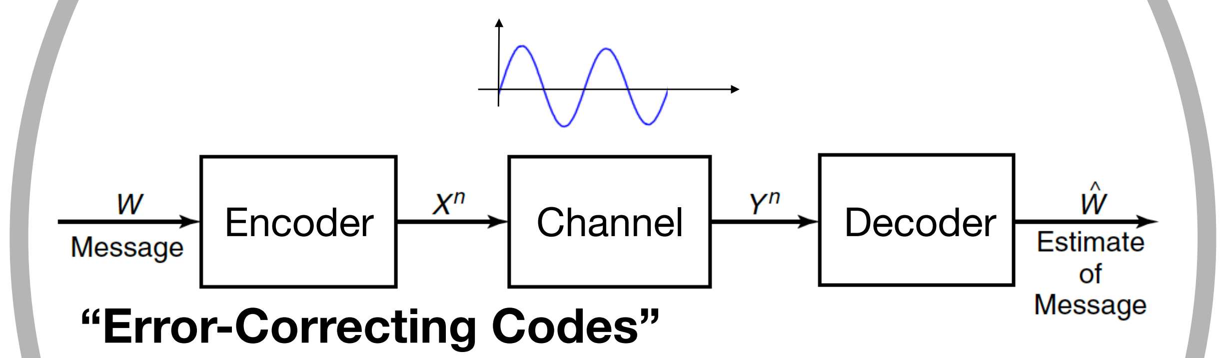
Haewon Jeong[1], Yaoqing Yang[2], Christian Engelmann[3], Vipul Gupta[2], Tze Meng Low[4], Pulkit Grover[4], Viveck Cadambe[5] and Kannan Ramchandran[2]

[1] Harvard University, [2] UC Berkeley, [3] Oak Ridge National Lab, [4] CMU, [5] PennState

Coded Computing

**Large-Scale
Computing Algorithms**

**Information Theory
Coding Theory**



“Error-Correcting Codes”

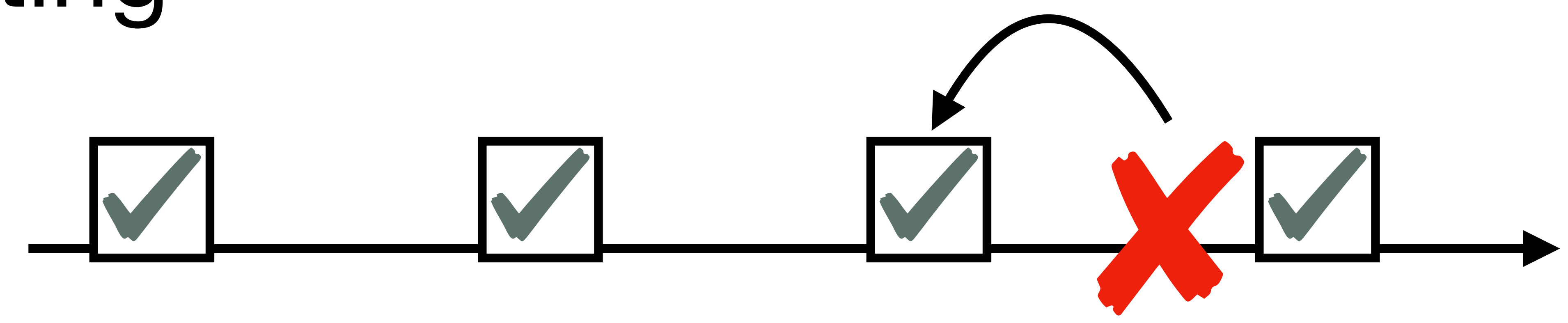
How do we optimize redundancy for desired reliability?

***Reliable Large-Scale
Computing Algorithms***

- Algorithm-based Fault-Tolerance (ABFT) [Chen et al.’05, ’06, ’08, ’11, Bosilca et al. ’09]

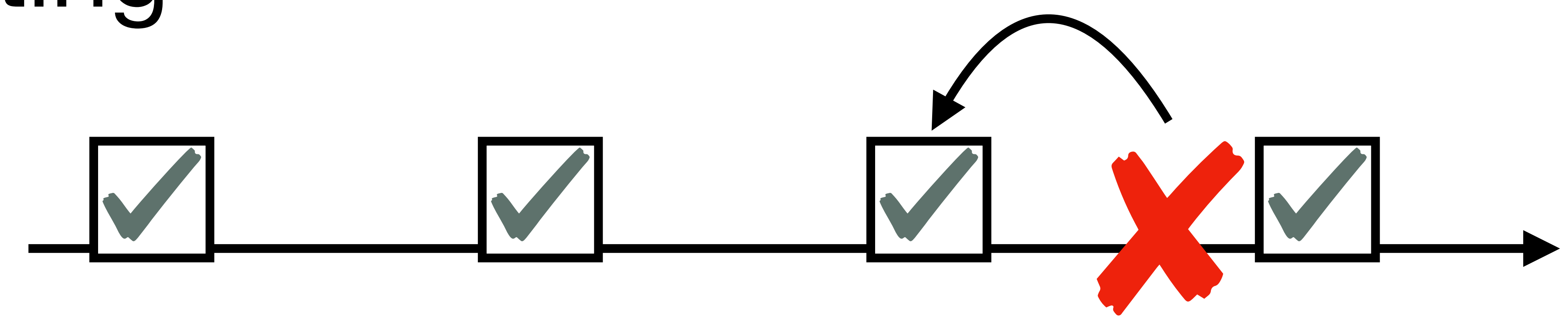
Traditional Reliability Techniques

- Checkpointing

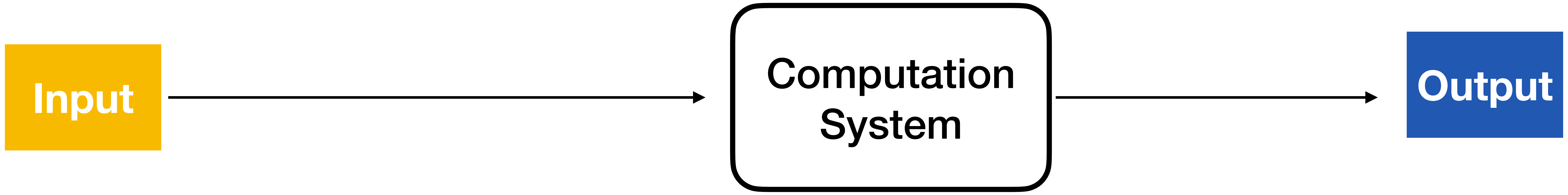


Traditional Reliability Techniques

- Checkpointing

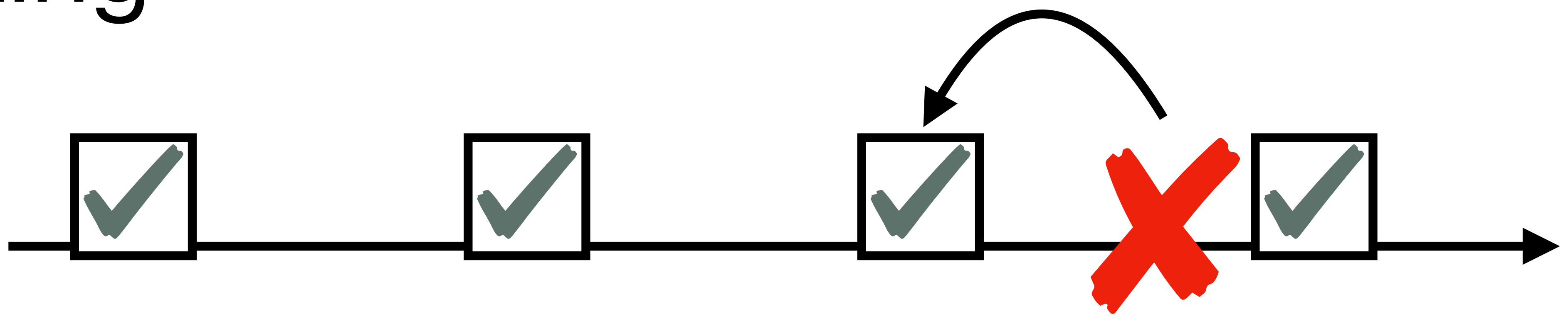


- Replication

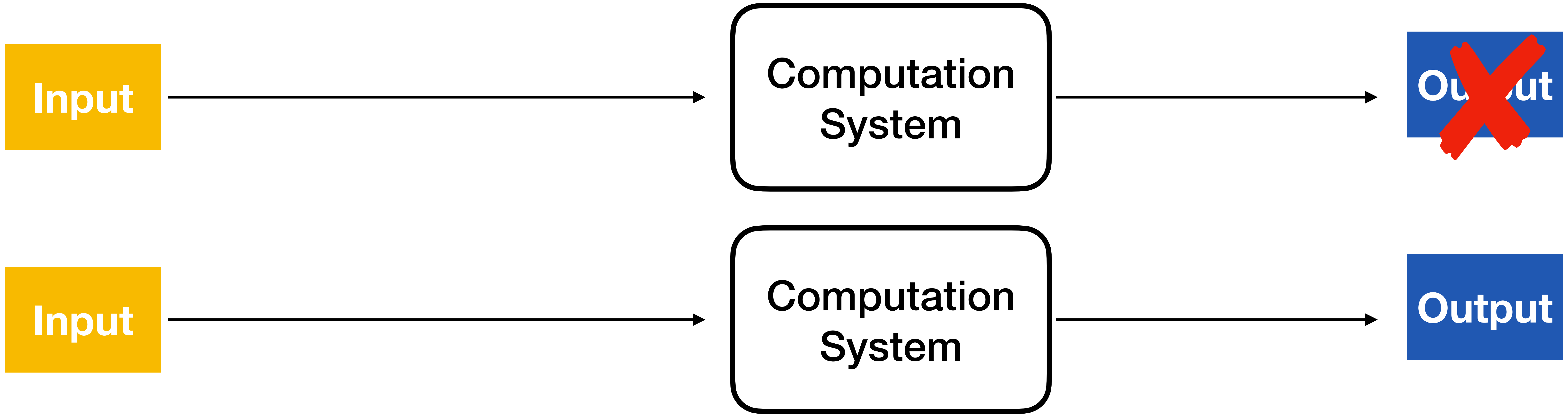


Traditional Reliability Techniques

- Checkpointing



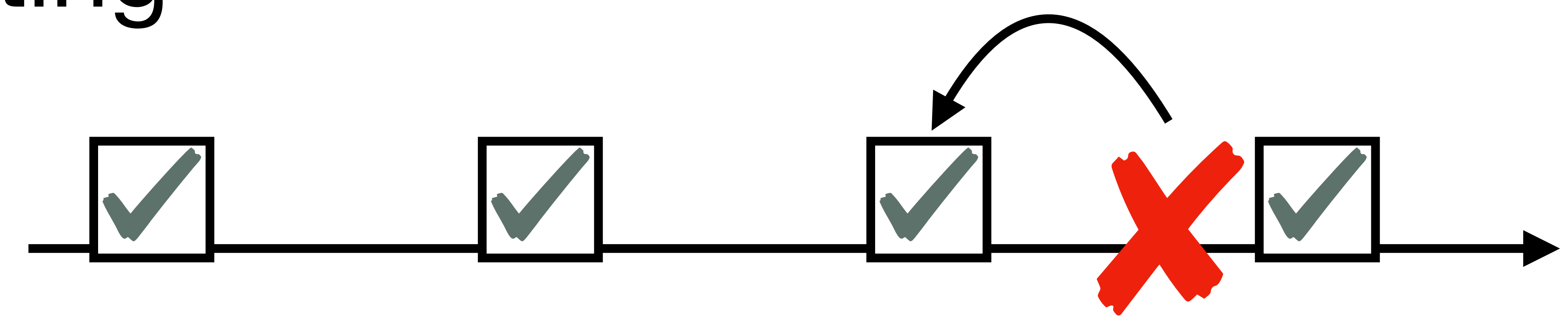
- Replication



["Replication is more efficient than you think" Benoit et al. SC '19]

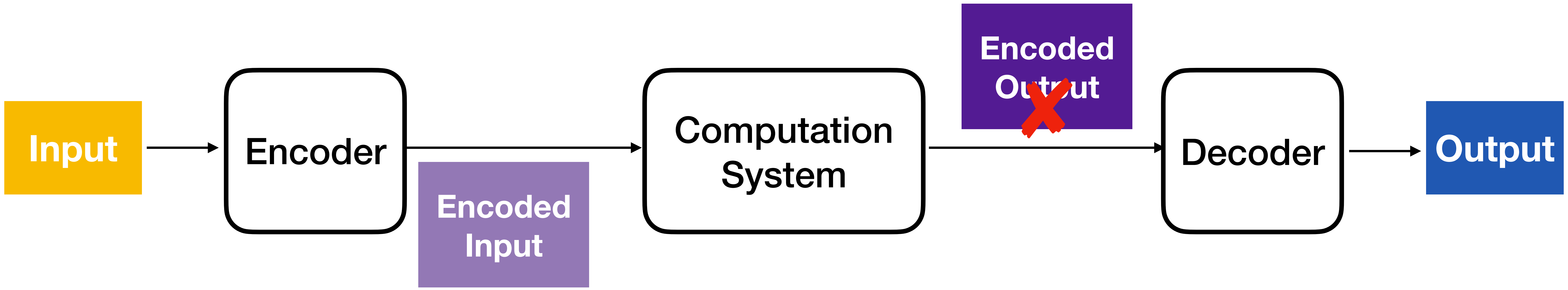
Traditional Reliability Techniques

- Checkpointing



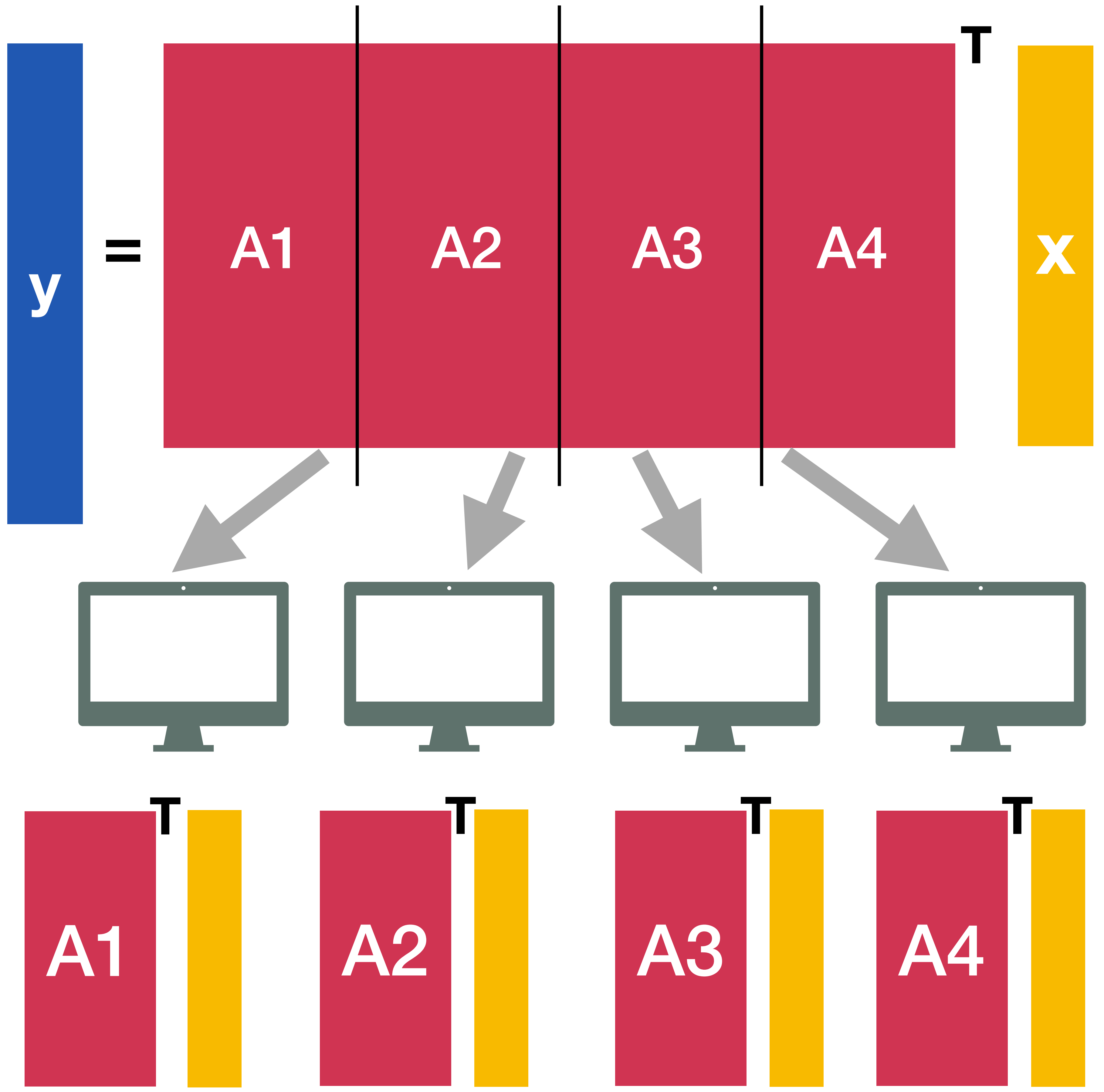
- Replication

- Coded Computing



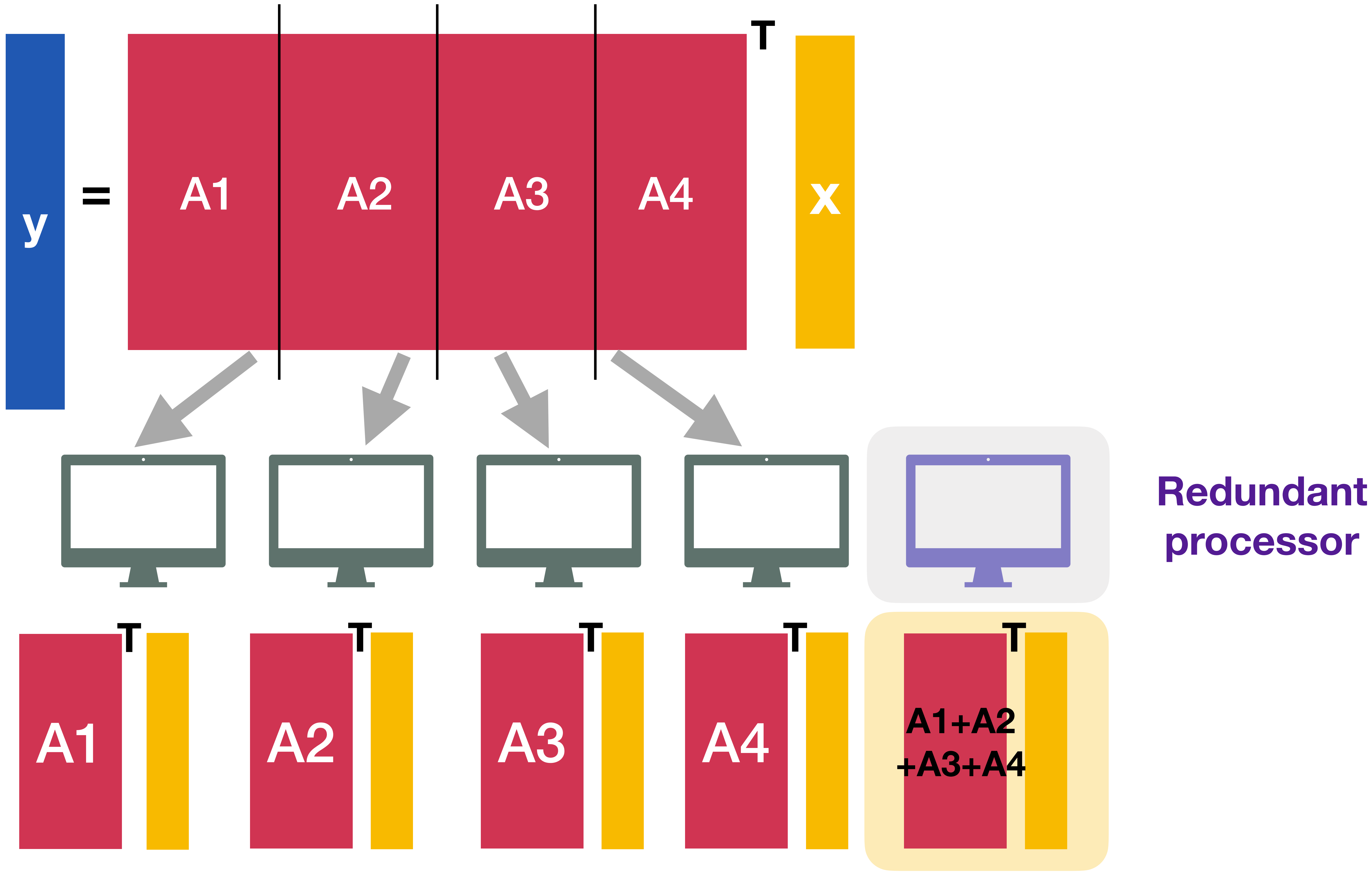
Basic Idea of Coded Computing

Through Matrix-Vector Multiplication Example



Basic Idea of Coded Computing

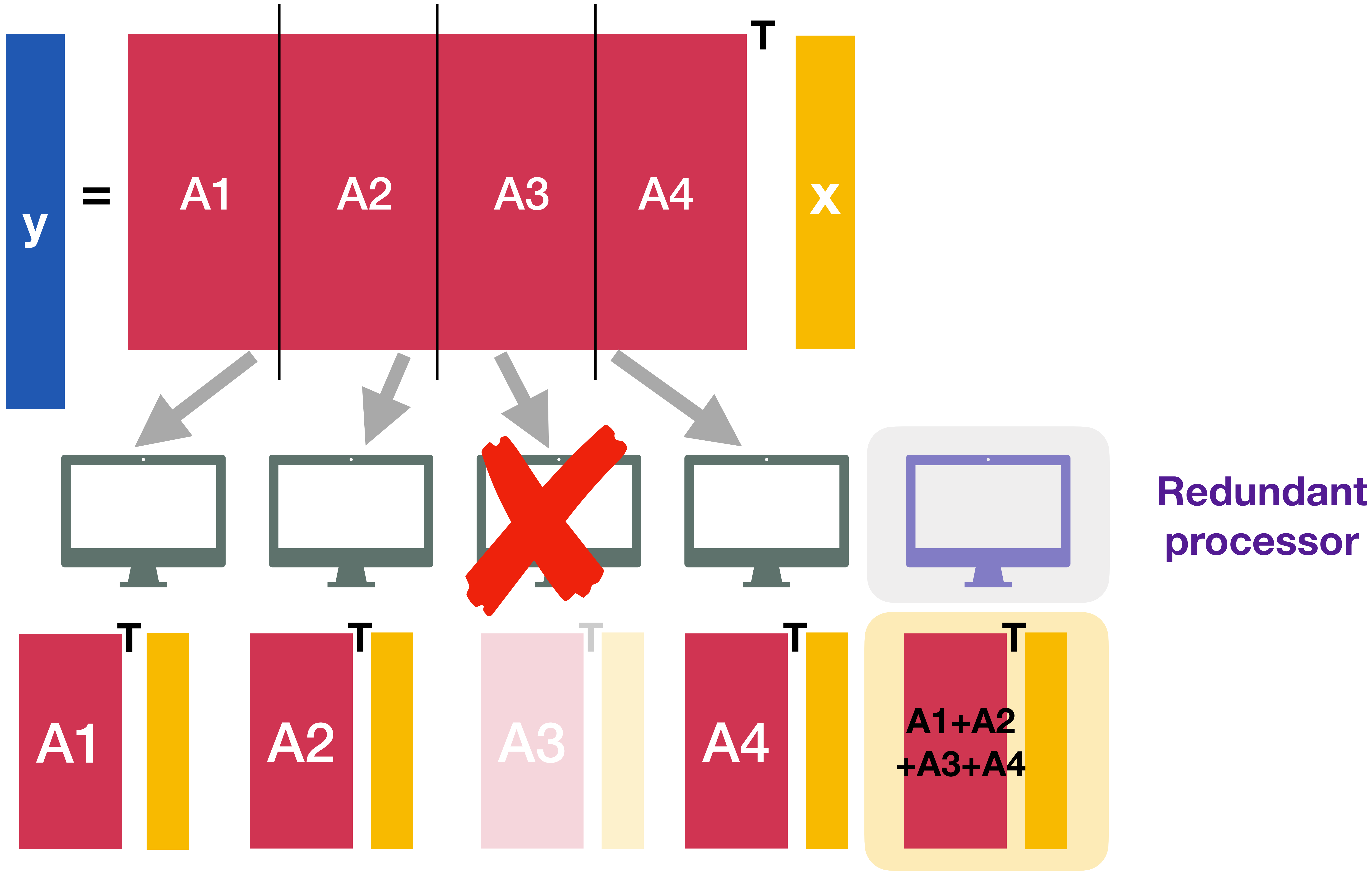
Through Matrix-Vector Multiplication Example



