Codes for Big Data: Erasure Coding for Distributed Storage

P. Vijay Kumar

Professor,
Department of Electrical Communication Engineering
Indian Institute of Science, Bangalore

The 3rd Annual Storage Developer Conference
Bengaluru

May 25-26, 2017

Thanks go out to

- Paul Talbut and Udayan Singh for the invite

and

- K. Gopinath and Siddhartha Nandi

for being kind enough to suggest my name..

Acknowledgements

Research Collaborators Joint work with:

- Birenjith Sasidharan, Myna Vajha, S. B. Balaji and Nikhil Krishnan (PhD students, IISc)
- Bhagyashree Puranik, Ganesh Kini and Vinayak Ramkumar (MTech students, IISc)
- Srinivasan Narayanamurthy, Syed Hussain and Siddhartha Nandi (NetApp ATG, Bengaluru, India)

Organization

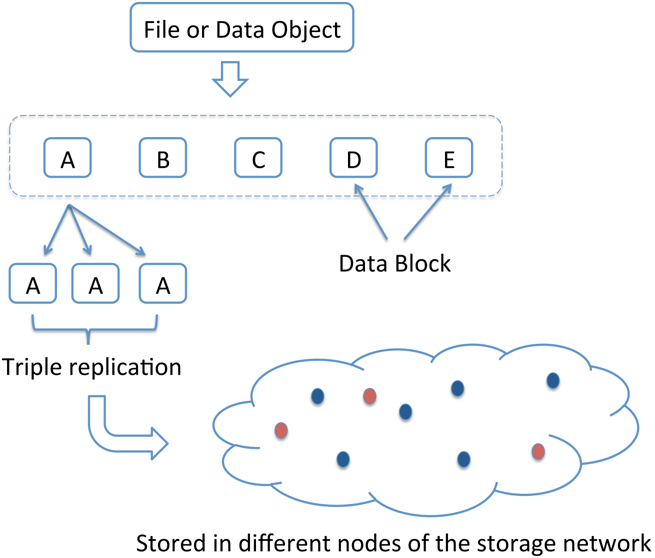
- Erasure Coding
- Node Failures and the Evolution of Coding Theory
- Regenerating Codes
- Locally Recoverable Codes (briefly)
- Codes with Local Regeneration (briefly)
- Codes for Multiple Erasures (briefly)
  - Codes for Data Availability
  - Codes with Sequential Recovery
- The Coupled-Layer MSR Code in Action

Erasure Coding

Fault Tolerance

- Fault tolerance is key to making data loss a very remote possibility
- A time-honored means of achieving fault tolerance is replication..

Triple Replication



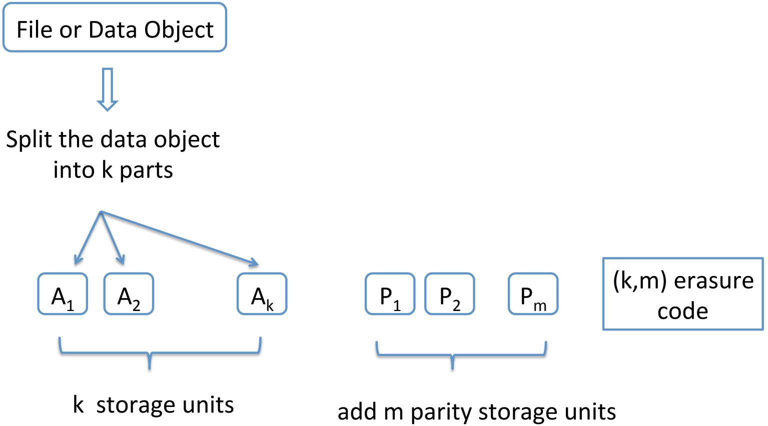
Drawback of Triple Replication

- But triple replication is poor in terms of storage efficiency: just 33%. Are there better ways ?