Matrix is encoded using ***error-correcting codes***

Basic Idea of Coded Computing

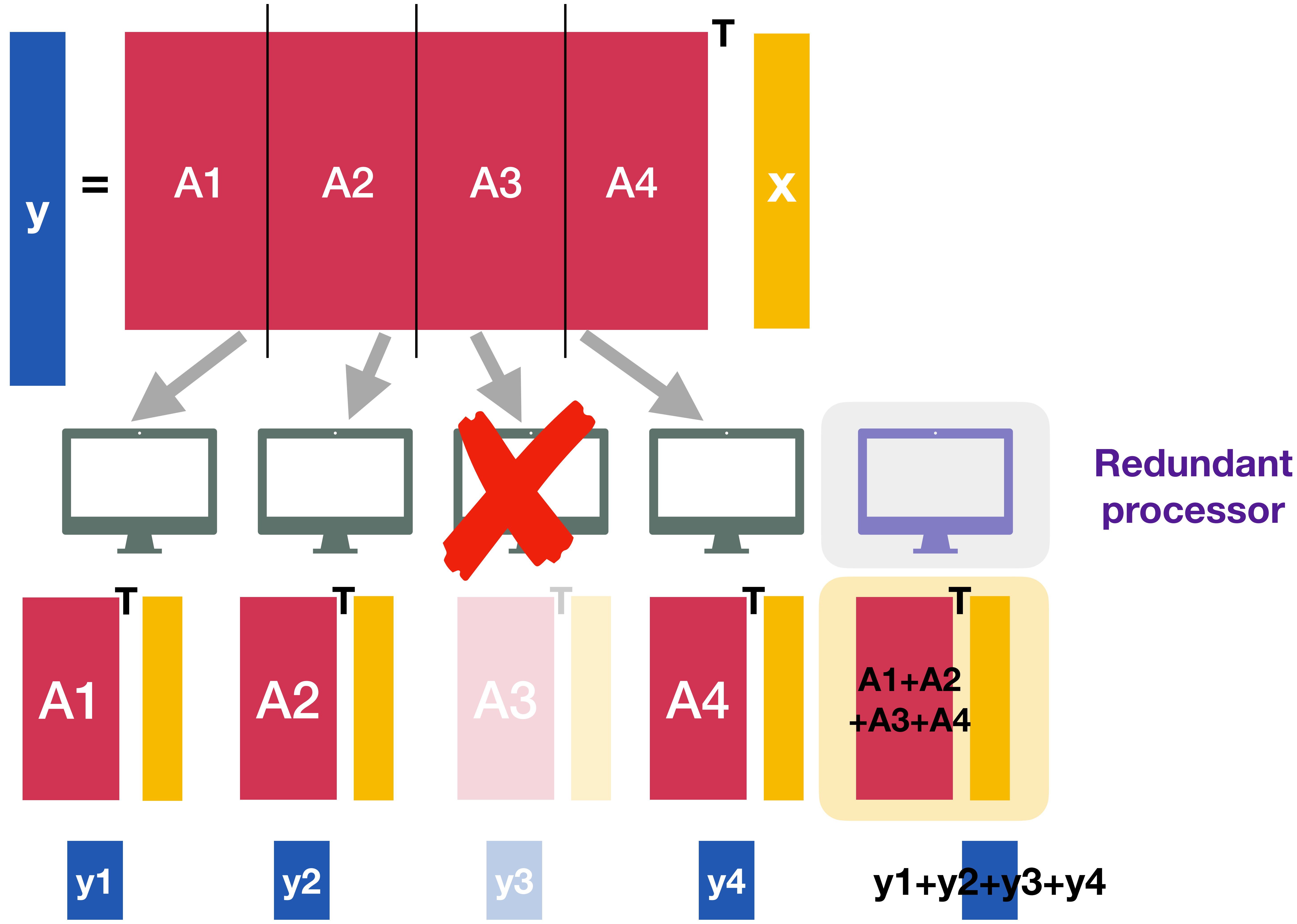
Through Matrix-Vector Multiplication Example



Matrix is encoded using ***error-correcting codes***

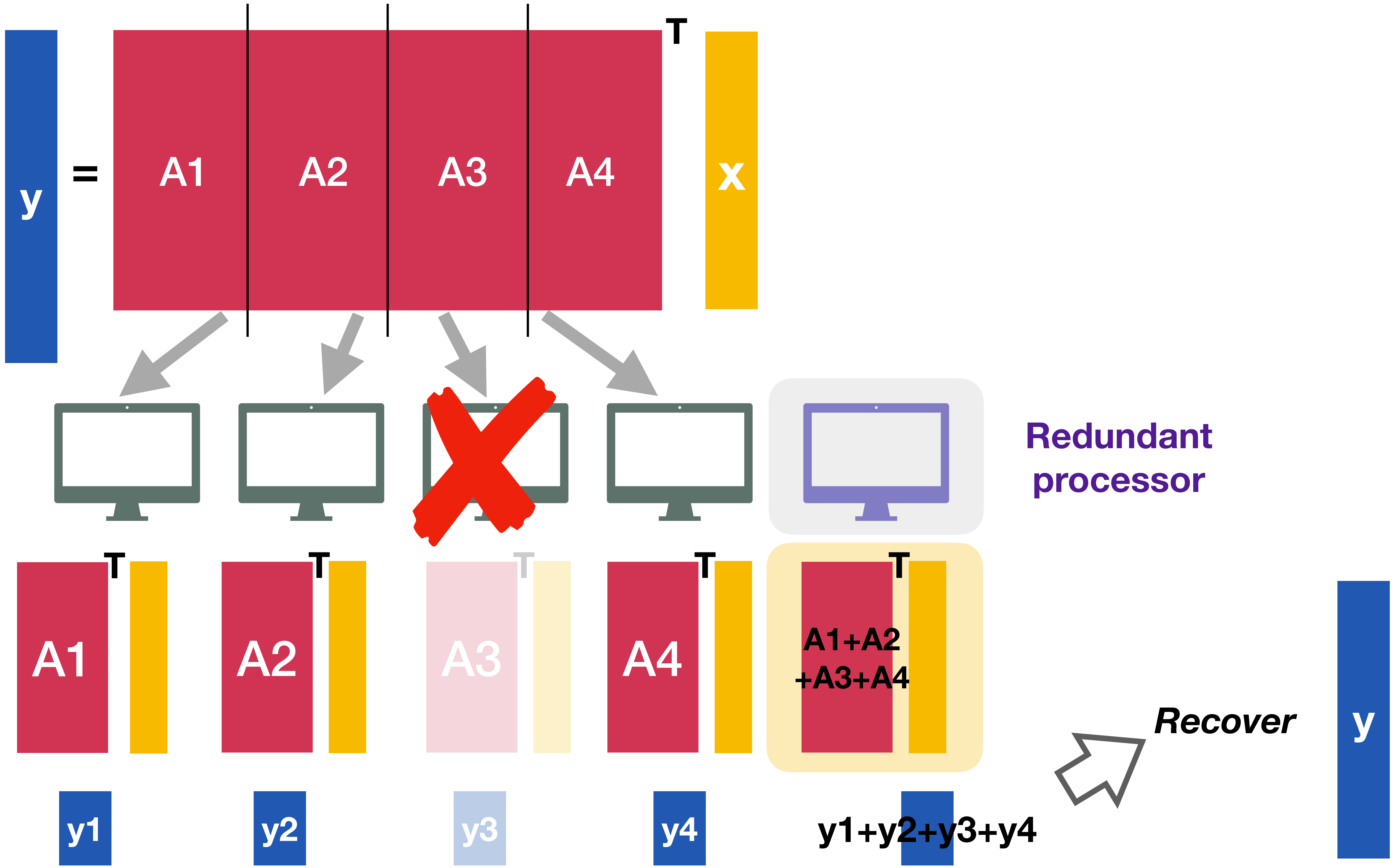
Basic Idea of Coded Computing

Through Matrix-Vector Multiplication Example



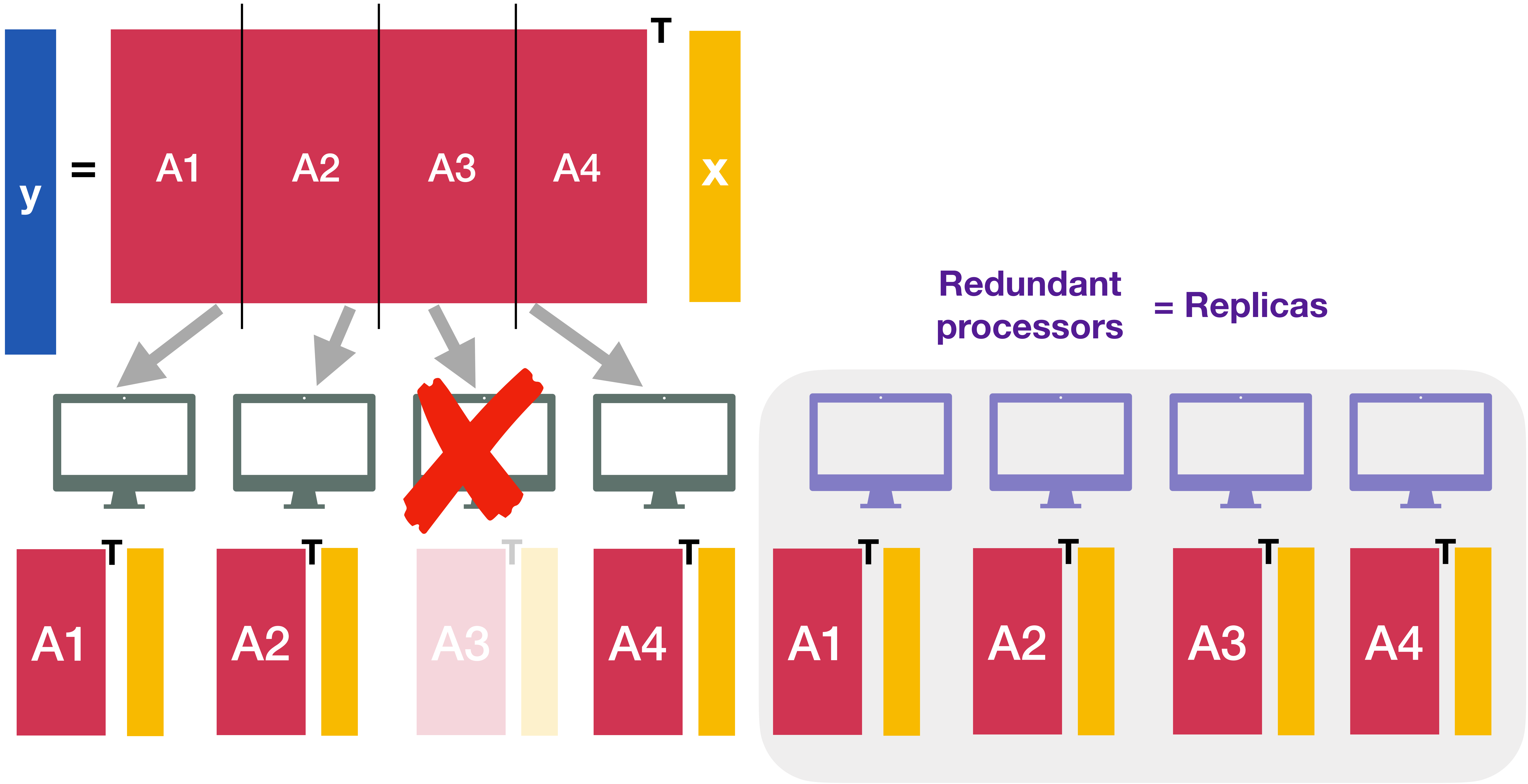
Basic Idea of Coded Computing

Through Matrix-Vector Multiplication Example



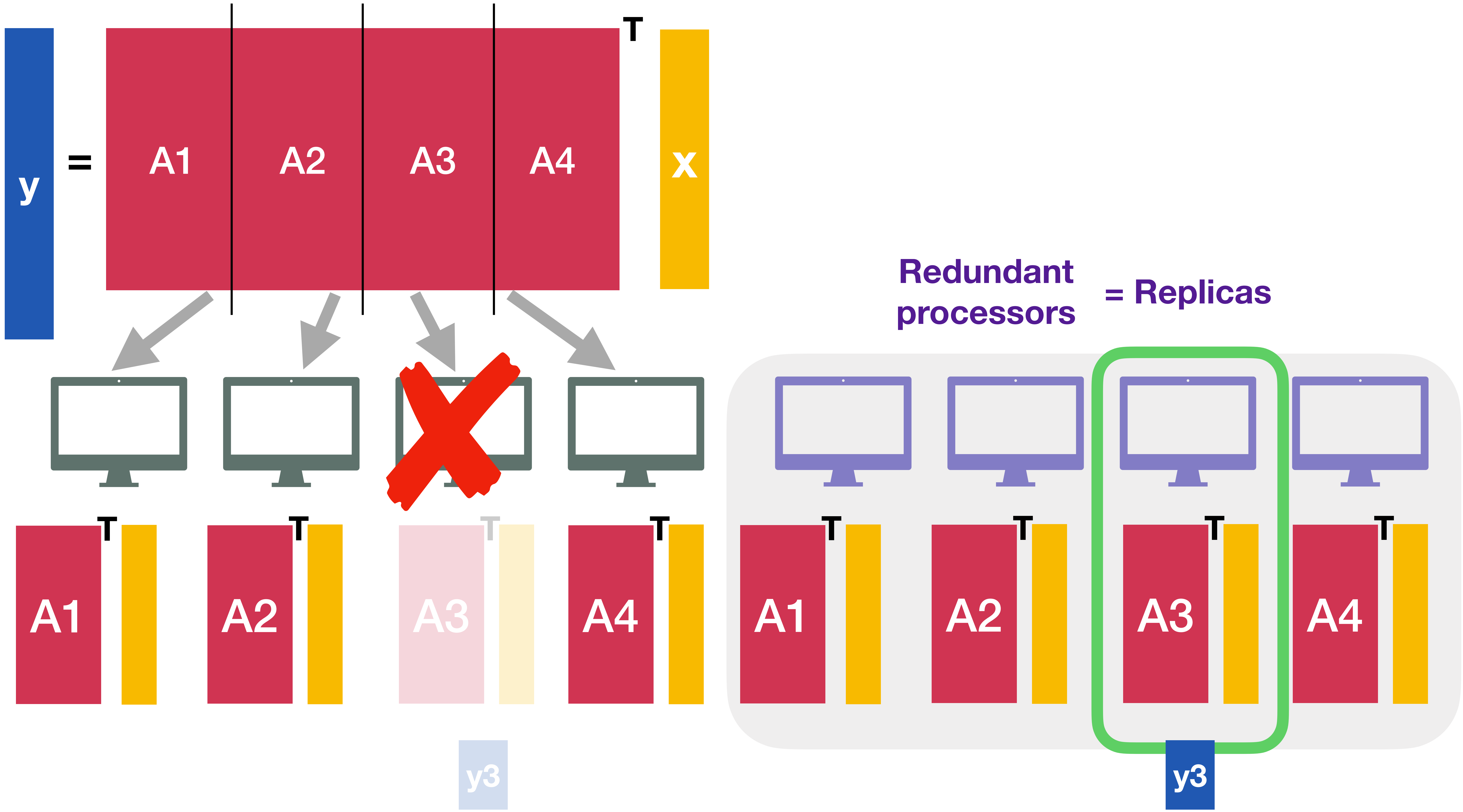
Basic Idea of Coded Computing

Comparison with simple replication



Basic Idea of Coded Computing

Comparison with simple replication



Recent Advances in Coded Computing

- **Coded computing for matrix multiplication** [Lee et al. ’17, Yu et al. ’17, Fahim et al. ’17, Wang et al. ’18, Baharav et al. ’18, Gupta et al. ’18, Jeong et al. ’18, Reisizadeh et al. ’19, Mallick et al. ’19, Aliasgari et al. ’19, Jeong et al. ’19, Yu et al. ’20]
- **Coded computing for distributed optimization** [Tandon et al. ’17, Raviv et al. ’18, Ye et al. ’18, Data et al. ’18, Lit et al. ’18, Karakus et al. ’19, Maity et al. ’19, Ozfatura et al. ’19, Amiri et al. ’20]
- **Coded computing for iterative algorithms** [Haddapour et al. ’18, Yang et al. ’18, Prakash et al. ’20]
- **Coded computing for blockchains** [Yu et al. ’19, Li et al. ’20]
- **Coded MapReduce** [Li et al. ’15, ’17, Ramkumar ’19]
- **Coded computing for Elastic/Serverless Computing** [Yang et al. ’19, Woolsey et al. ’20, Gupta et al. ’20]
- **Coded computing for federated learning** [Dhakal et al. ’19, Zhao ’19, Kim et al. ’20, Prakash et al. ’20]

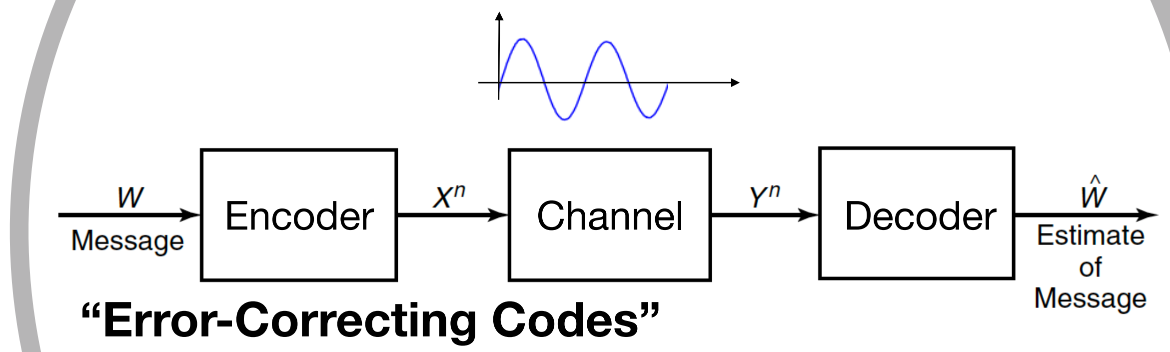
Why Coded Computing for HPC?

- Small resource overhead
  - Much smaller overhead than universal methods (e.g., checkpointing, replication)
  - No disk access
- Scalable: (Overhead of Coding)/(Total Execution Time) = $o(1)$ as $P \to \infty$
- Fault-tolerance at the application level
  - Reduces HW design burden
  - Can be built into libraries: `matmul( … , fault_tolerance =1)`
  - System-agnostic & flexible

Coded Computing vs ABFT

Large-Scale Computing Algorithms

Information Theory
Coding Theory



"Error-Correcting Codes"

How do we optimize redundancy for desired reliability?

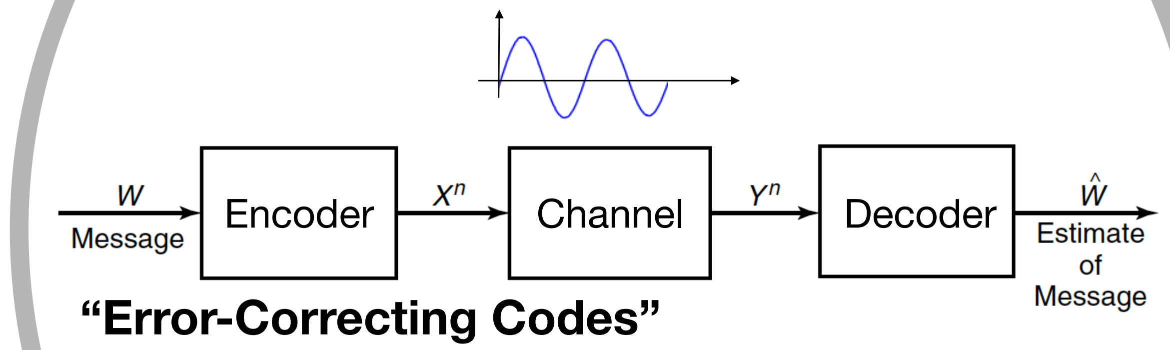
Coded Computing vs ABFT

Apply tools from Coding Theory
to practical HPC applications

ABFT [Huang et al. ’84, Jou ’86, Tao et al. ’93, Wang et al. ’94
Chen et al.’05, ’06, ’08, ’11, Bosilca et al. ’09]

Large-Scale
Computing Algorithms

Information Theory
Coding Theory



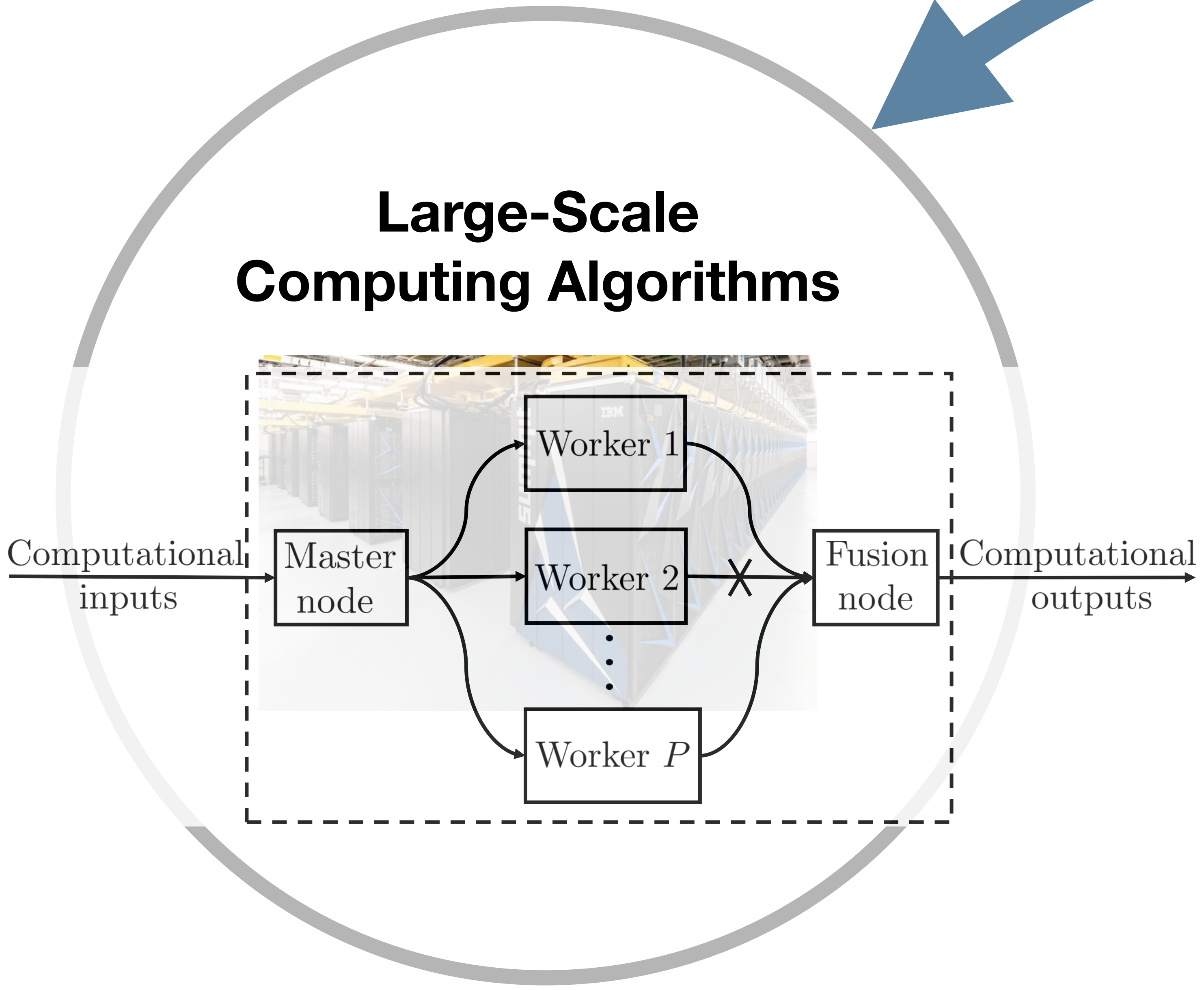
“Error-Correcting Codes”

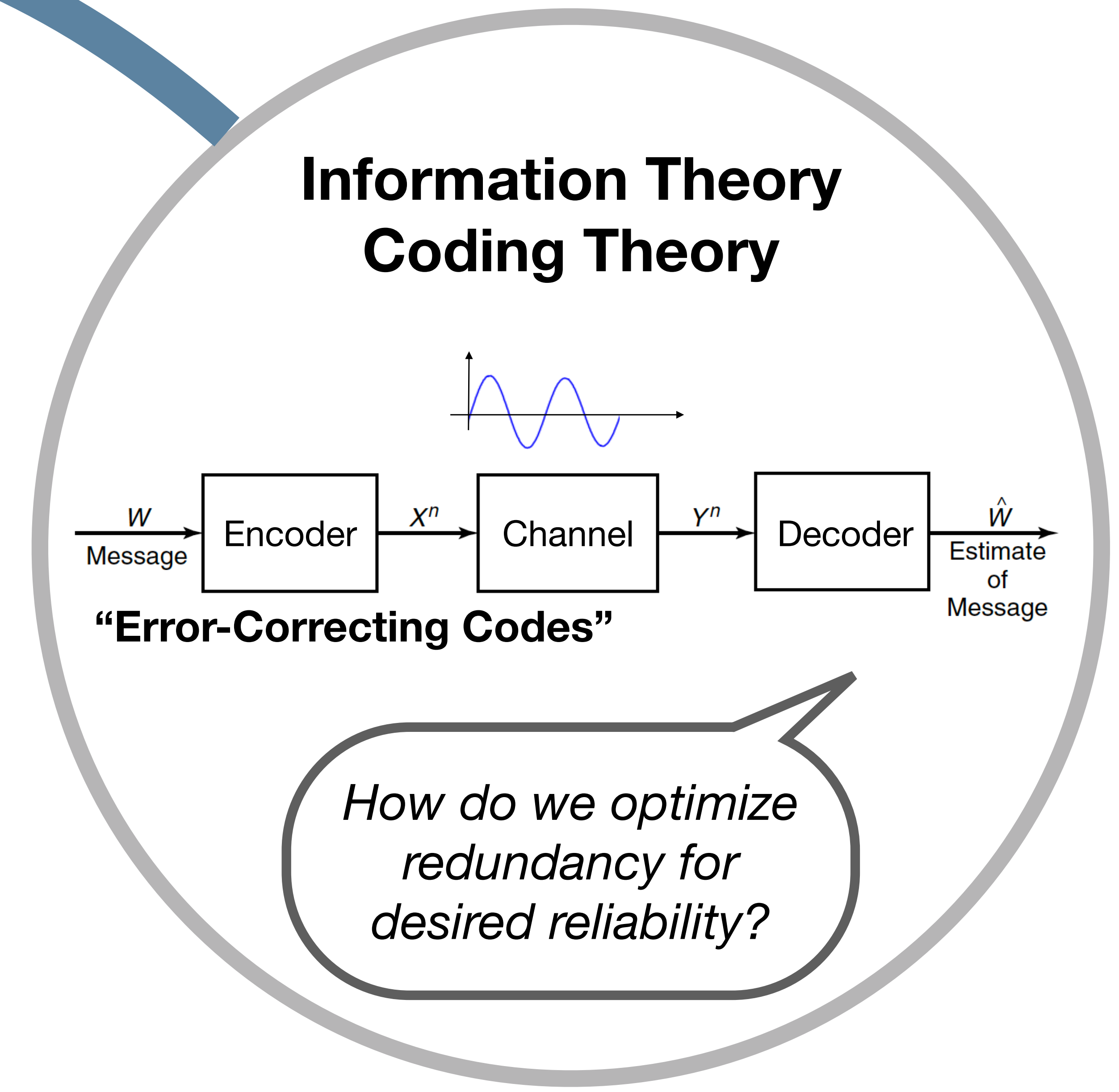
How do we optimize redundancy for desired reliability?

Coded Computing vs ABFT

Apply tools from Coding Theory
to practical HPC applications

ABFT [Huang et al. ’84, Jou ’86, Tao et al. ’93, Wang et al. ’94
Chen et al.’05, ’06, ’08, ’11, Bosilca et al. ’09]



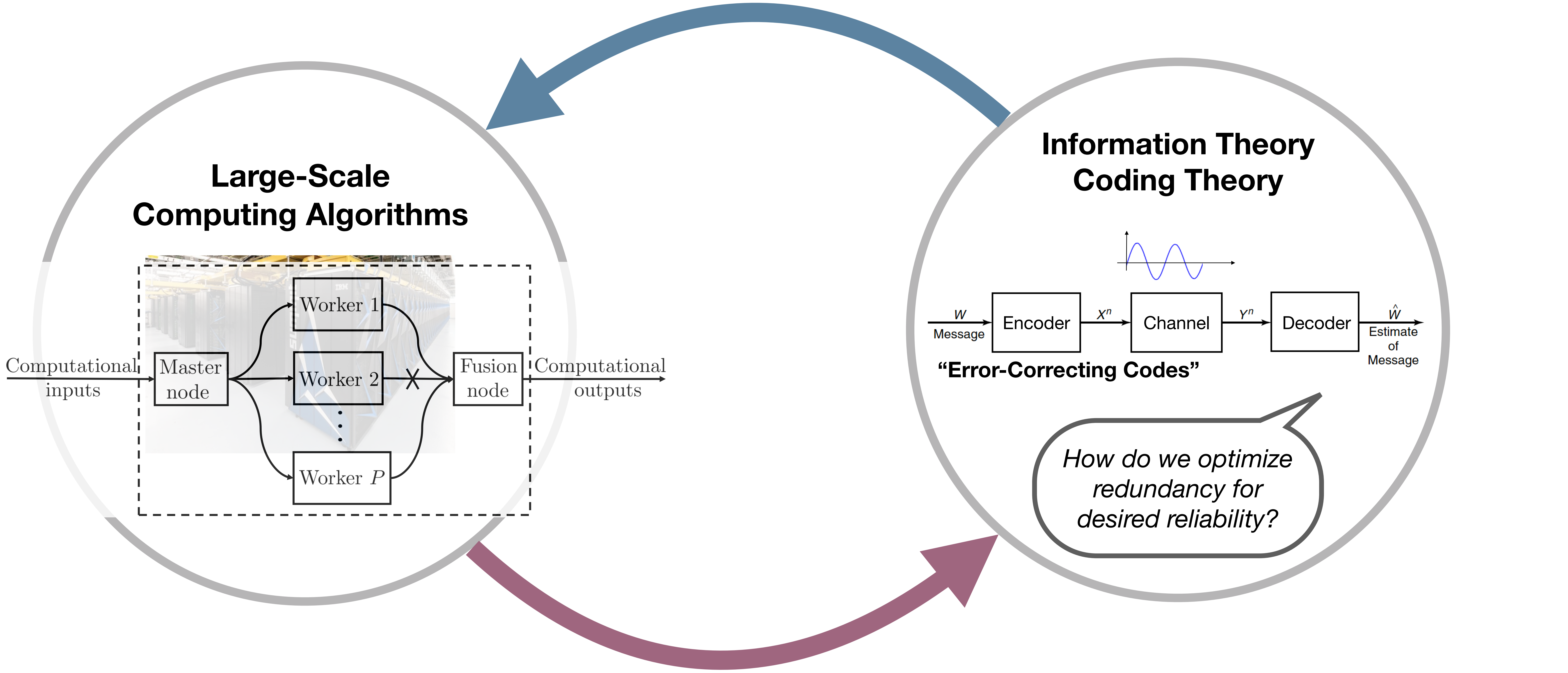


Coded Computing

Coded Computing vs ABFT

Apply tools from Coding Theory
to practical HPC applications

ABFT [Huang et al. ’84, Jou ’86, Tao et al. ’93, Wang et al. ’94
Chen et al.’05, ’06, ’08, ’11, Bosilca et al. ’09]



Coded Computing

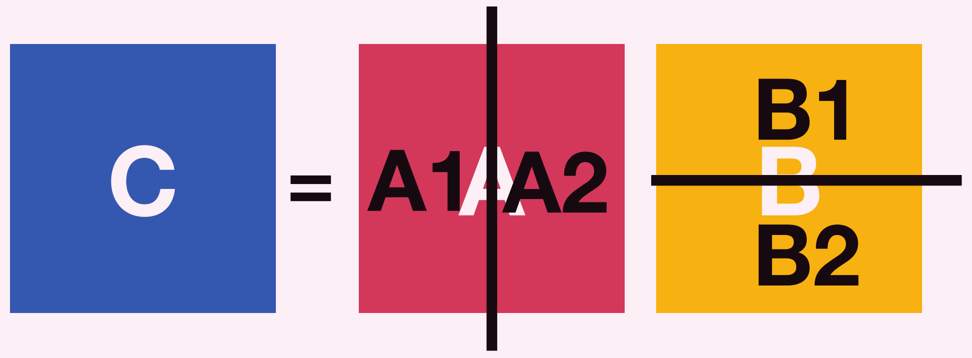
Develop new coding tools on an
abstract computing model

This Work : 3D Coded SUMMA

Fault-Tolerant Distributed Matrix Multiplication

MatDot Codes

[Fahim et al. ’17]

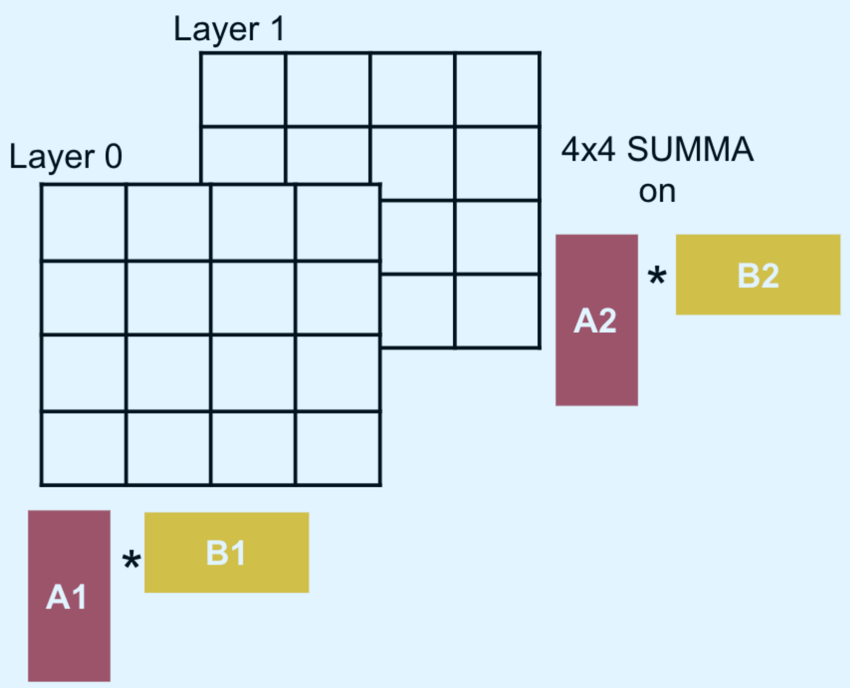


State-of-the-art coded computing for matrix multiplication

+

Communication-Avoiding Integration

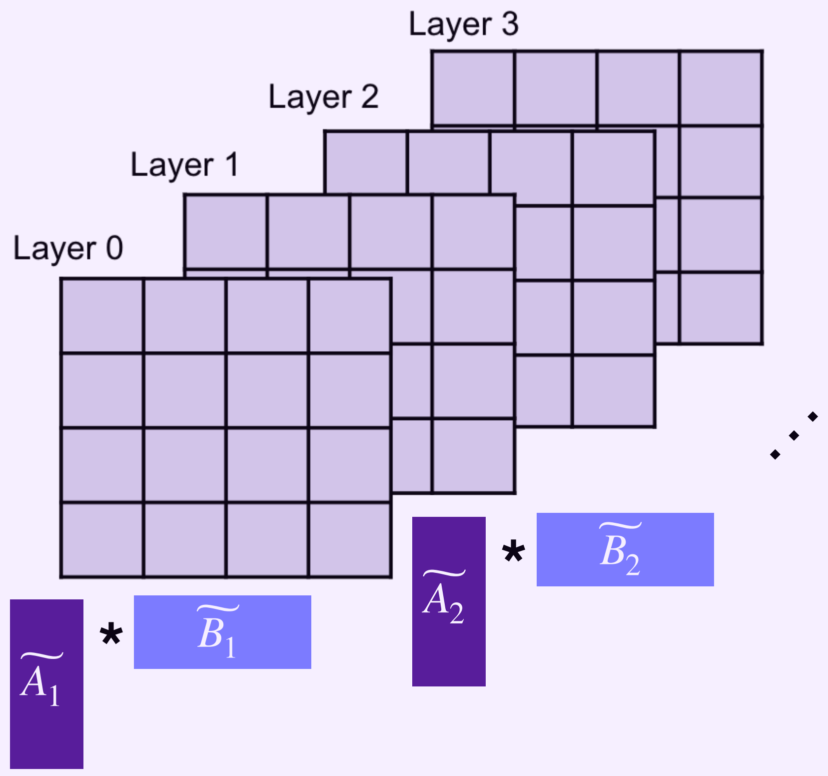
3D SUMMA



Communication-Efficient Matrix Multiplication Algorithm

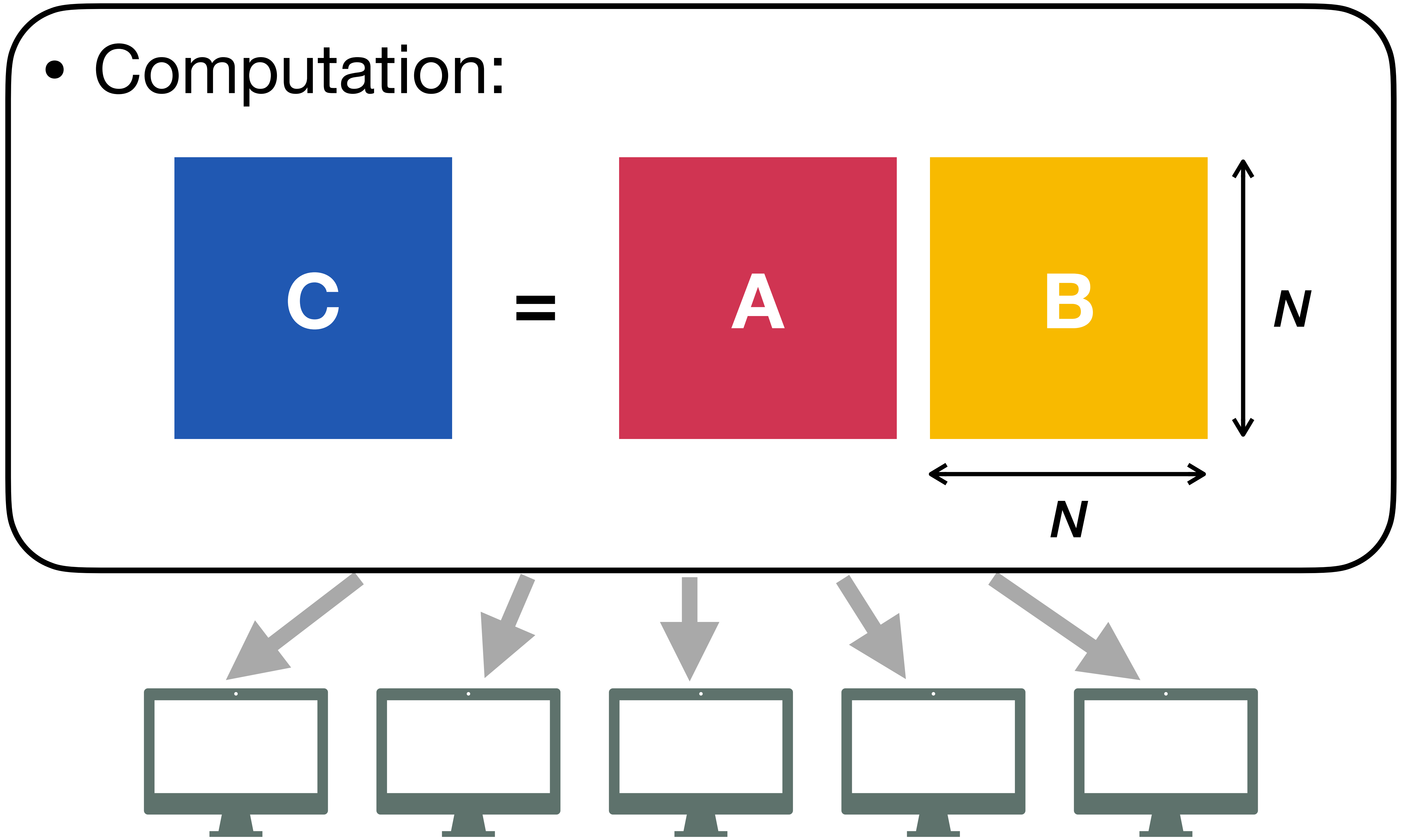
=

3D Coded SUMMA



MatDot Codes [Fahim et al. ’17, ’19]