Drawback of Triple Replication

- But triple replication is poor in terms of storage efficiency: just 33%. Are there better ways ?
- A well-known alternative is to use Erasure Coding (EC)

Erasure Coding of Data



Two Key Performance Measures

1. Storage efficiency

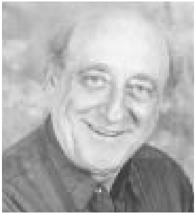
$$\frac{k}{k+m}$$

2. fault tolerance

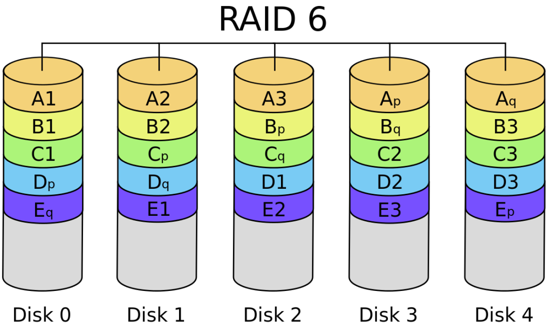
   - at most m storage units

3. Codes with maximum possible fault tolerance $\Rightarrow$ MDS codes

4. Reed-Solomon codes - a prime example



An Example MDS Code - The RAID 6 Code



Source: `https://upload.wikimedia.org/wikipedia/commons/thumb/7/70/RAID_6.svg/1280px-RAID_6.svg.png`

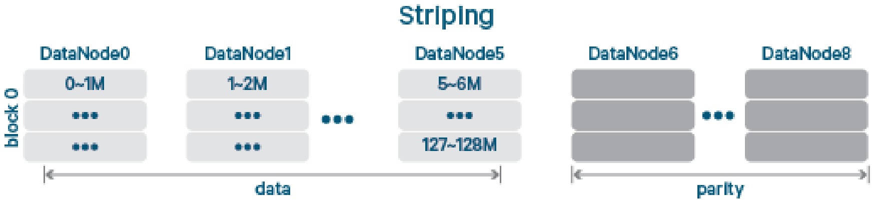
Other RS Codes in Practice

Storage Systems	Reed-Solomon codes
Linux RAID-6	RS(10,8)
Google File System II (Colossus)	RS(9,6)
Quantcast File System	RS(9,6)
Intel & Cloudera' HDFS-EC	RS(9,6)
Yahoo Cloud Object Store	RS(11,8)
Backblaze's online backup	RS(20,17)
Facebook's f4 BLOB storage system	RS(14,10)
Baidu's Atlas Cloud Storage	RS(12, 8)

H. Dau et al, "Repairing Reed-Solomon Codes with Single and Multiple Erasures," ITA, 2017, San Diego.

Evolution of HDFS to Incorporate EC ⇒ HDFS-EC

1. Typically, EC reduces the storage cost by 50% compared with 3x replication
2. Motivated by this, Cloudera and Intel initiated the HDFS-EC project
3. Targeted for release in Hadoop 3.0.
4. Employs a striped layout:



5. Possibility of incorporating more sophisticated EC schemes !

Zhe Zhang, Andrew Wang, Kai Zheng, Uma Maheswara G., and Vinayakumar, "Introduction to HDFS Erasure Coding in Apache Hadoop," September 23, 2015.

Node Failures and the Evolution of Coding Theory

Node Failures

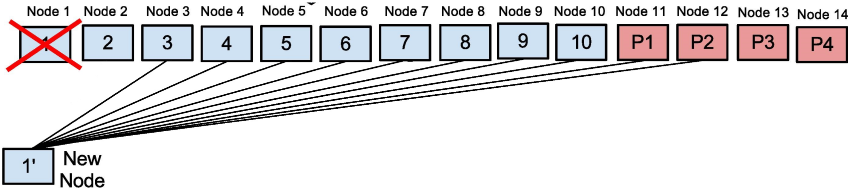
An important consideration is how efficiently the EC can handle node failures as such failures are commonplace:

Figure 1: Number of failed nodes over a single month period in a 3000 node production cluster of Facebook.

M. Asteris, D. Papailiopoulous, A. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data, " PVLDB, 2013.

RS Codes and Node Failures

Under the conventional approach, RS codes are inefficient in two respects at node repair:



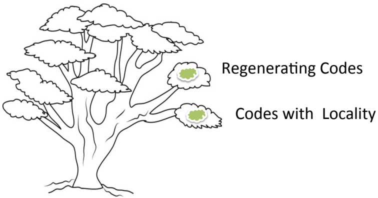
In the example Facebook $[10, 4]$ RS code,

1. the amount of data download (repair BW) equals 10 times the amount stored within the failed node
2. Also, 10 storage units need to be contacted for repair

there is room for improvement...