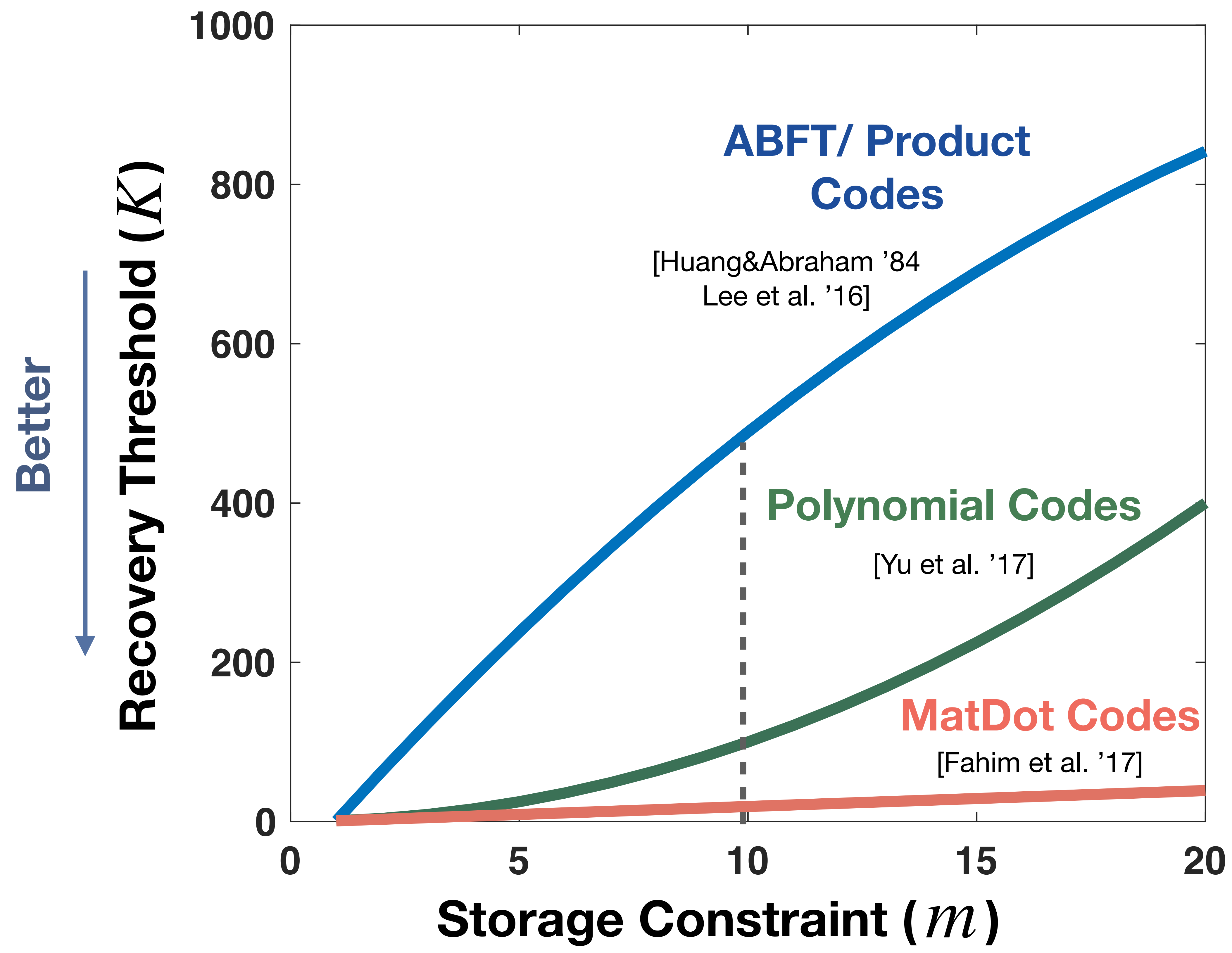
Coded Computing for Matrix Multiplication — Problem Setup



m: Storage Constraint
Each worker node can store only $1/m$ *of* ***A*** and ***B***

K : Recovery Threshold
Minimum number of successful workers to recover ***C***

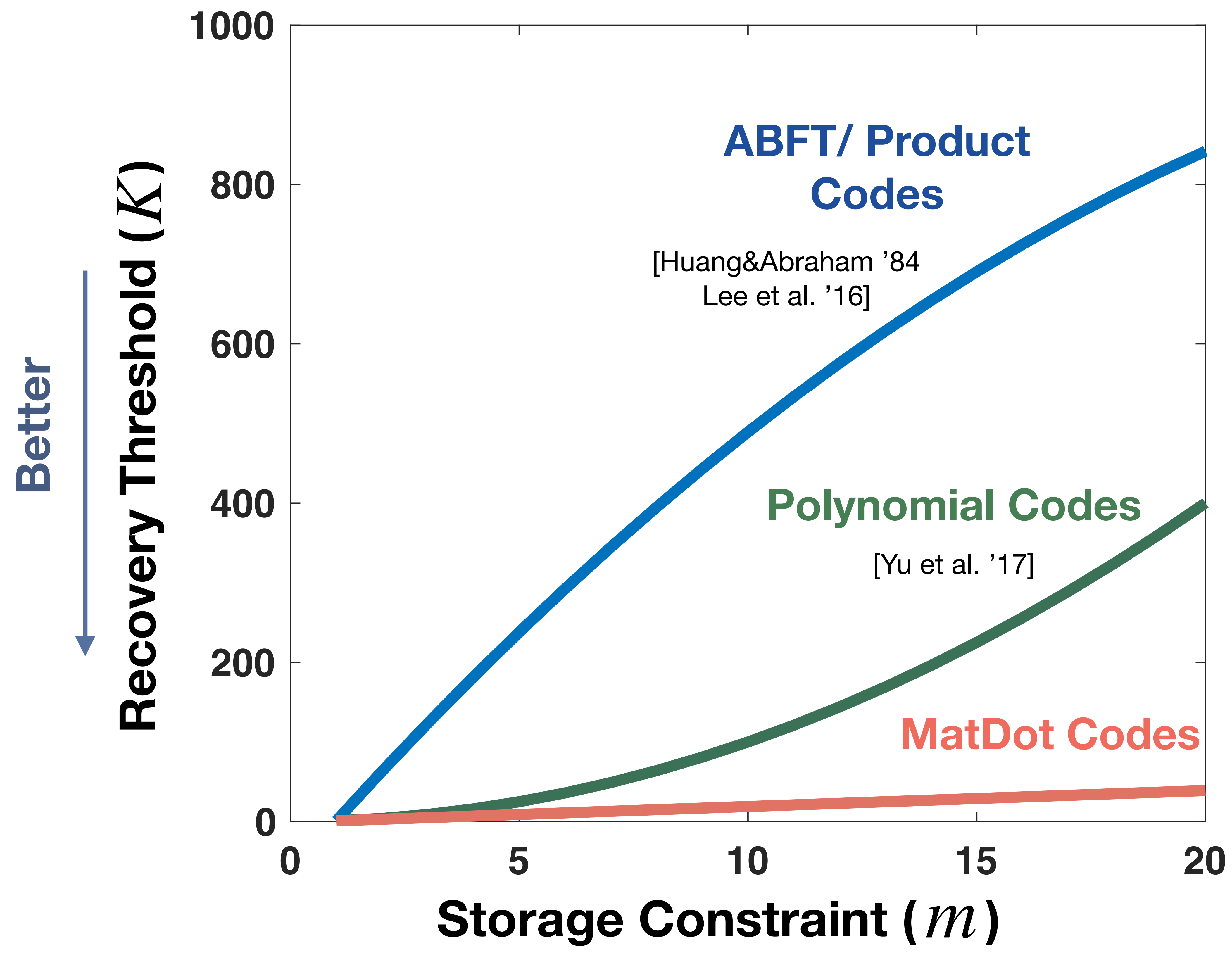
MatDot Codes Are Recovery Threshold Optimal



m: Storage Constraint
*Each worker node can store only $1/m$ of **A** and **B***

K : Recovery Threshold
*Minimum number of successful workers to recover **C***

MatDot Codes Are Recovery Threshold Optimal



$$K = m^2$$

⇕

$$K = 2m - 1$$

Provably Optimal !
[Yu et al. '18]

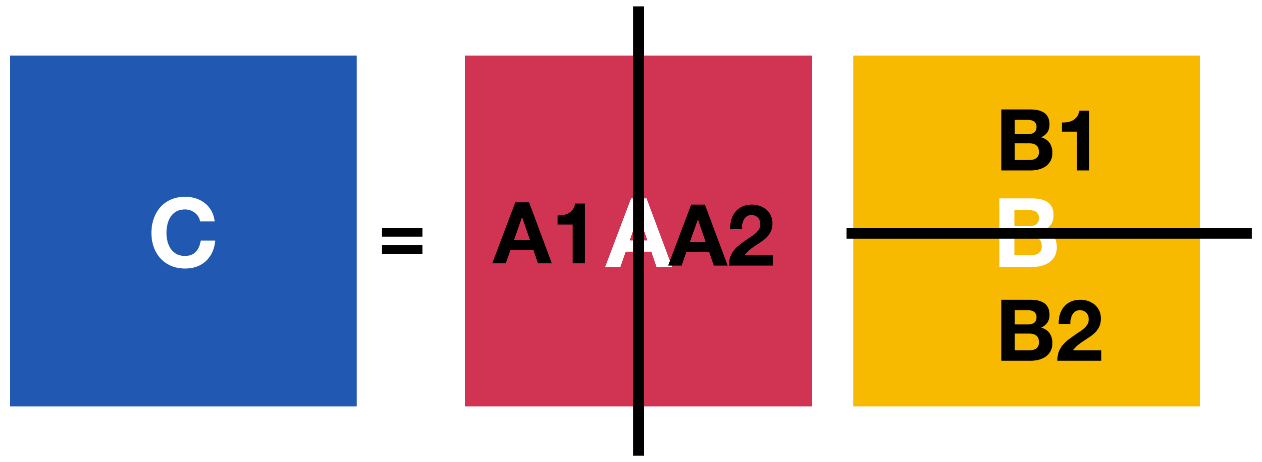
m: Storage Constraint
*Each worker node can store only 1/m of **A** and **B***

K : Recovery Threshold
*Minimum number of successful workers to recover **C***

Core Ideas of MatDot Codes

[Fahim et al. ’17, ’19]

- Split matrix multiplication into outer products:



= A1*B1 + A2*B2

- Adapt Reed-Solomon codes for this setting:

  - *Most well-known codes for storage*
  - *Construction based on polynomials*

$$p_{\mathbf{A}}(x) = \mathbf{A}_1 + \mathbf{A}_2 x$$

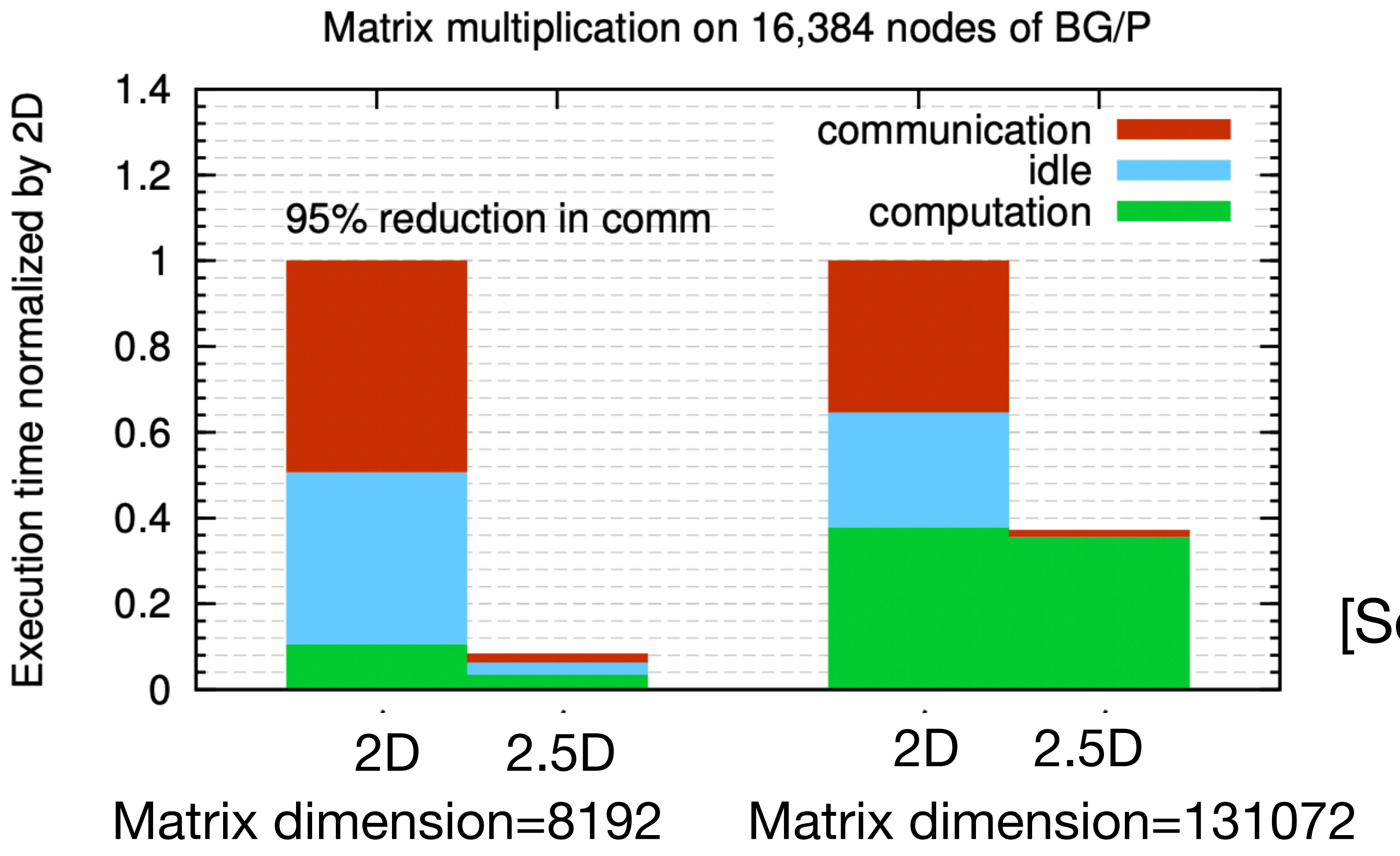
$$p_{\mathbf{B}}(x) = \mathbf{B}_2 + \mathbf{B}_1 x$$

$$p_{\mathbf{C}}(x) = p_{\mathbf{A}}(x) p_{\mathbf{B}}(x)$$

$$= \mathbf{A}_1\mathbf{B}_2 + (\mathbf{A}_1\mathbf{B}_1 + \mathbf{A}_2\mathbf{B}_2)x + \mathbf{A}_2\mathbf{B}_1 x^2$$

3D SUMMA : Communication-Efficient Parallel Multiplication

- More communication-efficient variant of Scalable Universal Matrix Multiplication Algorithm (SUMMA) [Schatz et al. ’16, van de Geijin ’97]
- Nodes are placed on a 3D grid (M x M x m, $m \leq M$). Perform SUMMA on each layer.
- Also known as 2.5D SUMMA [Solomonik&Demmel ’11]



(M=32, m=16)

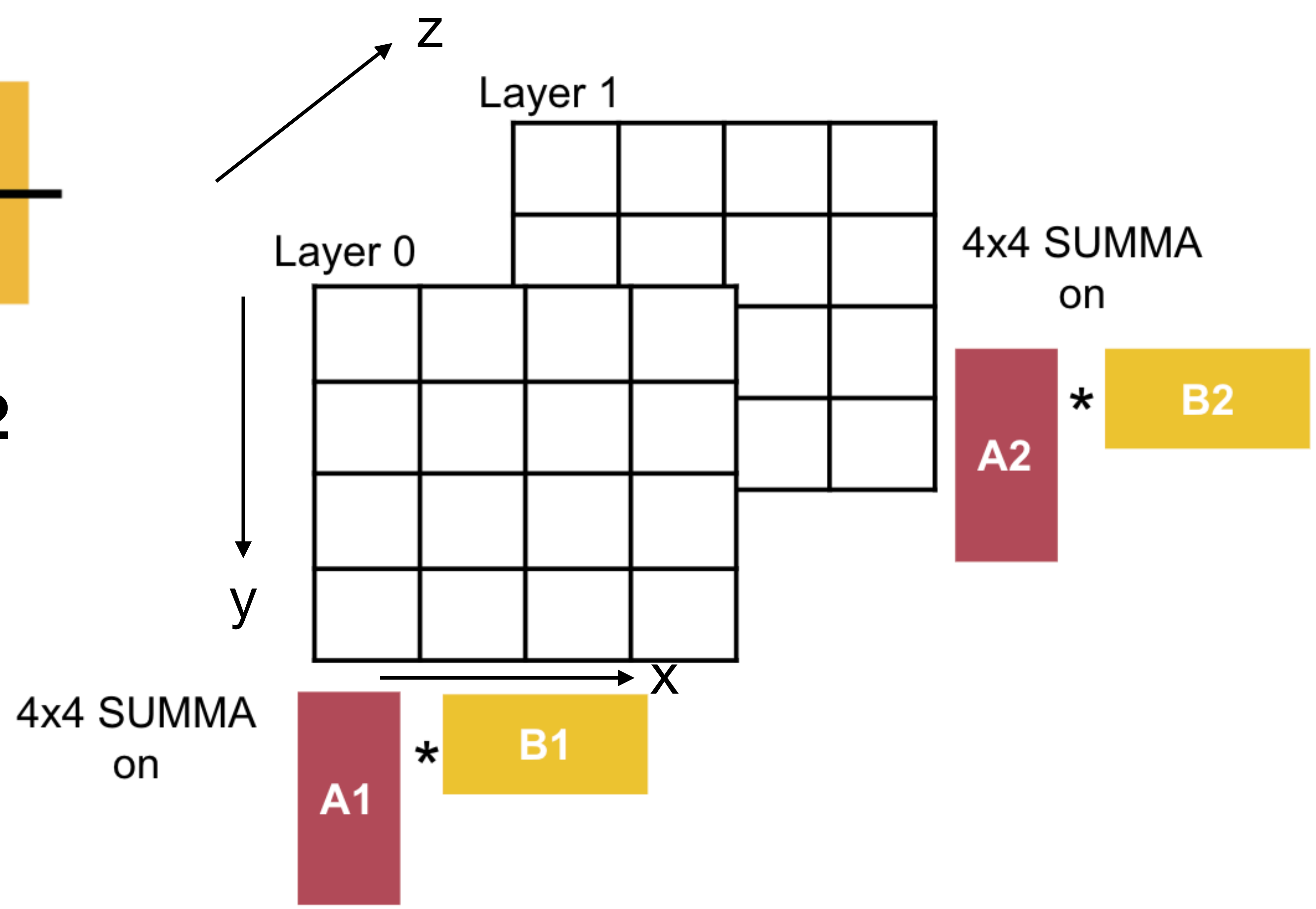
[Solomonik et al. ’11]

3D SUMMA : Algorithm Overview

Example (M= 4, m=2, P=32)

 # layers = # outer products

= A1*B1 + A2*B2

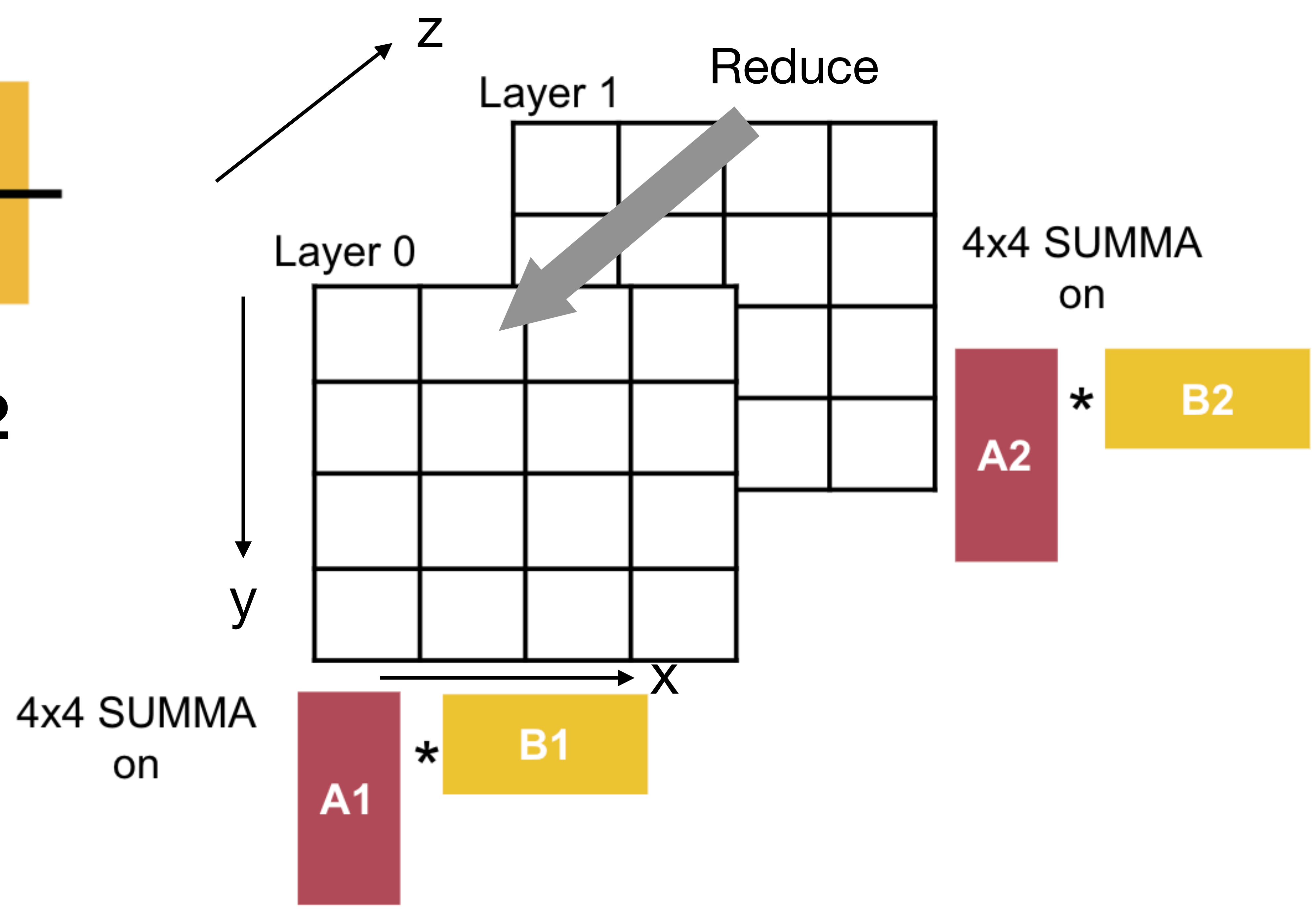


3D SUMMA : Algorithm Overview

Example (M= 4, m=2, P=32)