Coding Theory Responds

1. **Regenerating codes**
   - minimize the amount of data download (repair bandwidth) needed for node repair
2. **Locally recoverable codes**
   - minimize the number of helper nodes contacted for node repair, but also reduce repair bandwidth
3. **Novel and efficient approaches to RS repair** a more recent development



- A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," *IEEE Trans. Inform. Th.*, Sep. 2010.
- P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the Locality of Codeword Symbols," *IEEE Trans. Inf. Theory*, Nov. 2012.
- V. Guruswami, M. Wootters, "Repairing Reed-Solomon Codes," arXiv:1509.04764 [cs.IT] .

Some Comments

Regenerating Codes

1. Minimum Storage Regenerating (MSR) Codes are MDS codes
2. Regenerating codes are vector codes, each code symbol is a vector of code ℓ symbols
   - ℓ is called the sub-packetization level

Locally Recoverable Codes

1. Locally recoverable codes yield on storage efficiency for ease of node repair

Fresh approach to RS repair

1. regard RS codes as vector codes
2. minimize repair bandwidth under a constraint on sub-packetization level ℓ

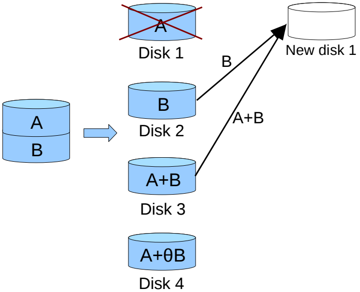
Regenerating Codes

- Focus here on the subclass of Minimum Storage Regenerating (MSR) Codes

Raid Code - Not Very Good at Handling Node Failure..

The conventional approach:

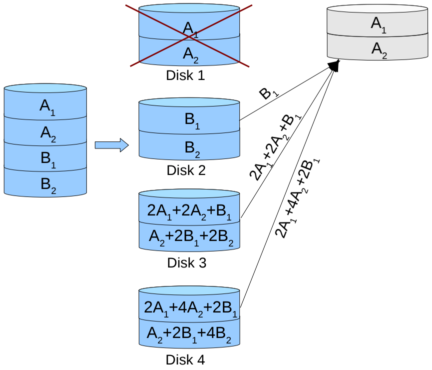
- Connect to any 2 nodes,
- Reconstruct *A and B*,
- Extract *A*



But downloading 2 units of data to revive a node that stores 1 units of data is clearly, wasteful of network bandwidth..

Replacing the RAID 6 Code with a Regenerating Code

- Here, each node now stores two "half-symbols"
- We download 3 half-symbols as opposed to 2 full-symbols
  - Can recover any of $\{A_1, A_2, B_1\}$



Evolution of MSR Codes

Code	Explicit	SE	SPL	OA	HN
Product-Matrix	Yes	Low	Low	No	d
Hadamard & Butterfly*	Yes	High	High	No	all
Zig-Zag Code	No	High	High	Yes	all
Sasidharan et al (1)	No	High	Low	Yes	all
Ye-Barg (1)	Yes	High	High	Yes	all
Ye-Barg (2)	Yes	High	Low	Yes	all
Sasidharan et al (2)	Yes	High	Low	No	d

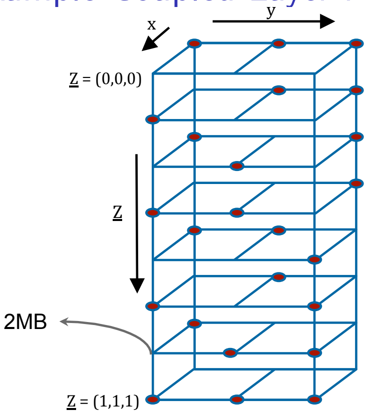
* $\Rightarrow$ limited to 2 parity nodes

- SE $\Rightarrow$ storage efficiency
- SPL $\Rightarrow$ sub-packetization level
- OA $\Rightarrow$ optimal access (number of symbols accessed for repair)
- HN $\Rightarrow$ number of helper nodes needed

References (MSR Codes with High Storage Efficiency)

1. K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal Exact-Regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-Matrix Construction," IEEE Trans. Inf. Theory, Aug. 2011.
2. D. S. Papailiopoulos, A. G. Dimakis, and V. Cadambe, "Repair optimal erasure codes through Hadamard designs," IEEE Trans. Inf. Theory, May 2013.
3. E. En Gad, R. Mateescu, F. Blagojevic, C. Guyot, and Z. Bandic, " Repair-Optimal MDS Array Codes Over GF (2)," in Proceedings IEEE International Symposium on Information Theory (ISIT), 2013.
4. Zhiying Wang, Itzhak Tamo, Jehoshua Bruck, "Optimal Rebuilding of Multiple Erasures in MDS Codes, " IEEE Trans. Information Theory, Feb. 2017.
5. B. Sasidharan, G. K. Agarwal, and P. V. Kumar, "A high-rate MSR code with polynomial sub-packetization level, " in IEEE International Symposium on Information Theory, ISIT 2015.
6. S. Goparaju, A. Fazeli, and A. Vardy, "Minimum storage regenerating codes for all parameters," IEEE Information Theory Transactions, April 2017.
7. M. Ye and A. Barg, "Explicit constructions of high-rate MDS array codes with optimal repair bandwidth, " IEEE Information Theory Transactions, April 2017.
8. M. Ye and A. Barg, "Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization, " CoRR, vol. abs/1605.08630, 2016.
9. B Sasidharan, M Vajha, PV Kumar, "An Explicit, Coupled-Layer Construction of a High-Rate MSR Code with Low Sub-Packetization Level, Small Field Size and $d < (n-1)$, " CoRR, vol. abs/1701.07447, 2017, to be presented at ISIT 2017.

Example Coupled-Layer MSR Code

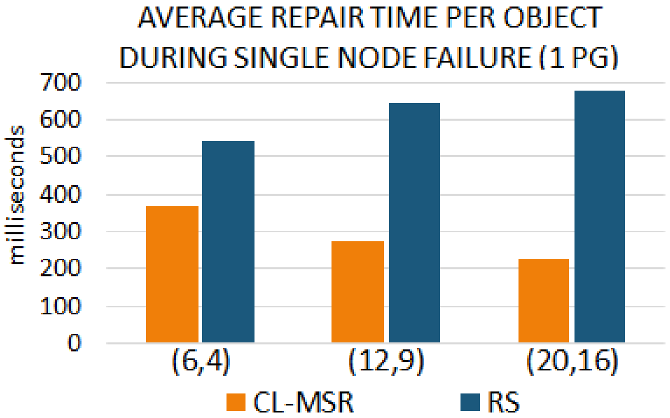


- Our coupled-layer perspective on the Ye-Barg construction (2)
- a $(4, 2)$ MSR code
- 6 nodes, sub-packetization level is $\ell = 8$
- $6 \times 8 = 48$ points
- in the example to follow, each point stores 2MB

---

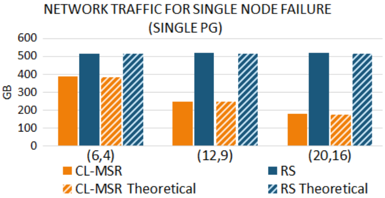
1. M. Ye, and A. Barg, "Explicit constructions of optimal- access MDS codes with nearly optimal sub-packetization, " May 2016.
2. B. Sasidharan, M. Vajha, and PVK. "An Explicit, Coupled-Layer Construction of a High-Rate MSR Code with Low Sub-Packetization Level, Small Field Size and $d < (n-1)$, " to be presented at ISIT 2017.