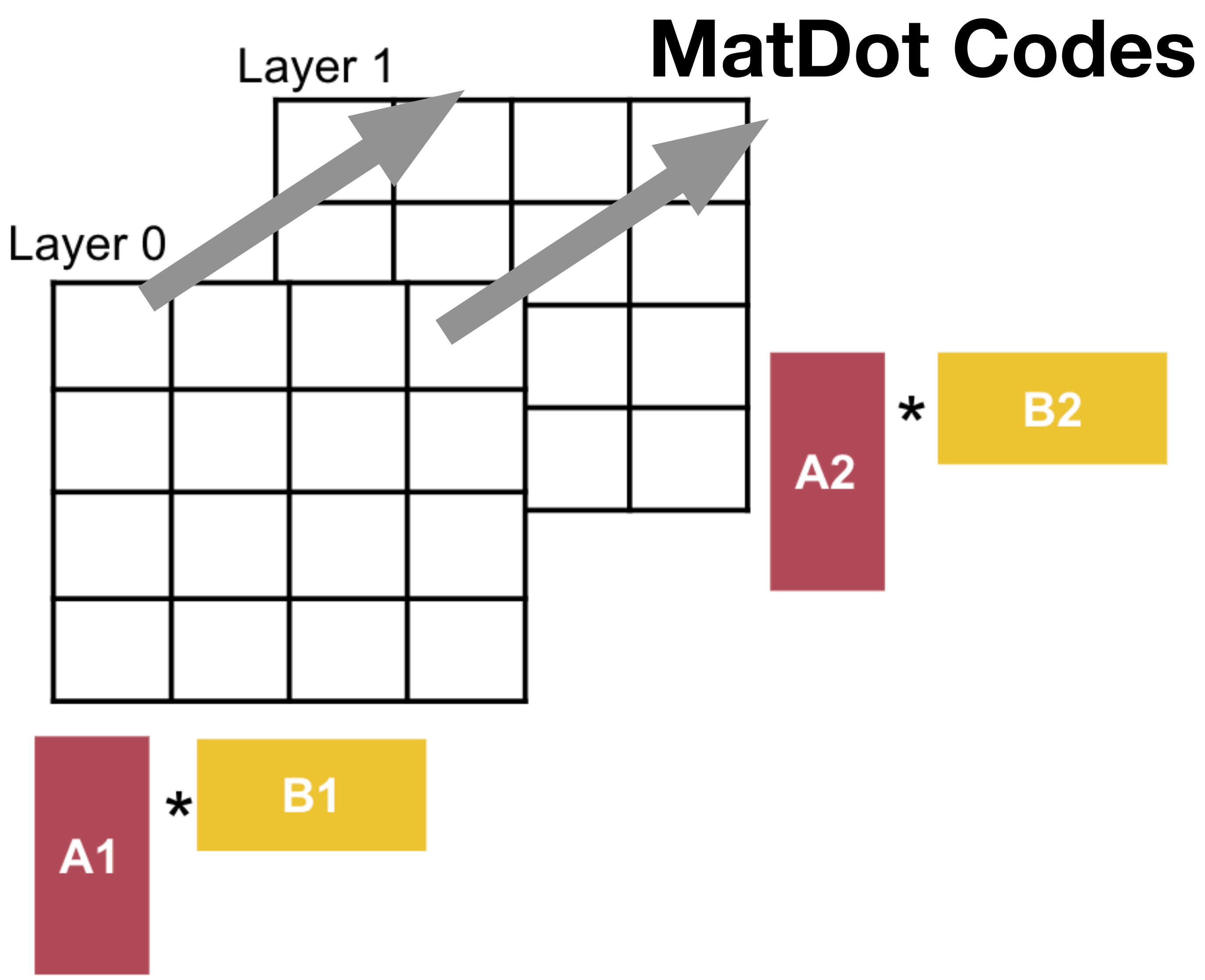
 # layers = # outer products

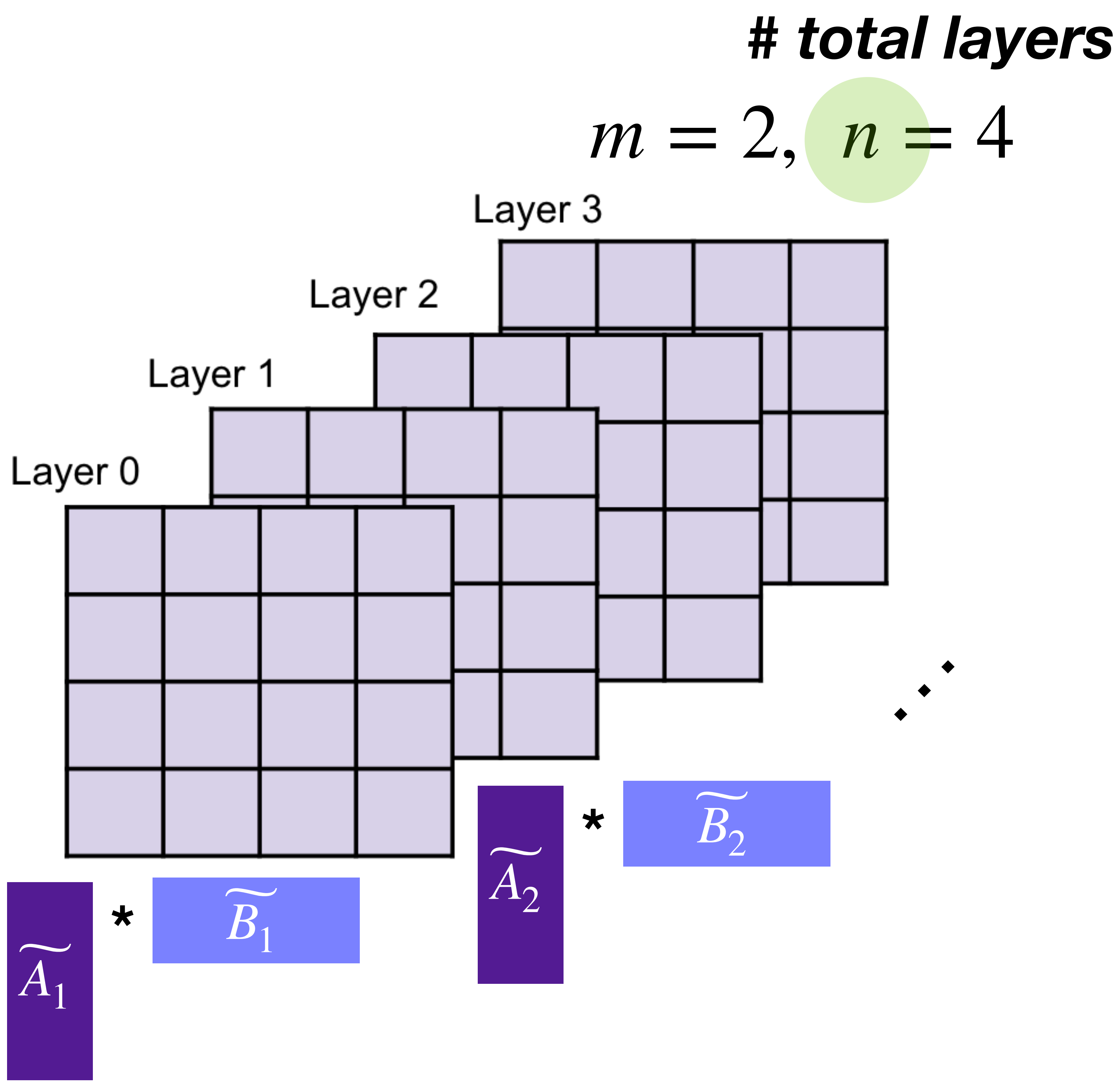
= A1*B1 + A2*B2



3D *Coded* SUMMA

- Apply MatDot Codes across layers



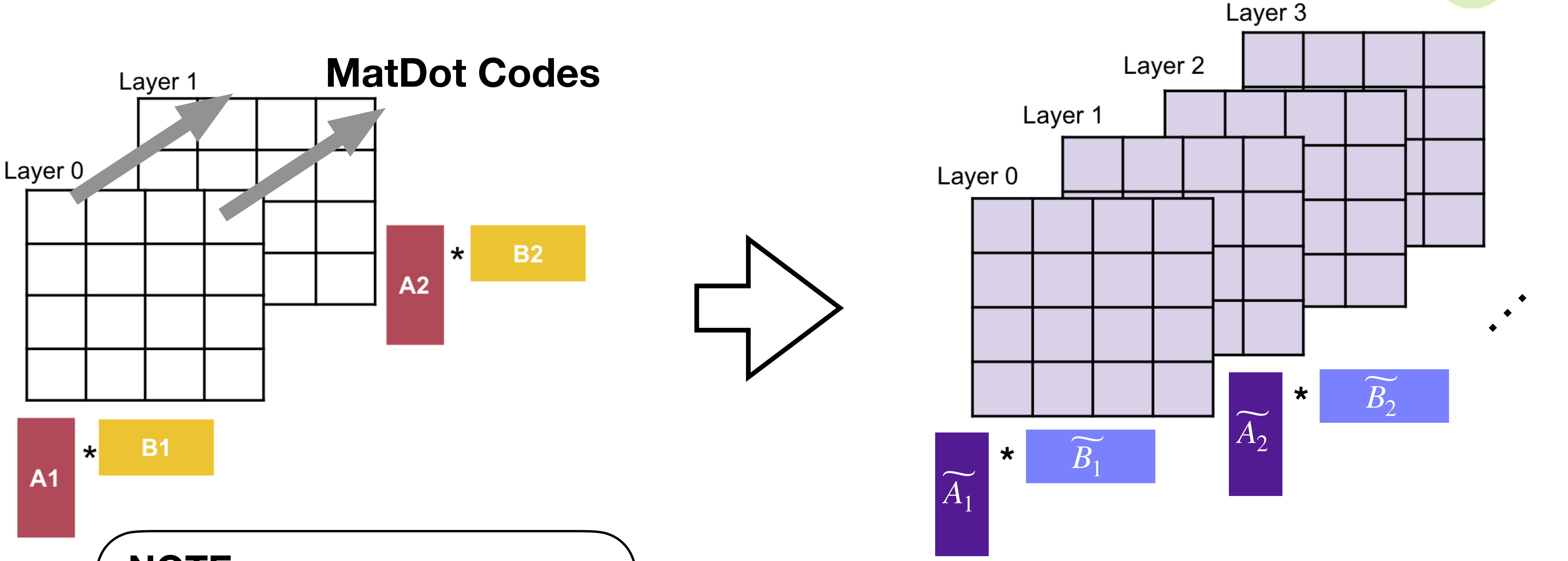


3D *Coded* SUMMA

- Apply MatDot Codes across layers

 # total layers

$m = 2, n = 4$



NOTE

Recovery threshold of MatDot codes:

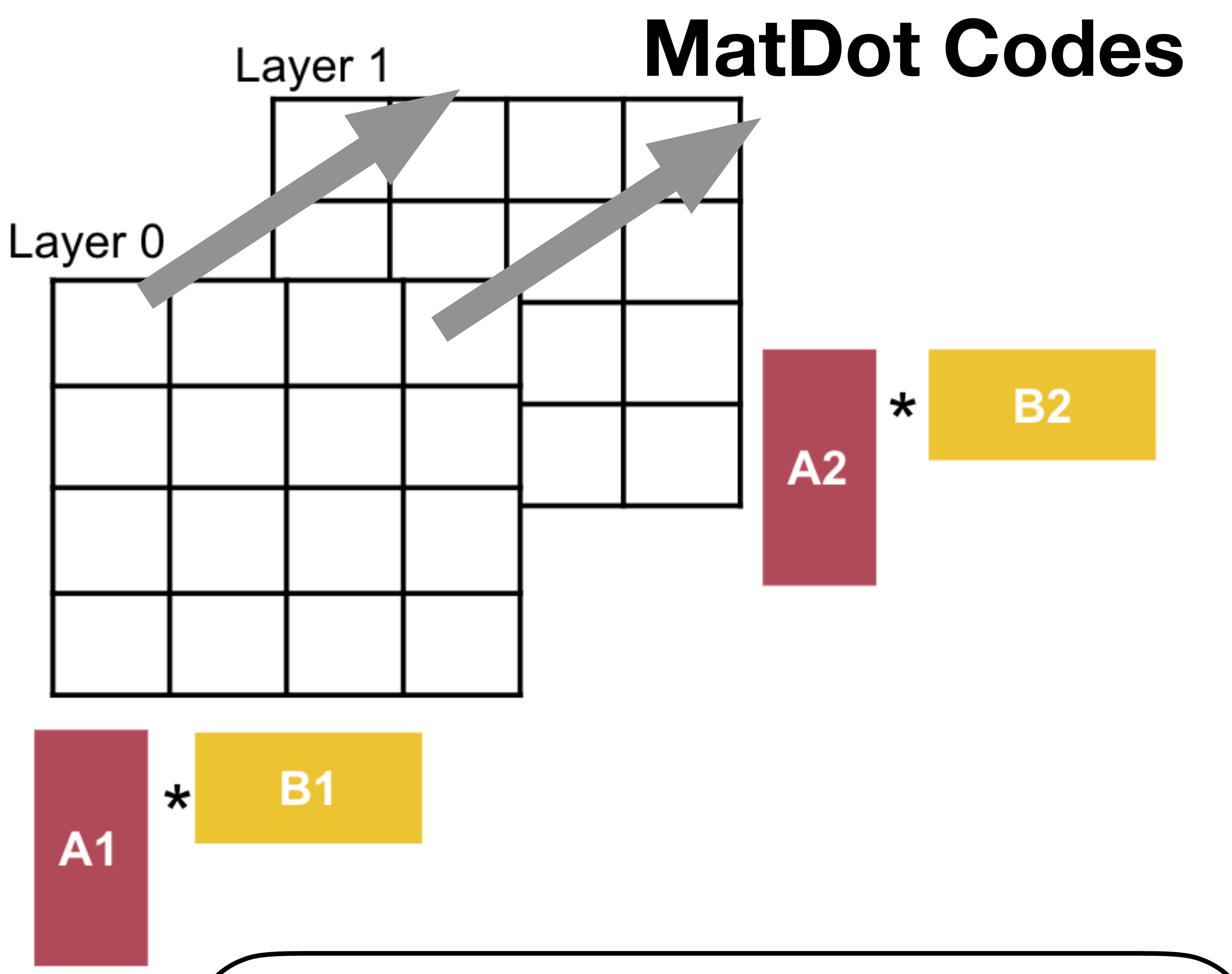
$$K = 2m - 1$$

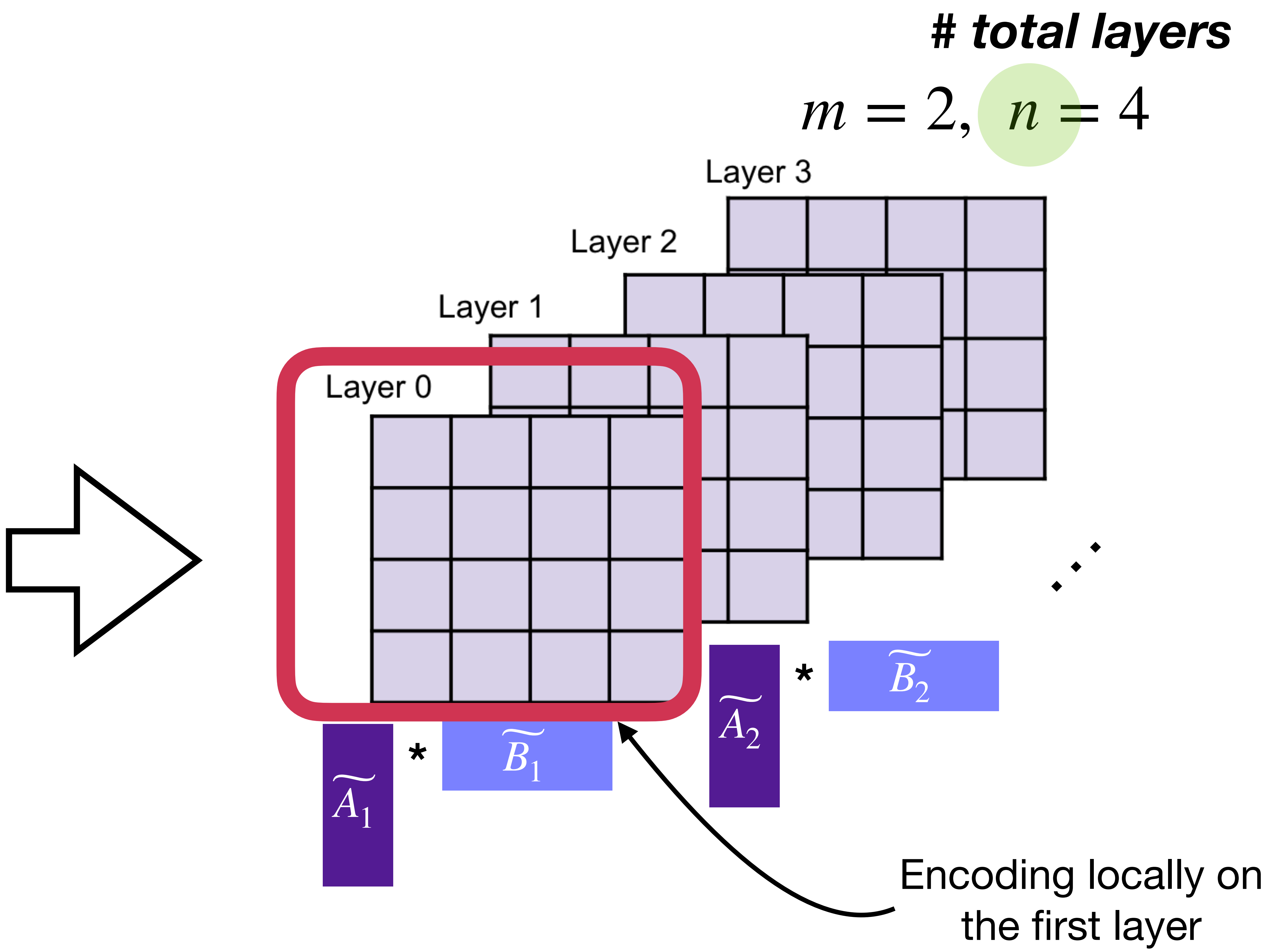
We will add m redundant layers

Total of $n = 2m$ layers

3D *Coded* SUMMA

- Apply MatDot Codes across layers





NOTE

Recovery threshold of MatDot codes:

$$K = 2m - 1$$

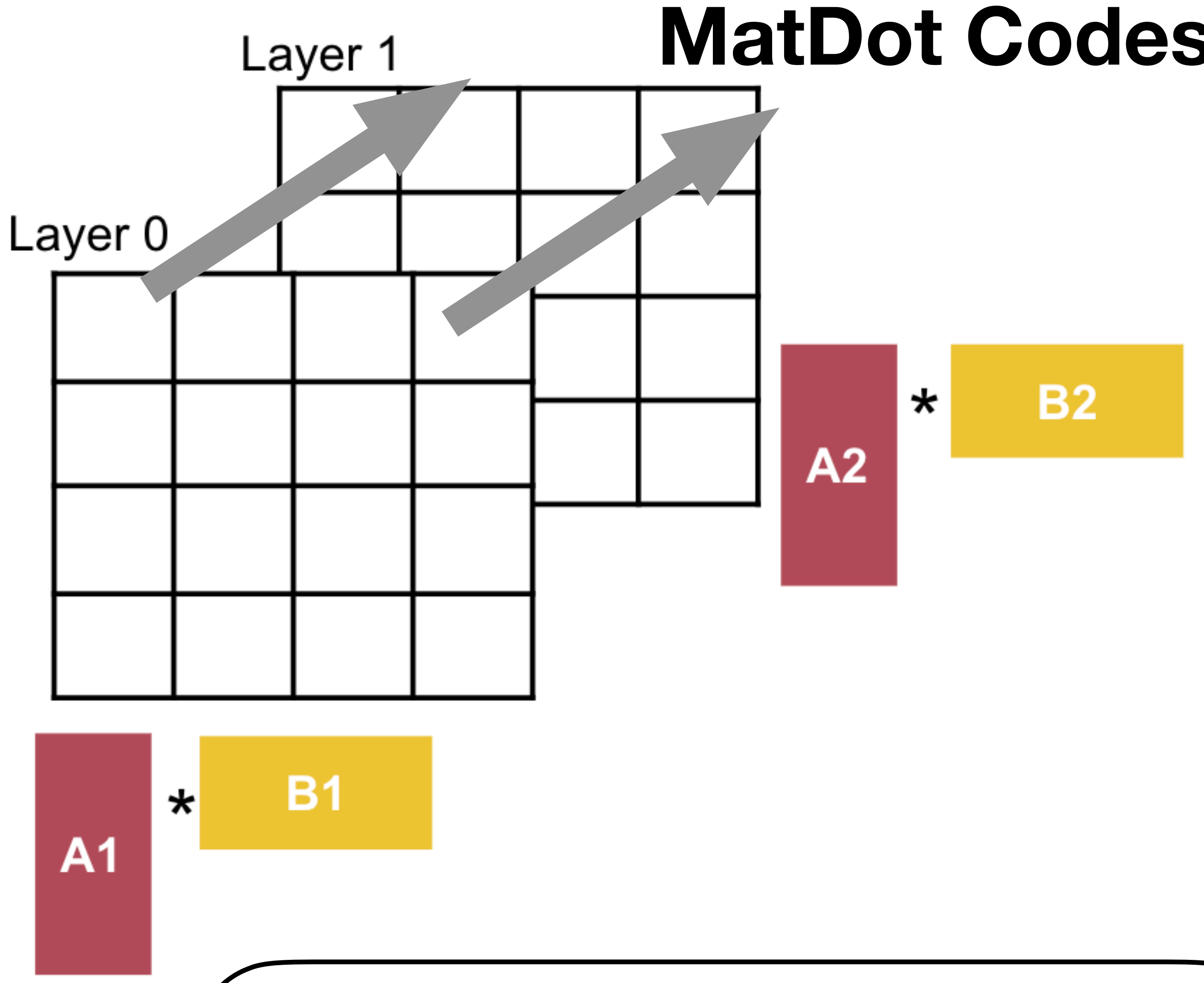
We will add m redundant layers

Total of $n = 2m$ layers

3D *Coded* SUMMA

- Apply MatDot Codes across layers

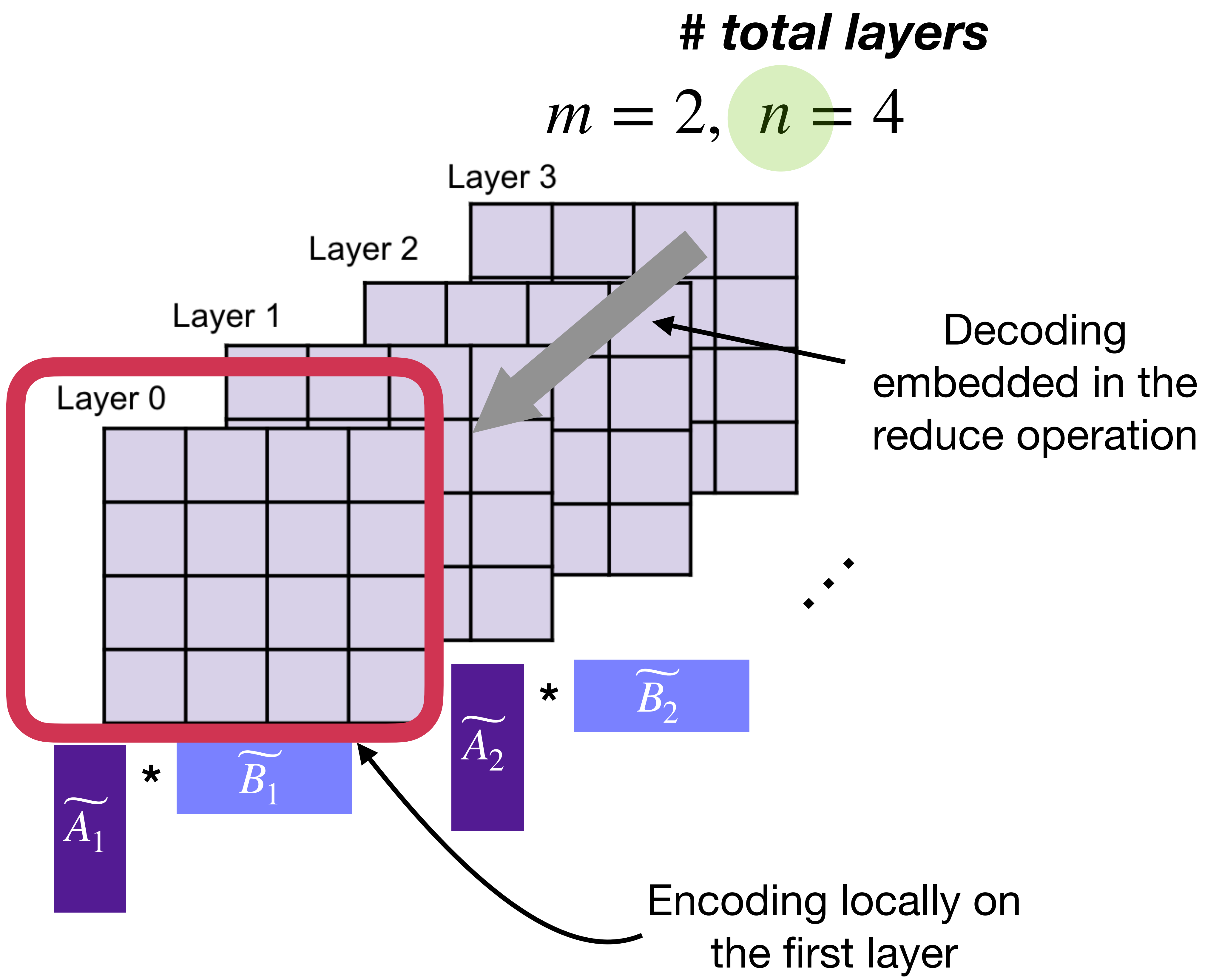
MatDot Codes



NOTE

Recovery threshold of MatDot codes:

$$K = 2m - 1$$

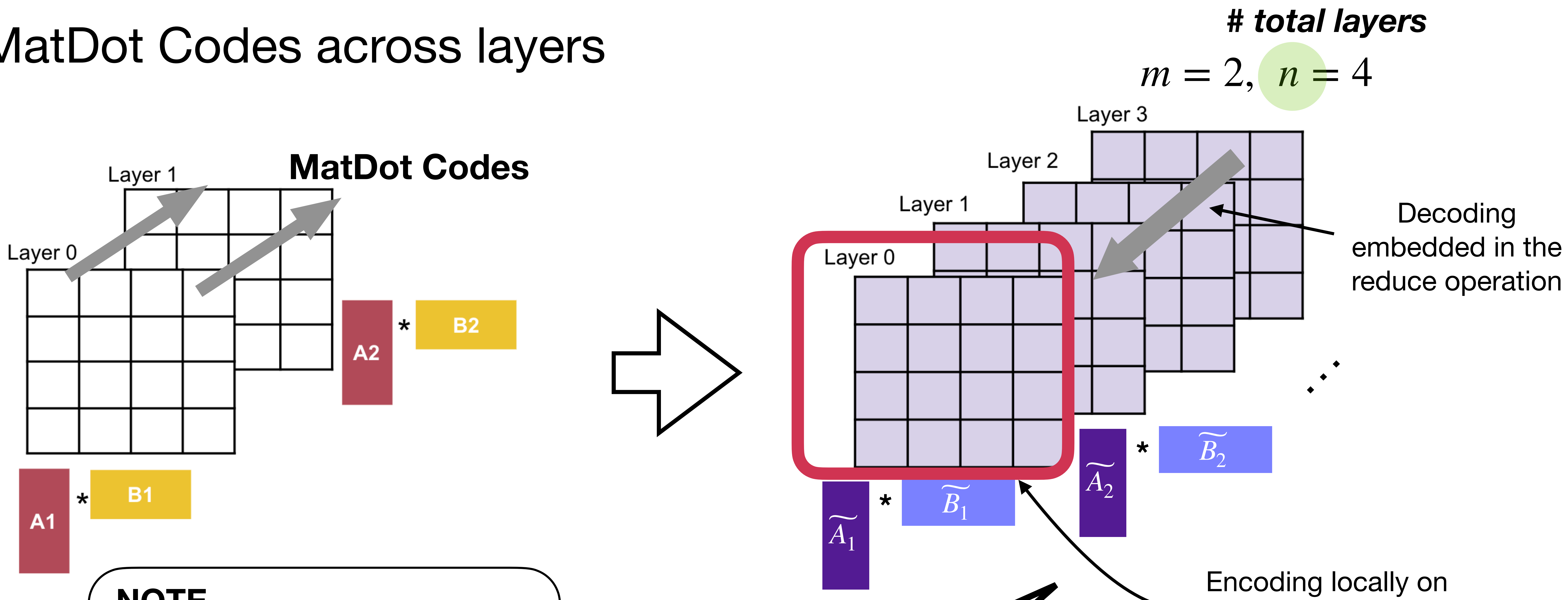


We will add m redundant layers

Total of $n = 2m$ layers

3D *Coded* SUMMA

- Apply MatDot Codes across layers



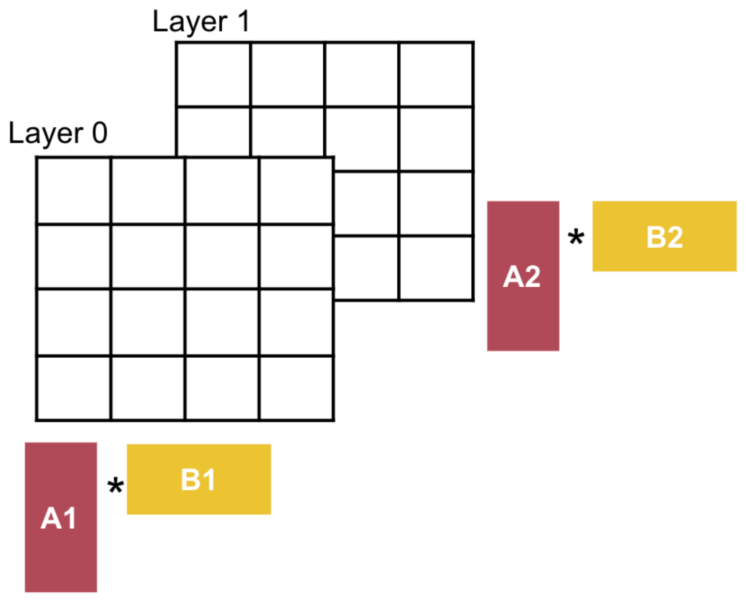
NOTE

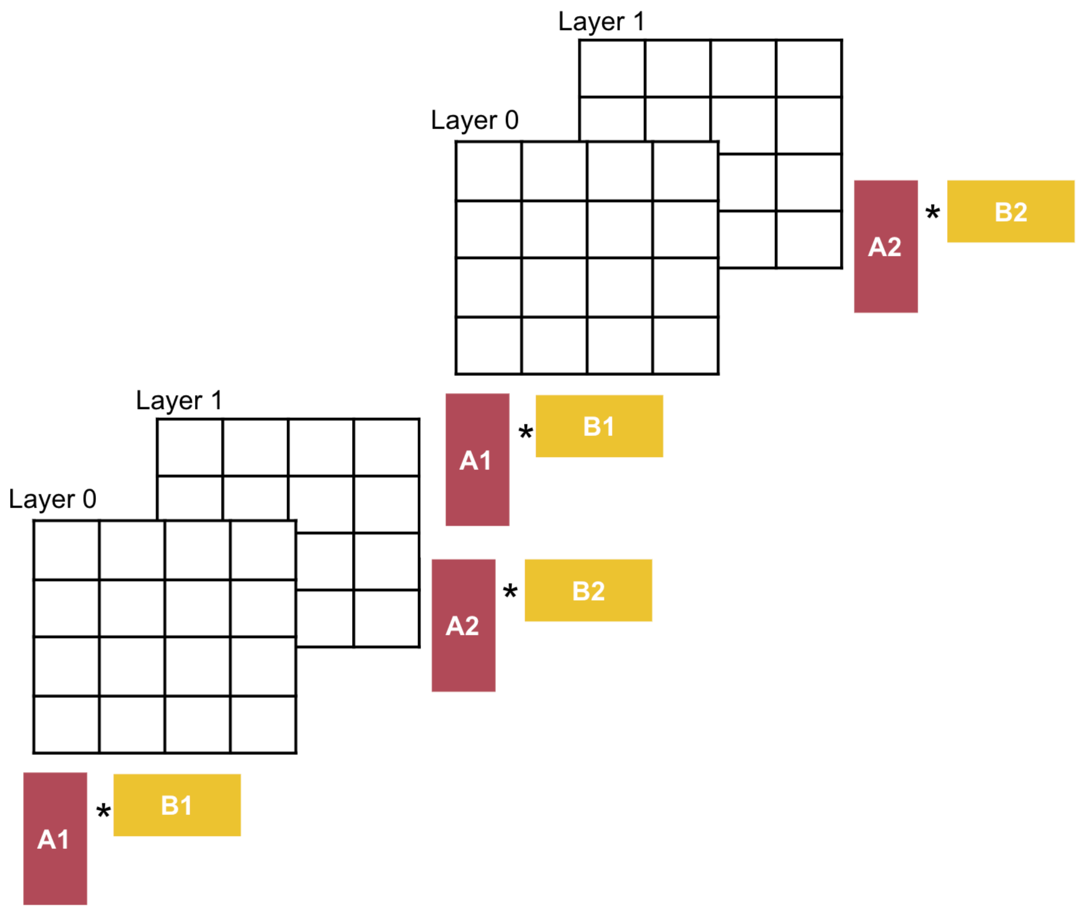
Recovery threshold of MatDot codes:

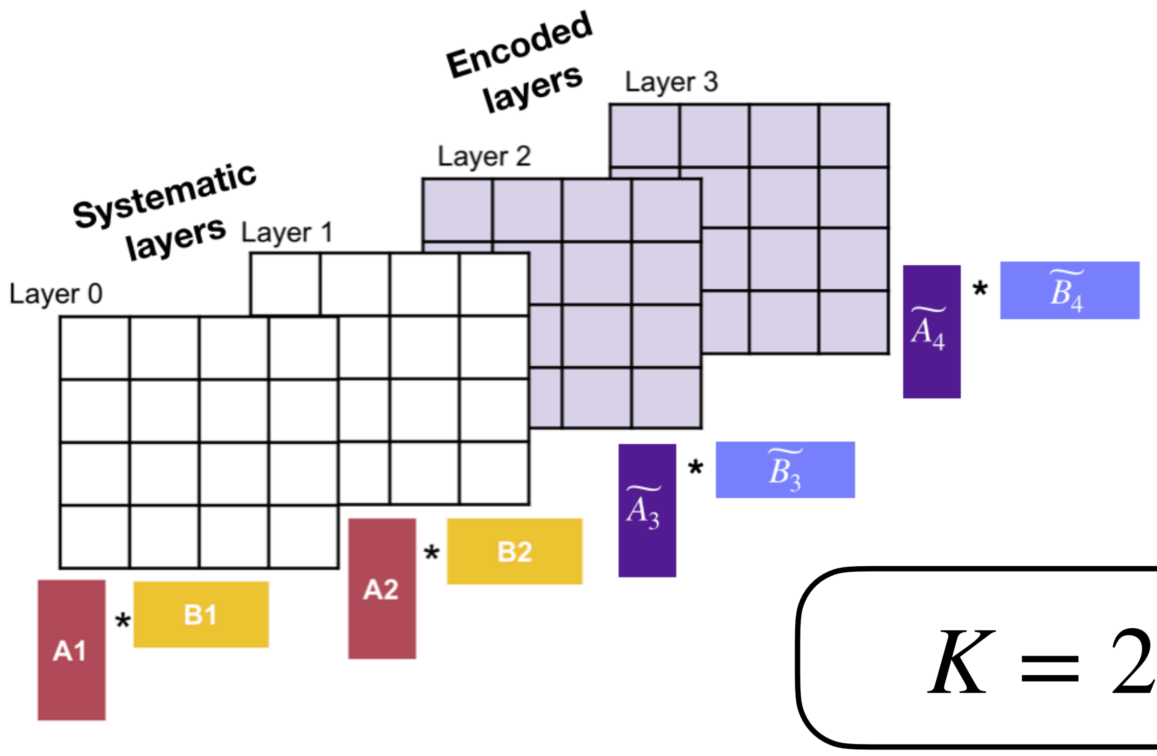
$$K = 2m - 1$$

***Minimal communication overhead** for encoding/ decoding!*

Processor Overhead Comparison



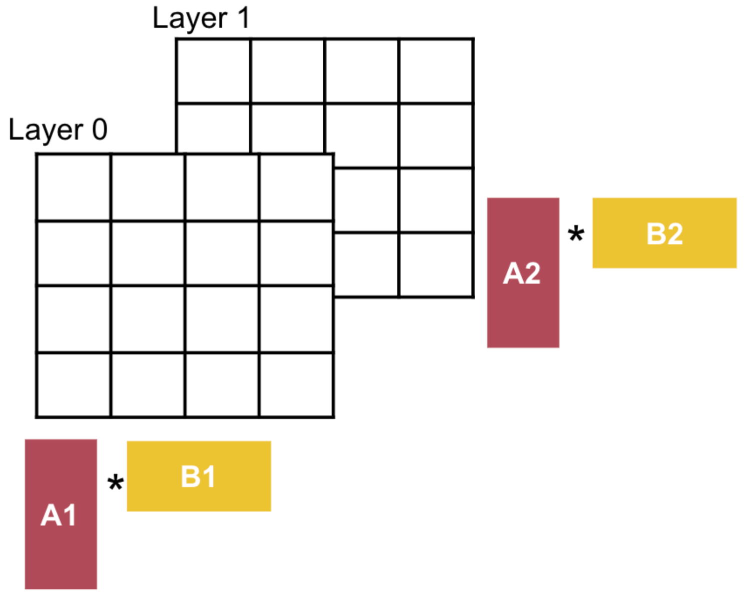


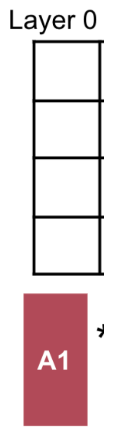


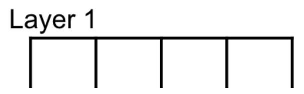
$K = 2m - 1$

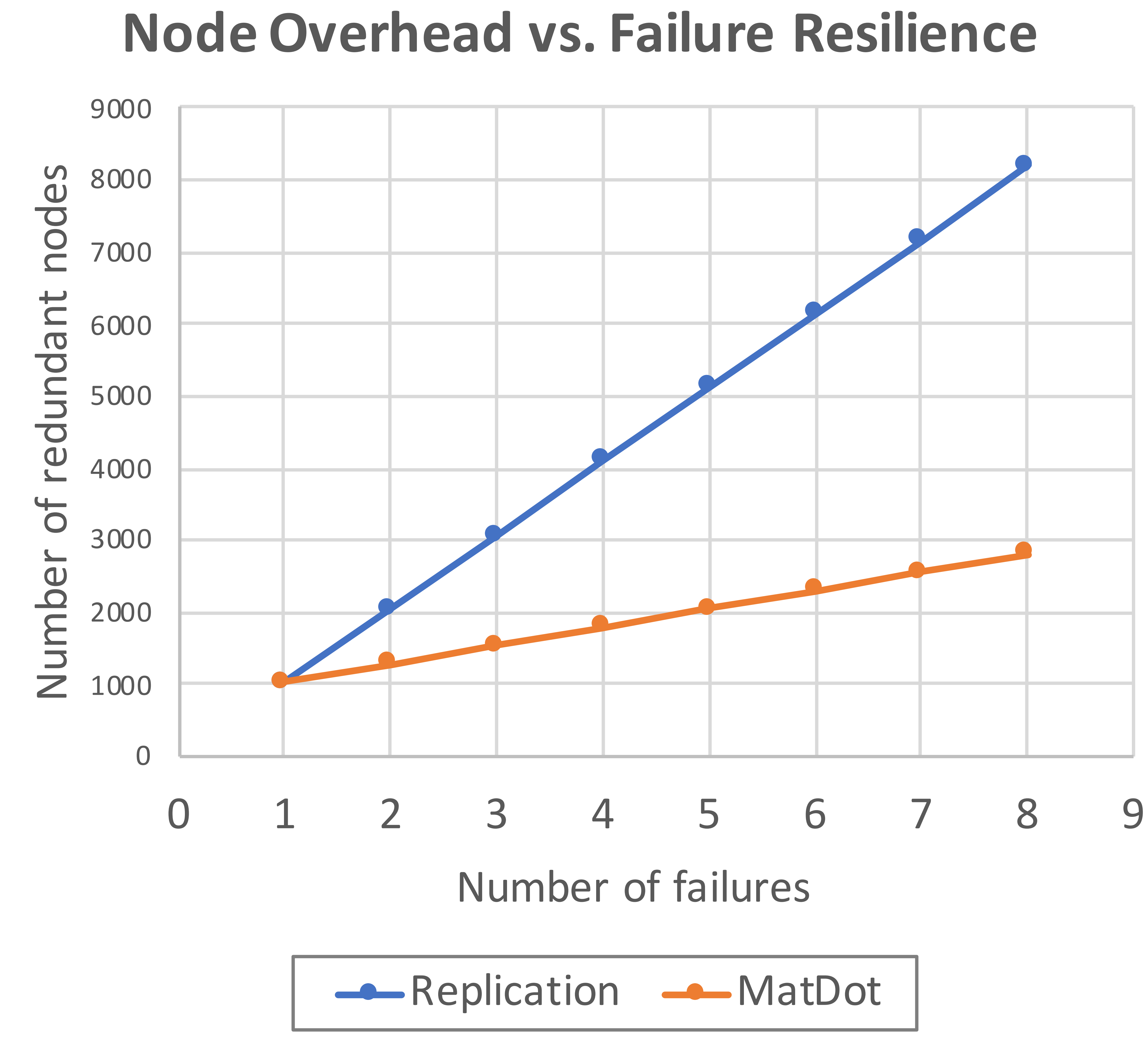
	Uncoded	Replication	MatDot
Single-Failure Resilience	✗	$n = 2m$	$n = K + 1 = 2m$
Two Failure Resilience (Soft Error Correction)	✗	$n = 3m$ ***Ex:*** *(M=32, m=8) n =24* *Total: 24,576 nodes*	$n = K + 2 = 2m + 1$ ***Ex:*** *(M=32, m=8) n = 17* *Total: 17,408 nodes*