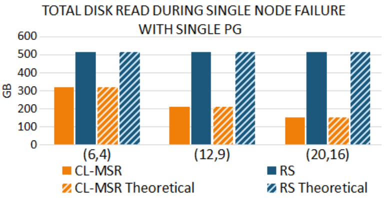
Performance of the Coupled-Layer MSR Code



1. A comparison of actual repair time is shown. In the figure,
   - the $(6,4)$ code is in our present notation a $(4,2)$ code
   - the $(12,9)$ code is in our present notation a $(9,3)$ code
   - the $(20,16)$ code is in our present notation a $(16,4)$ code

Performance of the Coupled-Layer MSR Code

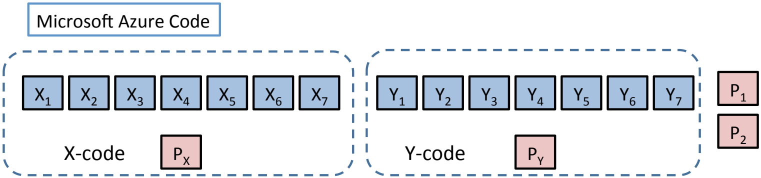




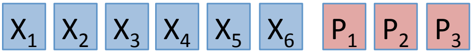
- Similar gains in network bandwidth and disk read
- Thus a larger sub-packetization level is not necessarily a problem for implementation

Locally Recoverable Codes

Windows Azure Storage Coding Solution

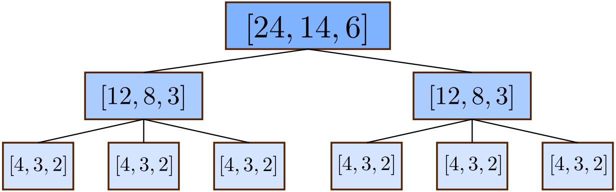


Comparison: In terms of reliability and number of helper nodes contacted for node repair, the two codes are comparable. The overheads however are quite different, 1.29 for the Azure code versus 1.5 for the RS code. This difference has reportedly saved Microsoft millions of dollars.



Huang, Simitci, Xu, Ogus, Calder, Gopalan, Li, Yekhanin, "Erasure Coding in Windows Azure Storage," USENIX, Boston, MA, 2012.

Codes with Hierarchical Locality



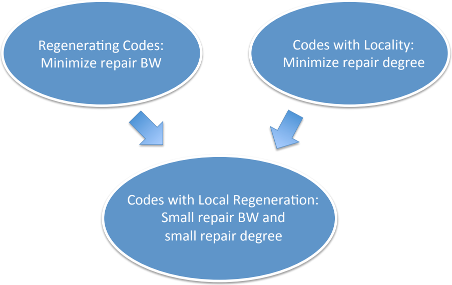
- $[4, 3, 2]$ code $\Rightarrow$ (3,1) code
- $[12, 8, 3]$ code $\Rightarrow$ (8,4) code
- $[24, 14, 6]$ code $\Rightarrow$ (14,10) code

- Codes with hierarchical locality do exactly that by calling for help from an intermediate layer of codes when the local code fails.
- These codes may be regarded as the "middle codes".

B. Sasidharan, G. K.Agarwal, PVK, "Codes With Hierarchical Locality," arXiv:1501.06683 [cs.IT].

Codes with Local Regeneration

Codes with Local Regeneration



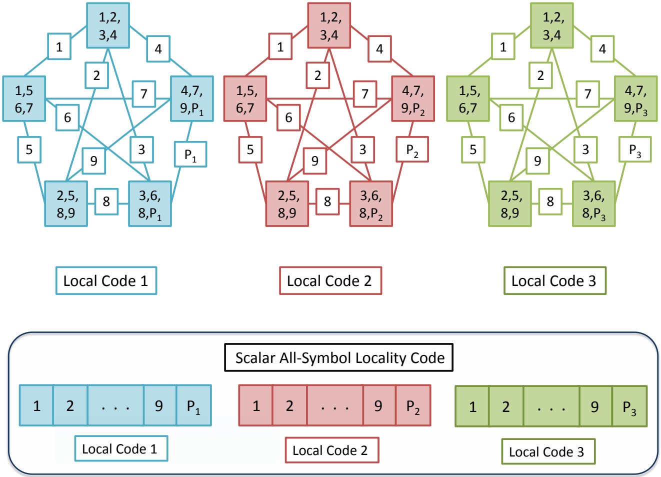
- A single code that has both locality and regeneration properties
- and inherent double replication of data

---

1. G. M. Kamath, N. Prakash, V. Lalitha, PVK, 'Codes With Local Regeneration and Erasure Correction," T-IT, Aug. 2014 .

An Example Code with Local Regeneration

The construction makes can make use of an all-symbol local scalar code and is also optimal:



Codes with Availability (Recovery from Simultaneous Multiple Erasures)

Recovery in Parallel

c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{31}	$\cancel{c_{32}}$	$\cancel{c_{33}}$	c_{34}	c_{35}
c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

- Last column is a parity check on entries to the left in the same row
- Last row is a parity check on entries above in the same column
- Can recover *locally* from 2 erasures in parallel

Codes with Sequential Recovery (Recovery from Simultaneous Multiple Erasures)

Sequential Recovery

c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

- Same code as before
- Can recover locally from 3 erasures *in a sequential manner*
- Sequential recovery enables codes with larger storage efficiency

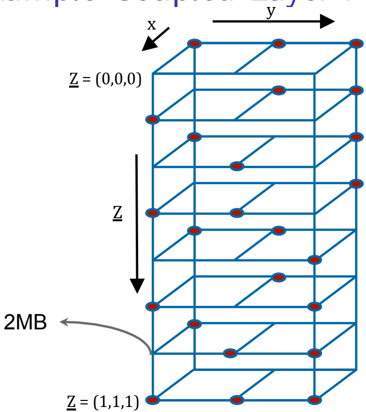
References - Codes for Multiple Erasures

1. A. Wang and Z. Zhang, "Repair locality with multiple erasure tolerance," IEEE Trans. Inf. Theory, Nov. 2014.
2. N. Prakash, V. Lalitha, and P. V. Kumar, "Codes with locality for two erasures," in Proc. IEEE Int. Symp. Inform. Theory (ISIT) 2014.
3. W. Song and C. Yuen, "Binary locally repairable codes - sequential repair for multiple erasures," in Proc. IEEE GLOBECOM, 2016.

Functioning of an Example, Coupled-Layer MSR Code

- Goal: To show that a larger sub-packetization level is not necessarily a problem for implementation

Example Coupled-Layer MSR Code



- Our coupled-layer perspective on the Ye-Barg construction (2)
- a $(4, 2)$ MSR code
- 6 nodes, sub-packetization level is $\ell = 8$
- $6 \times 8 = 48$ points
- in the example to follow, each point stores 2MB

---

1. M. Ye, and A. Barg, "Explicit constructions of optimal- access MDS codes with nearly optimal sub-packetization, " May 2016.
2. B. Sasidharan, M. Vajha, and PVK. "An Explicit, Coupled-Layer Construction of a High-Rate MSR Code with Low Sub-Packetization Level, Small Field Size and $d < (n-1)$, " to be presented at ISIT 2017.

Consider a file of size 64MB

64MB

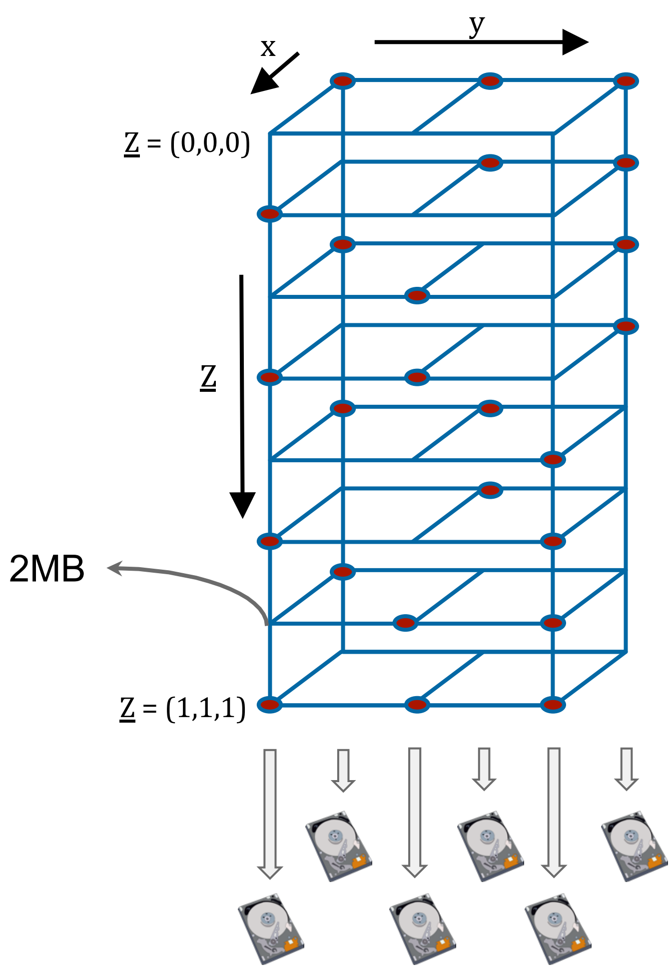
- Will encode via a [k=4, m=2] MSR Code
- Called **the Coupled-Layer MSR Code**