Processor Overhead Comparison





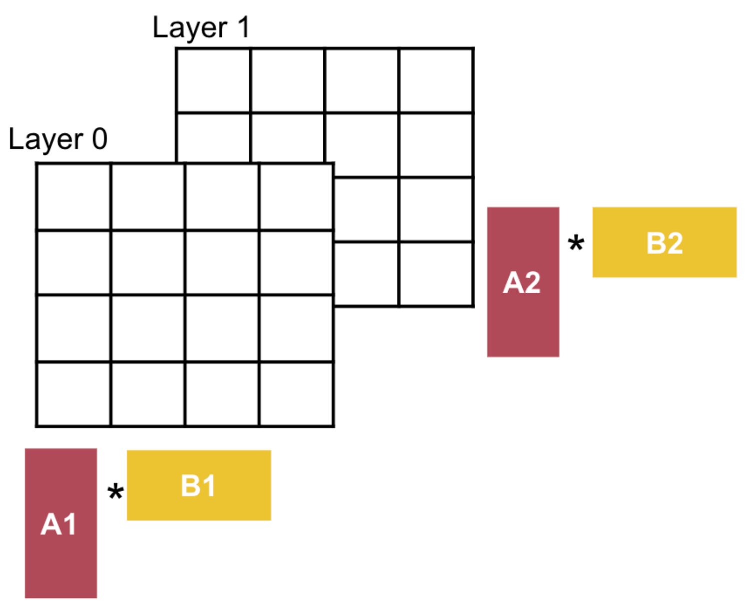


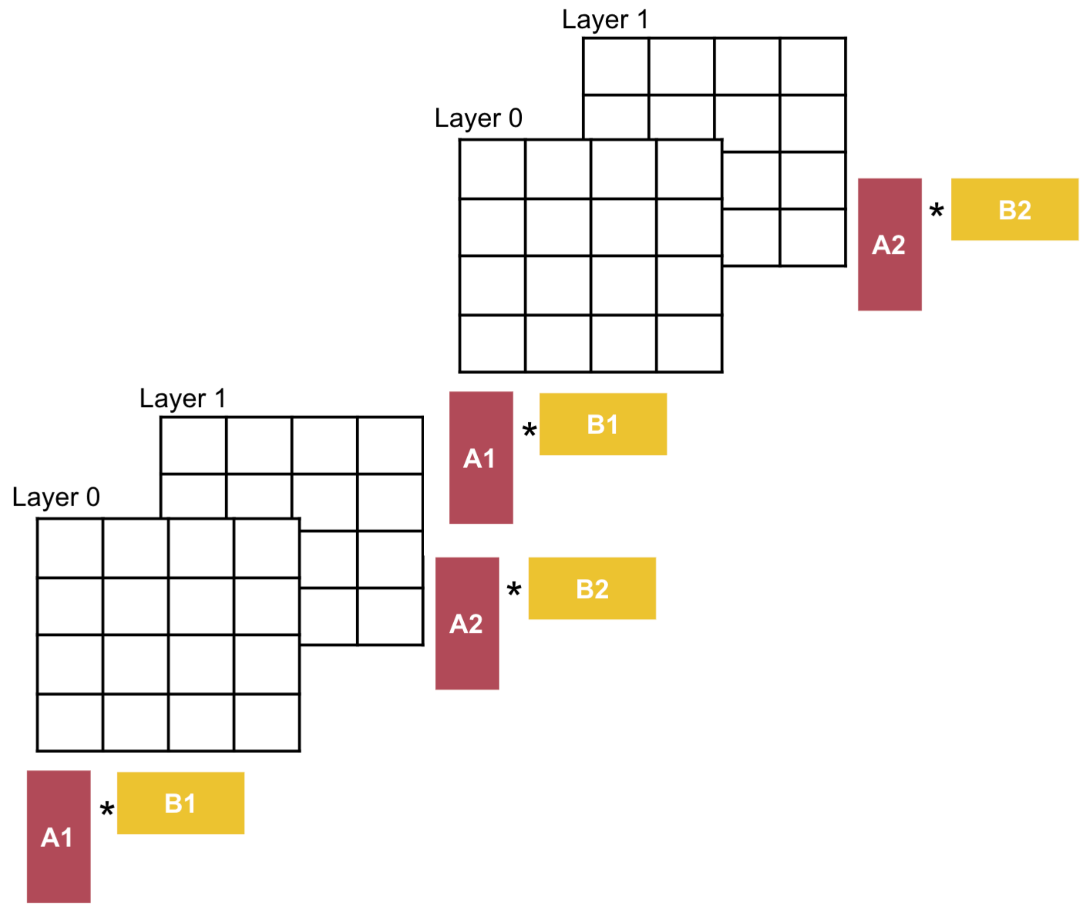


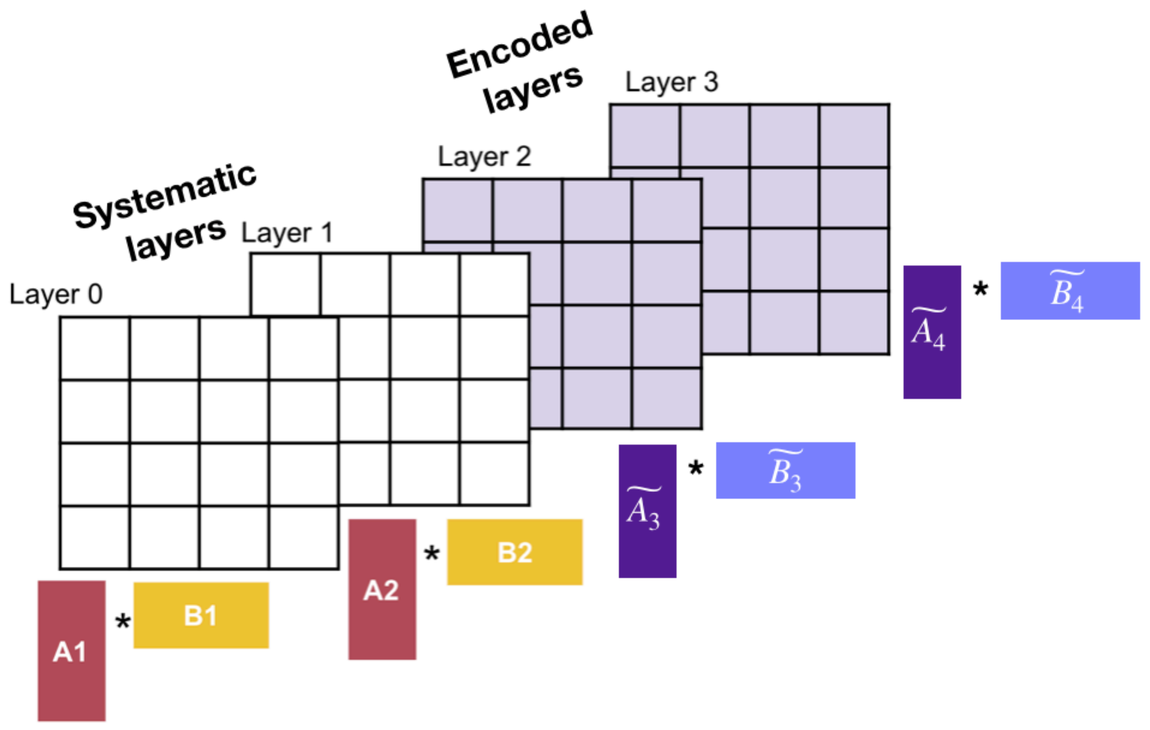
$m - 1$

	Uncoded
Single-Failure Resilience	✗
Two Failure Resilience (Soft Error Correction)	✗

Processor Overhead Comparison







	Uncoded	Replication	MatDot
Single-Failure Resilience			$n = K + 1 = 2m$
Two Failure Resilience (Soft Error Correction)		***Ex:*** *(M=32, m=8) n =24* *Total: 24,576 nodes*	$n = K + 2 = 2m + 1$ ***Ex:*** *(M=32, m=8) n = 17* *Total: 17,408 nodes*

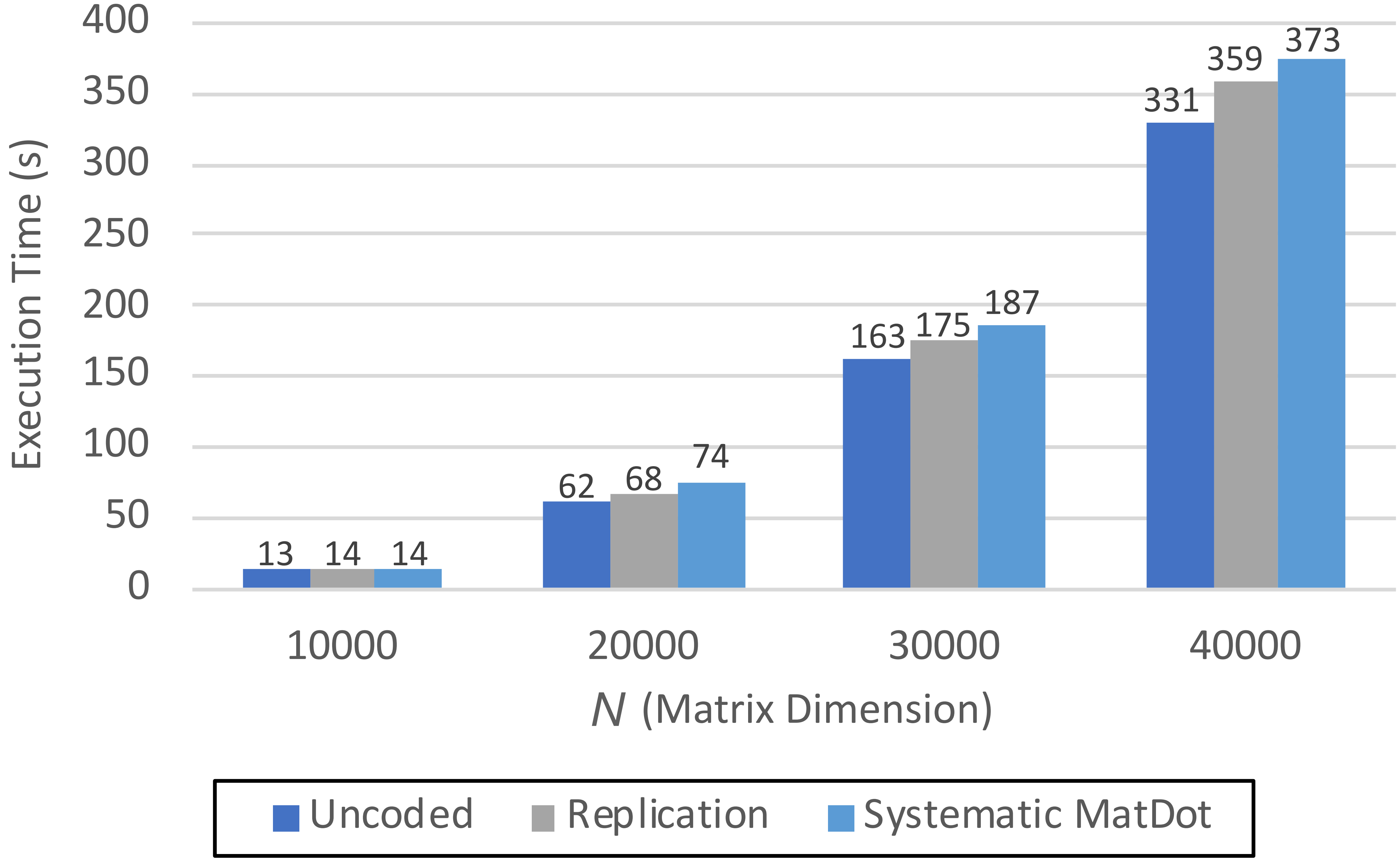
What about latency overhead for encoding and decoding?

Experimental Setup

- Machine Spec (sal9000.ornl.gov):
  - 40 Compute Nodes, two 12-Core AMD Opteron(tm) processors/node, 960 cores in total
  - 64 GB DRAM/node, 2.5 TB DRAM in total
  - Gigabit Ethernet Network Interconnect under one switch
- One core = One MPI process = One logical node on the grid
- We assume that we know which node has failed.
- Recorded Latencies
  - Memory Allocation
  - MatDot Encoding
  - MPI Scatter
  - SUMMA Total
  - MatDot Decoding
  - MPI Reduce

Experimental Results

Latency Comparison for (M=8, m=2, n=4)



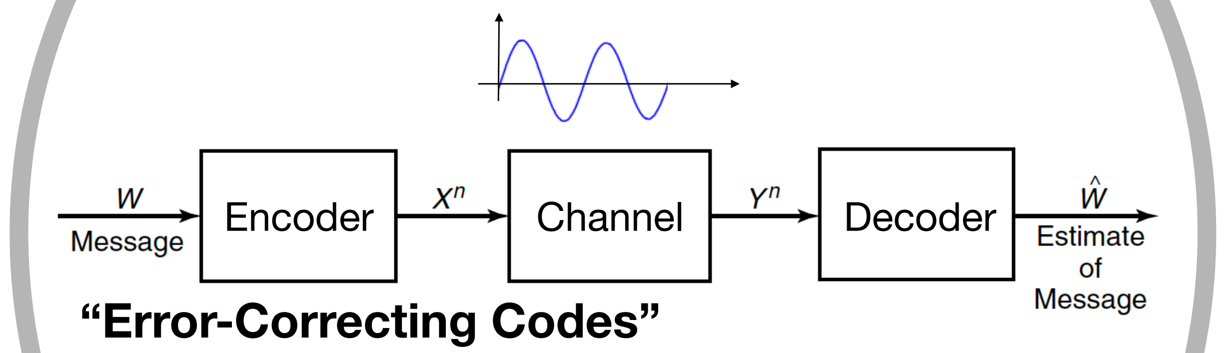
- *10-20% overhead compared to uncoded.*
- *5-10% overhead compared to replication.*

Overhead of encoding/ decoding is small !

Apply tools from Coding Theory
to practical HPC applications

Large-Scale
Computing Algorithms

Information Theory
Coding Theory

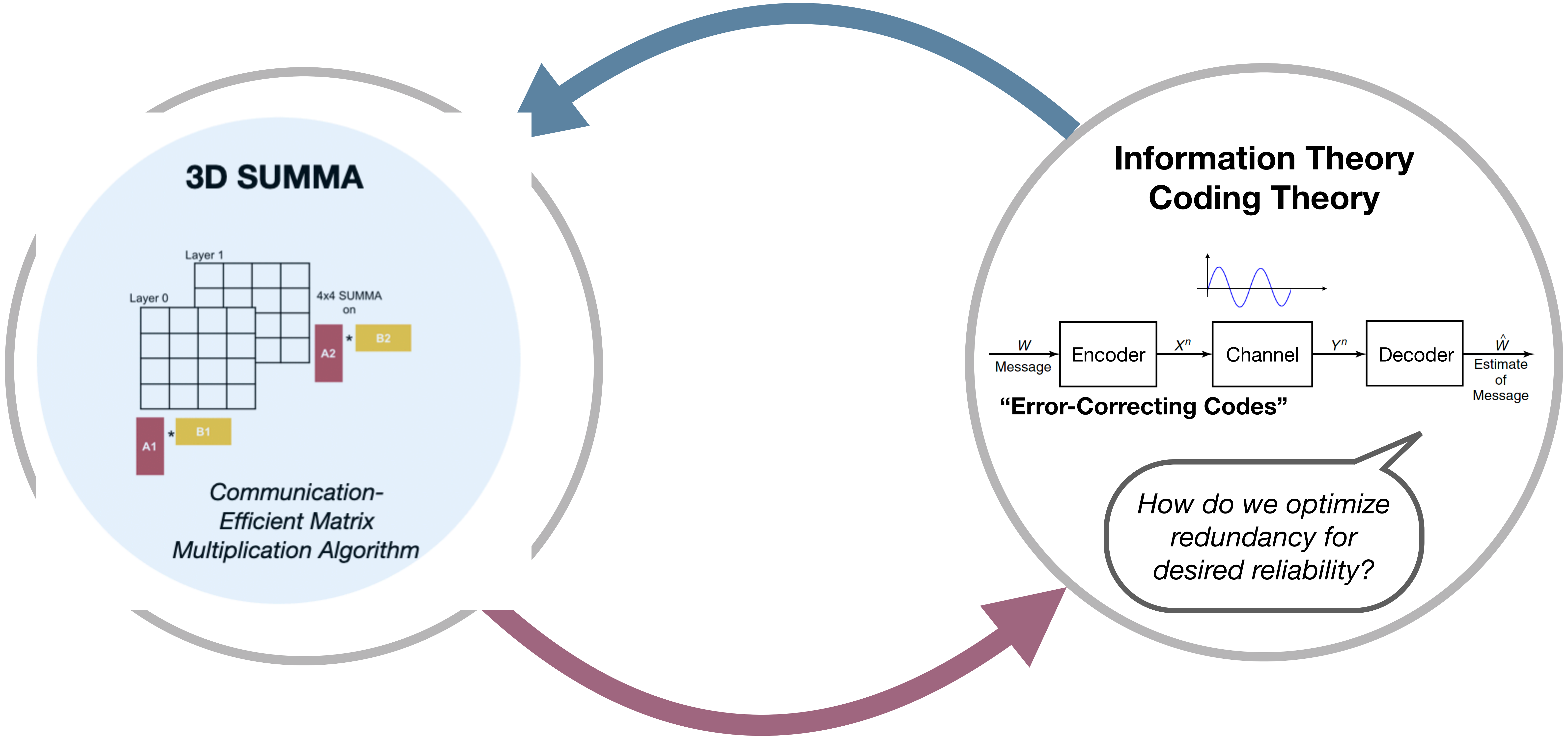


“Error-Correcting Codes”

How do we optimize redundancy for desired reliability?

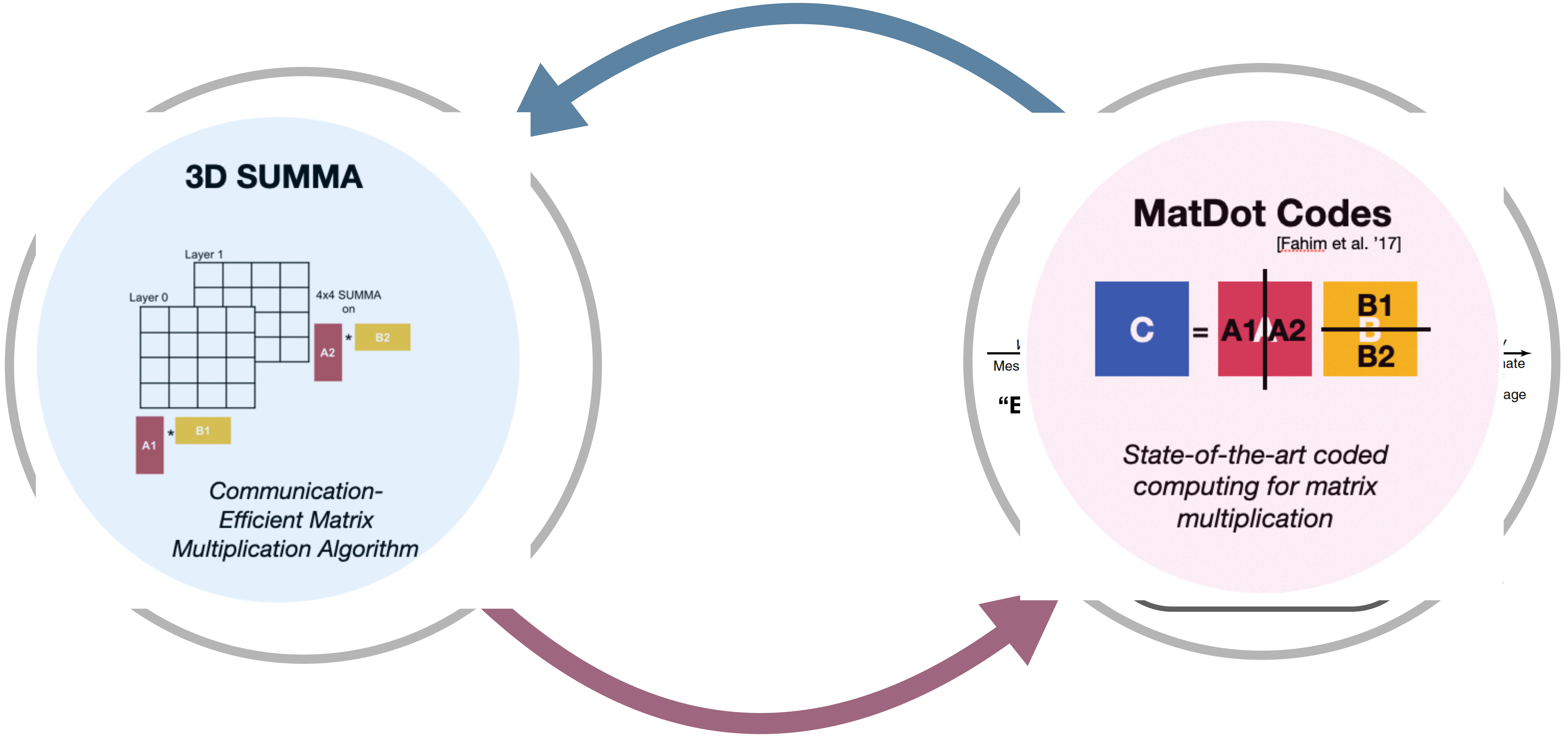
Develop new coding tools on an
abstract computing model

Apply tools from Coding Theory
to practical HPC applications



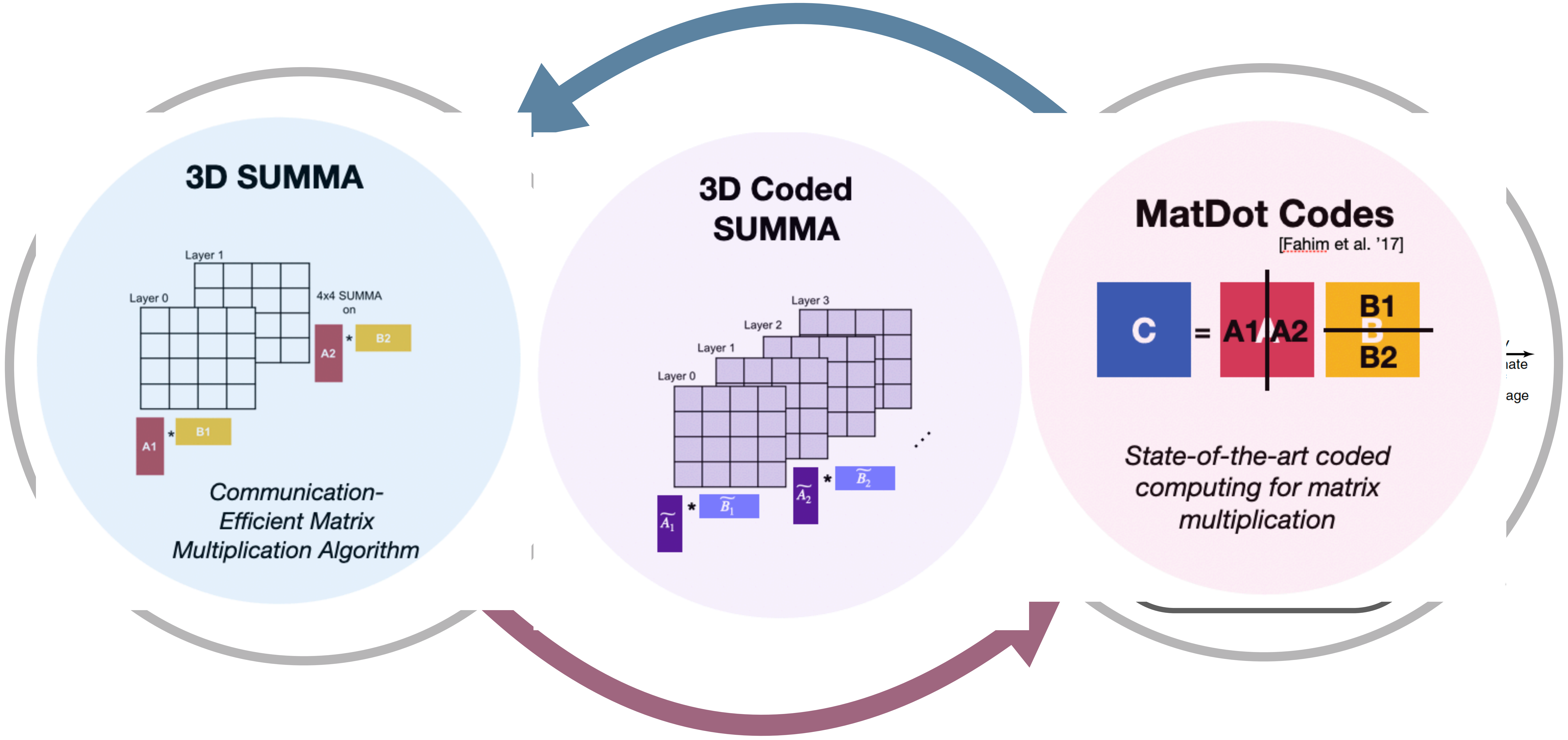
Develop new coding tools on an
abstract computing model