Step 1: Break file into k = 4 data chunks, each of 16MB.

16MB	16MB	16MB	16MB

Data cube representation of CL-MSR Code

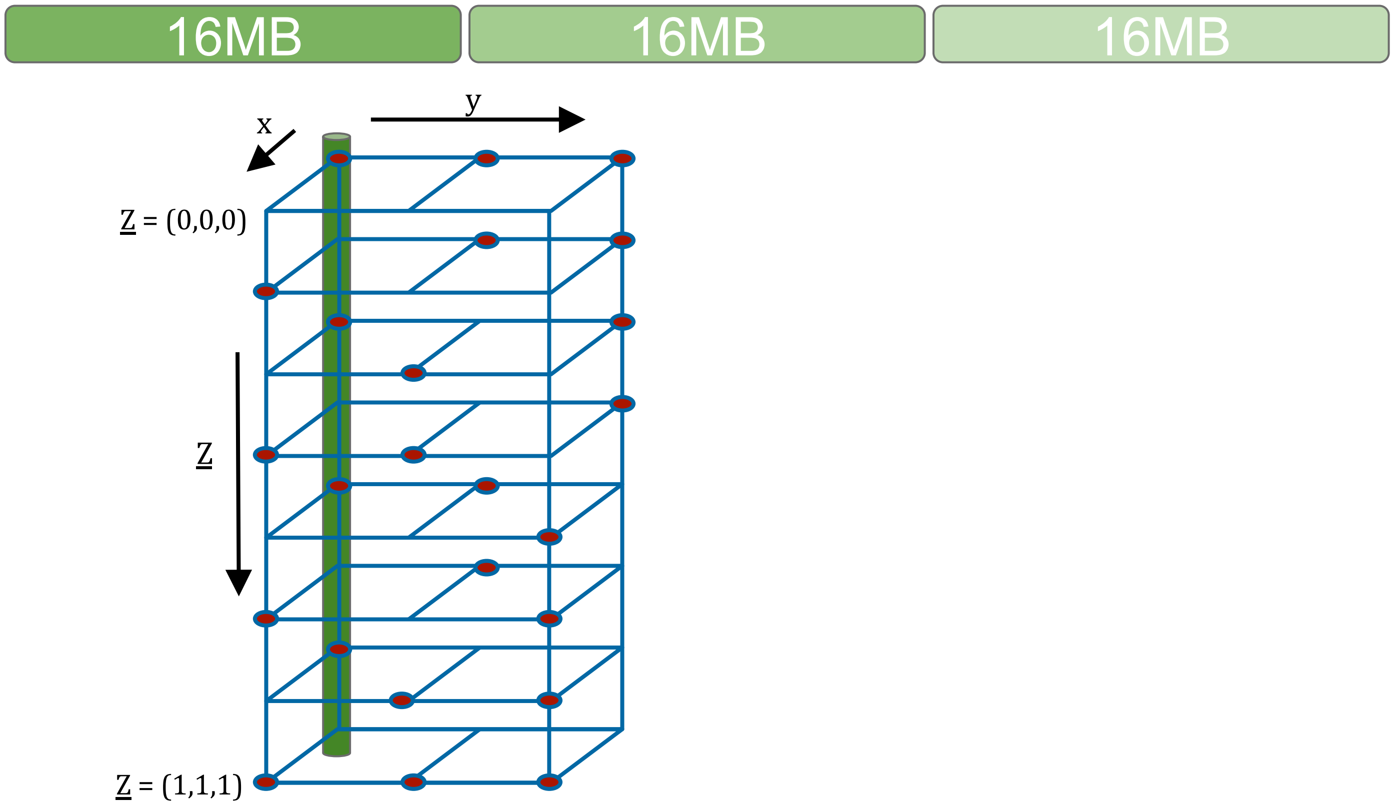
16MB	16MB	16MB	16MB