Apply tools from Coding Theory to practical HPC applications



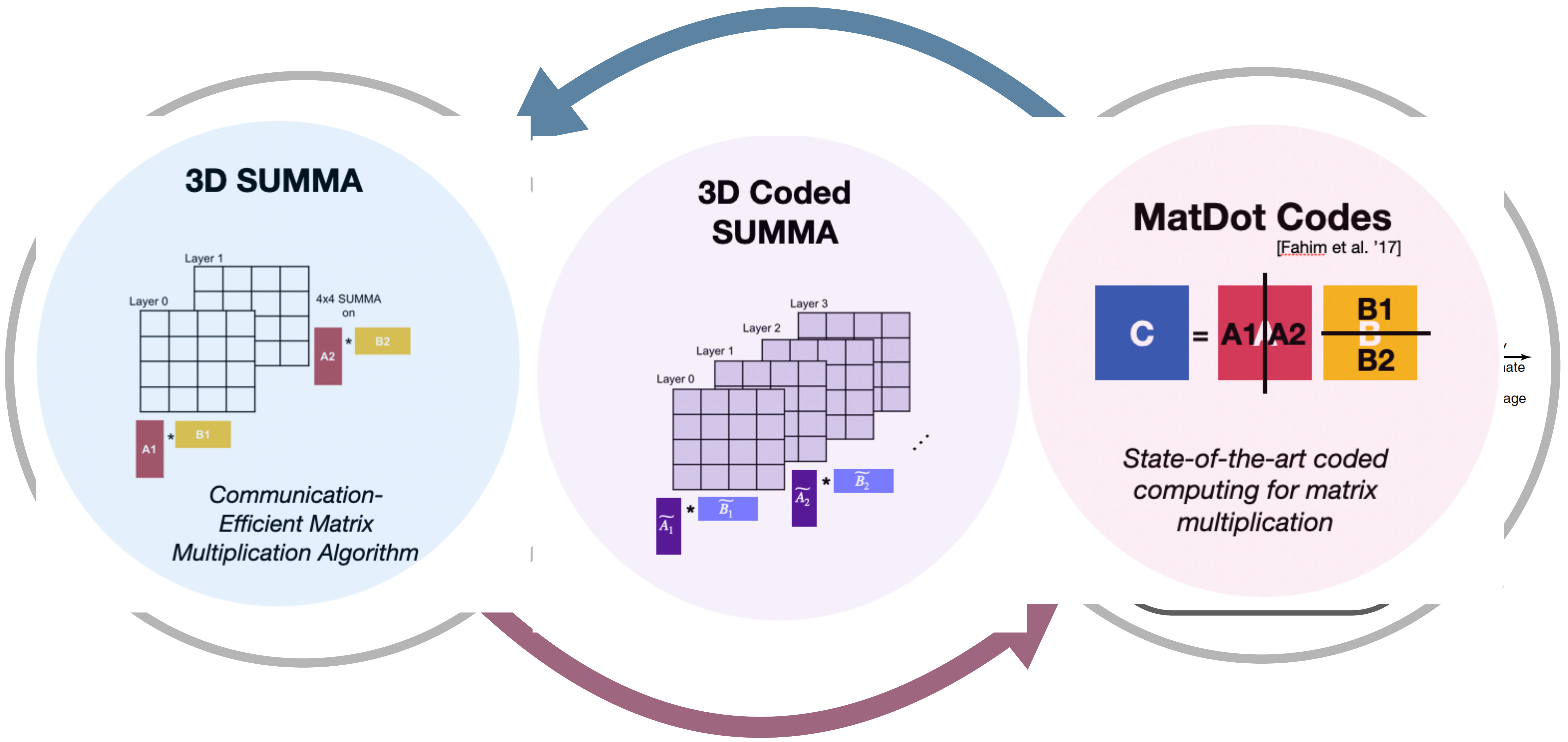
Develop new coding tools on an abstract computing model

Apply tools from Coding Theory
to practical HPC applications

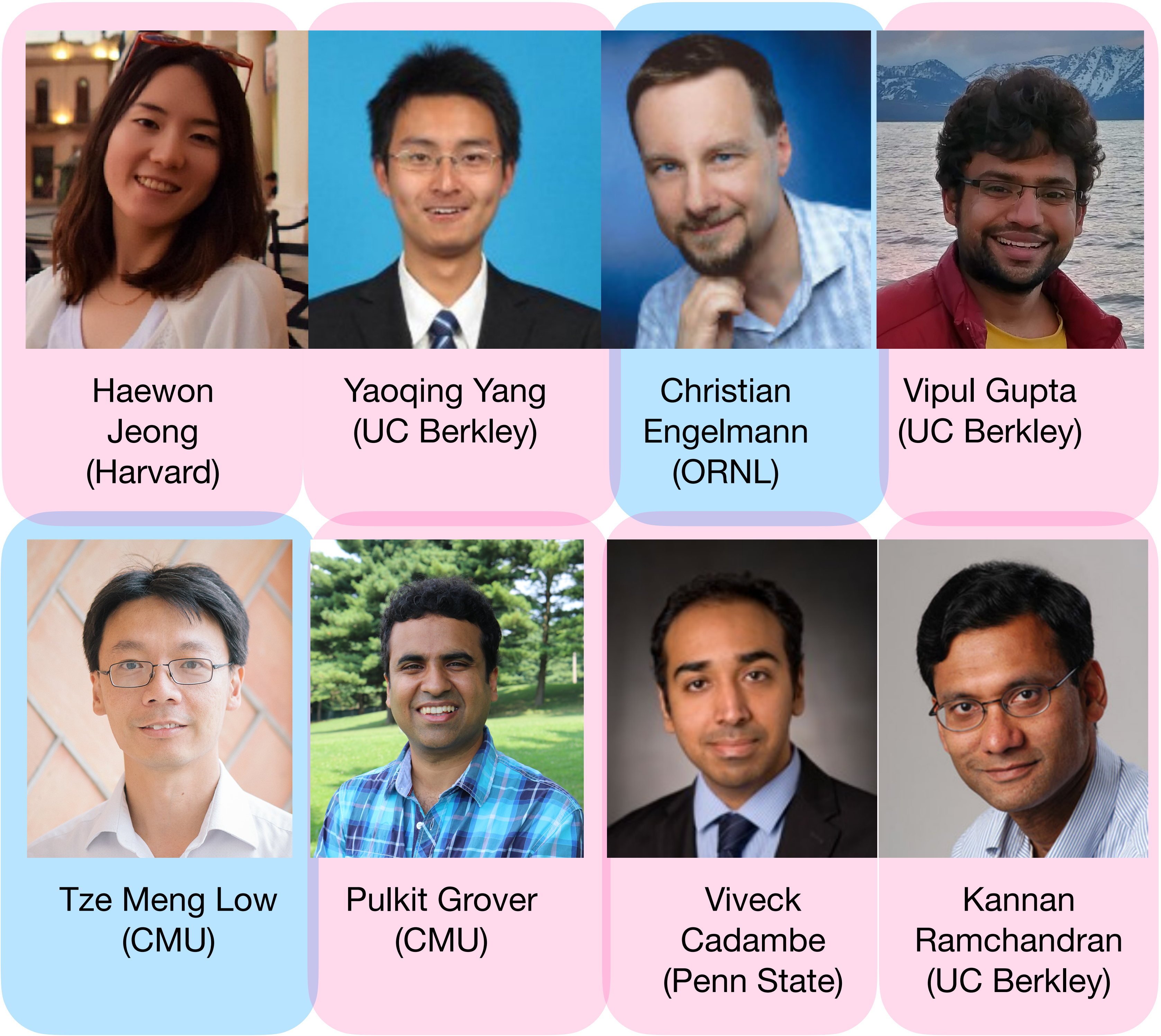


Develop new coding tools on an
abstract computing model

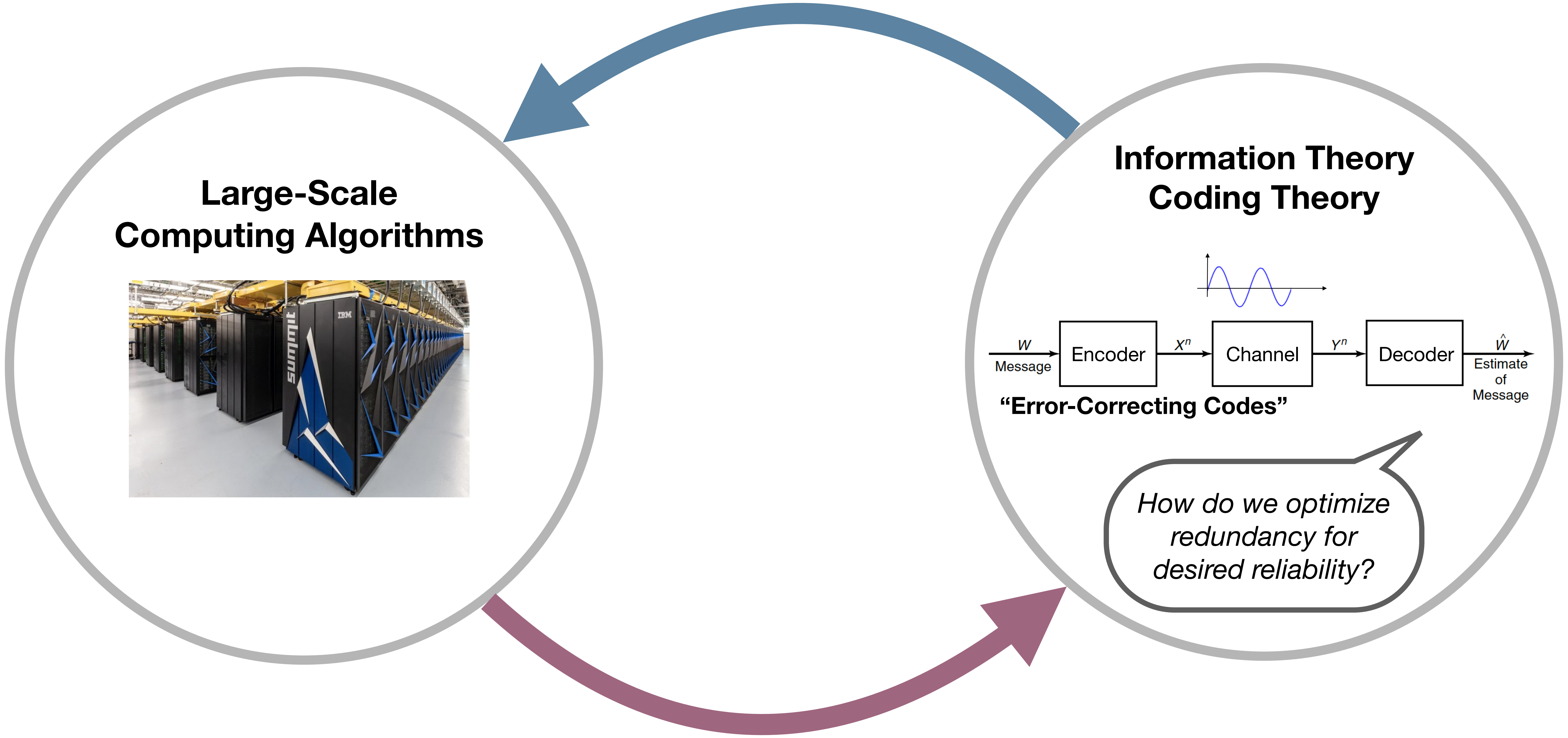
Apply tools from Coding Theory
to practical HPC applications



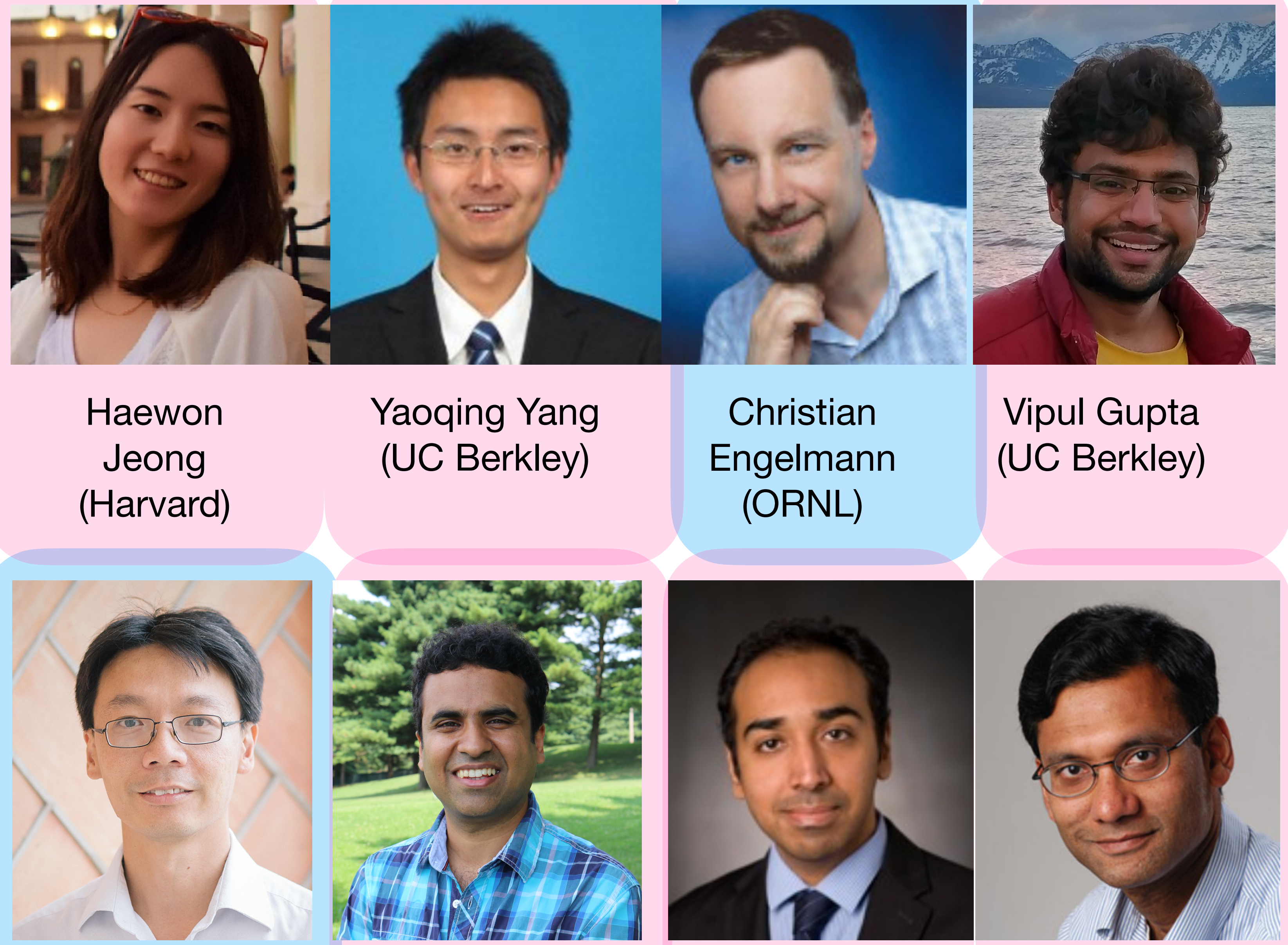
Develop new coding tools on an
abstract computing model



Apply tools from Coding Theory
to practical HPC applications



Develop new coding tools on an
abstract computing model



Haewon Jeong (Harvard)

Yaoqing Yang (UC Berkley)

Christian Engelmann (ORNL)

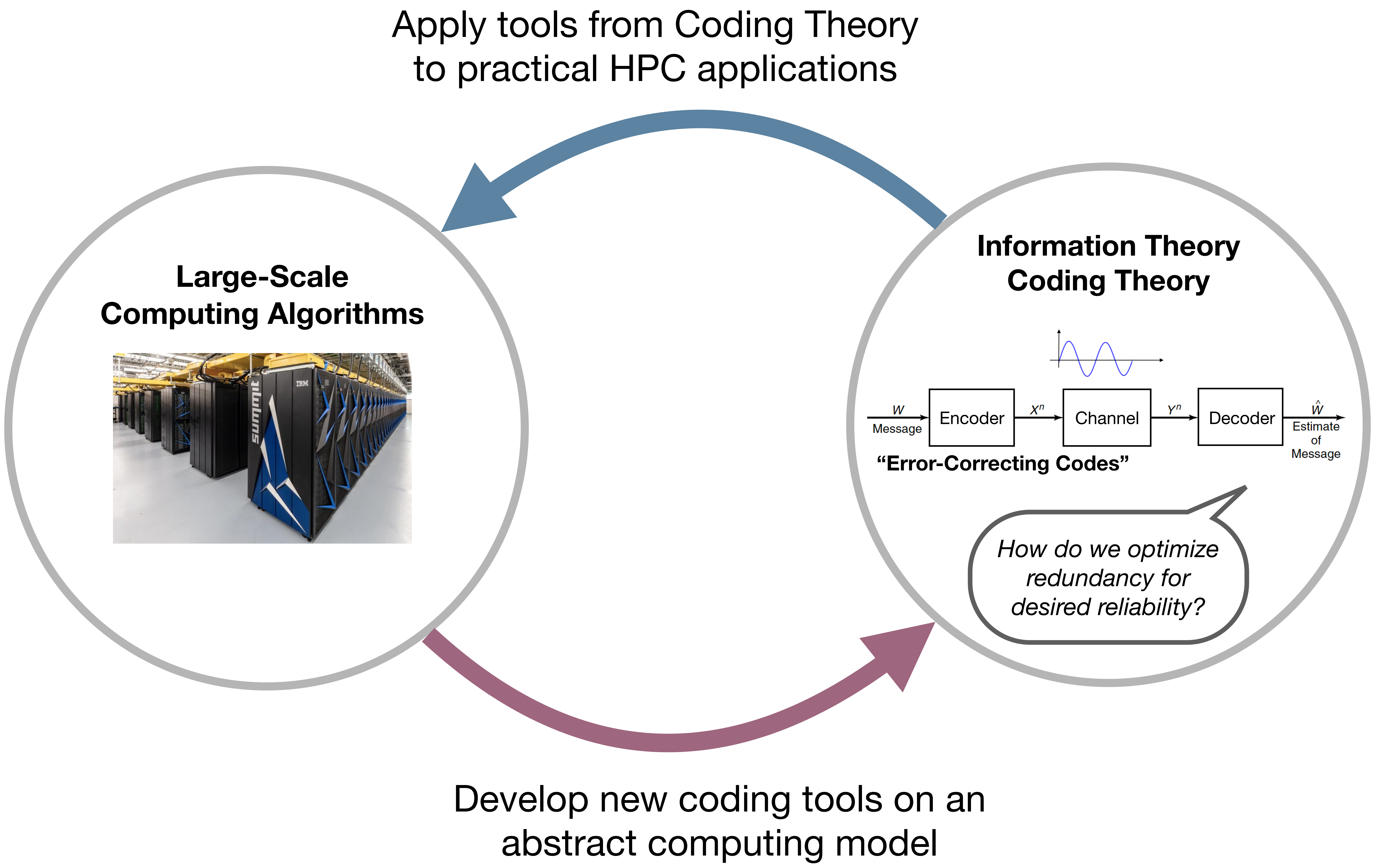
Vipul Gupta (UC Berkley)

Tze Meng Low (CMU)

Pulkit Grover (CMU)

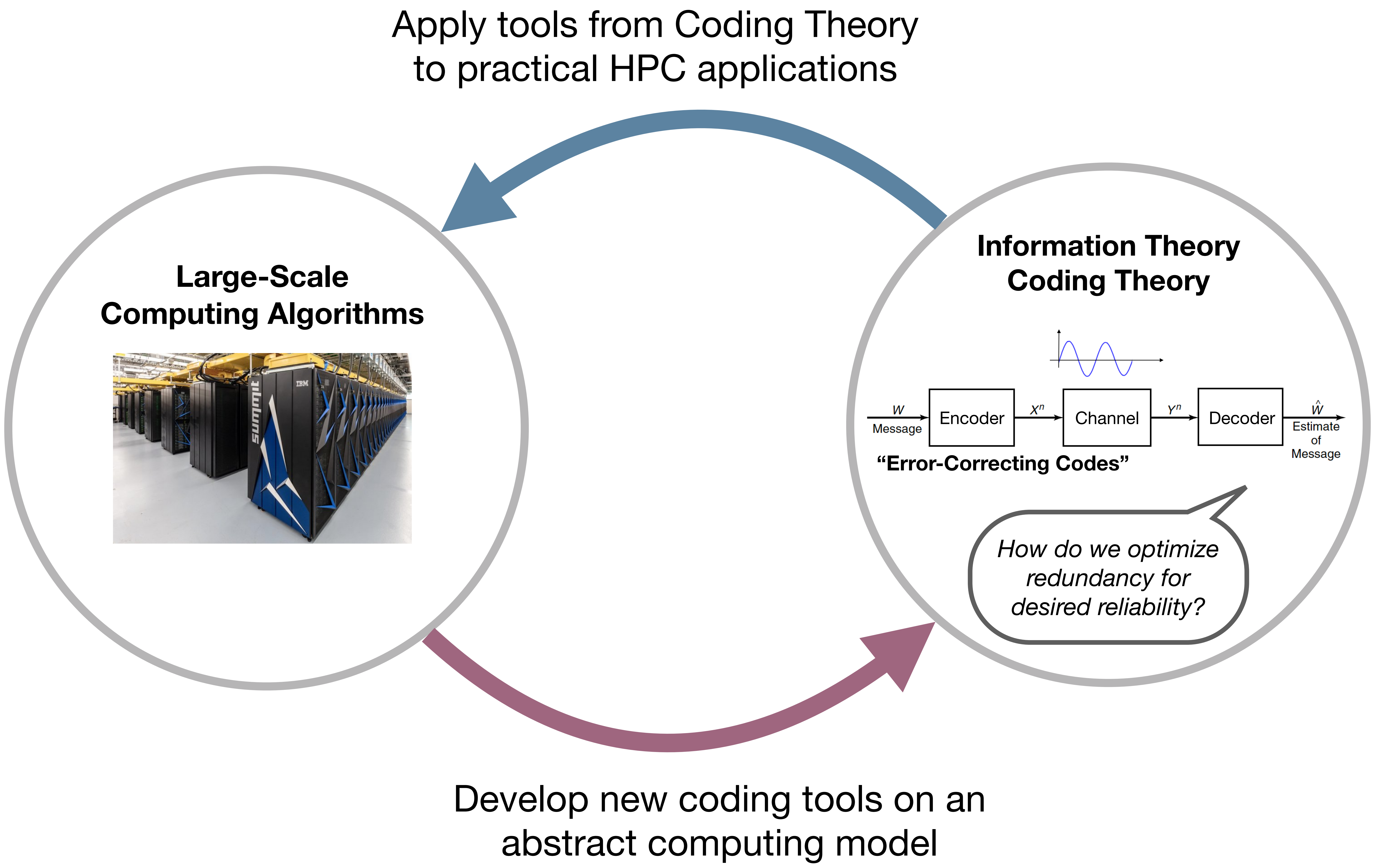
Viveck Cadambe (Penn State)

Kannan Ramchandran (UC Berkley)



Future Directions

- Coding for other computation primitives. (ex) Sparse matrix operations
- Coding to reduce communication/ storage access.
- Approximate coding for ML applications.
- Experiments on the gradient coding ideas on the HPC setting.



Future Directions

- Coding for other computation primitives. (ex) Sparse matrix operations
- Coding to reduce communication/ storage access.
- Approximate coding for ML applications.
- Experiments on the gradient coding ideas on the HPC setting.

More questions? Haewon Jeong (haewon@seas.harvard.edu)