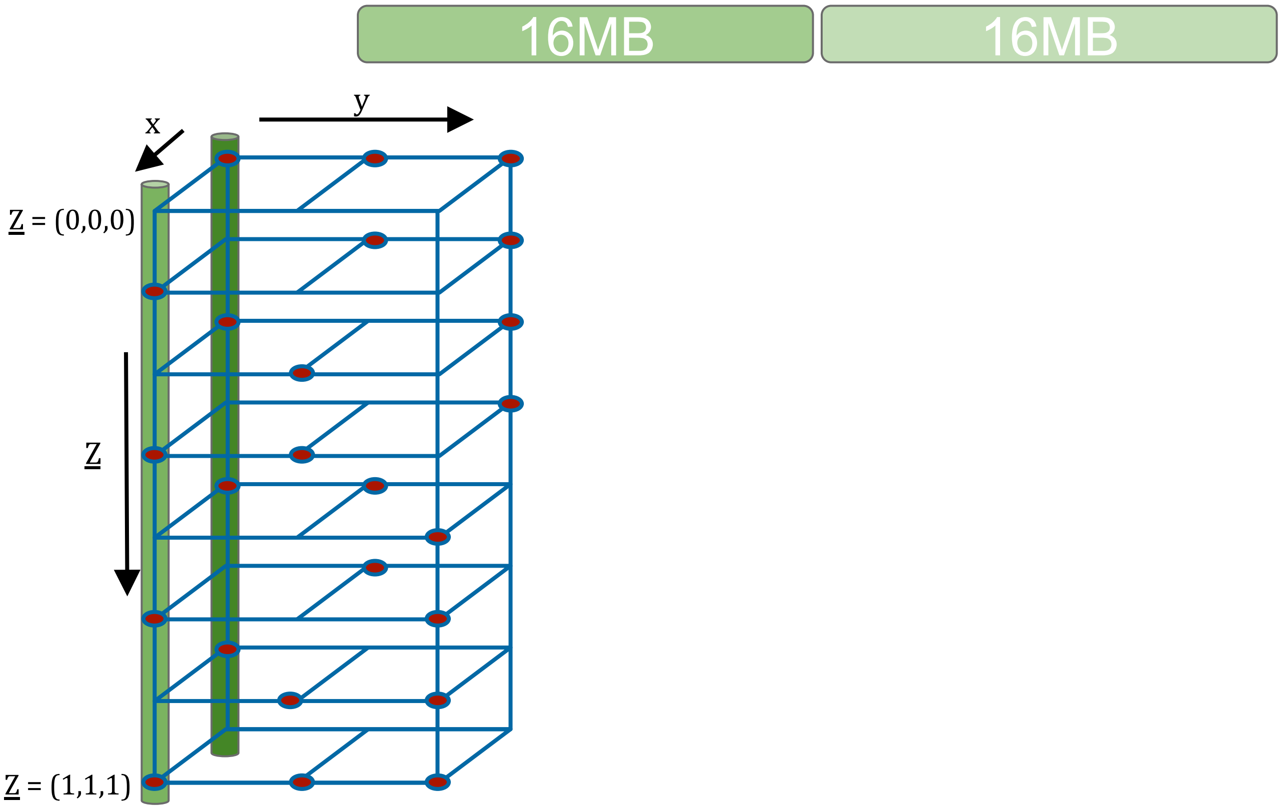
The cube has:

- 6 columns, each associated to a distinct node
- 8 horizontal planes.
- A column has 8 points
- Each point corresponds to 2MB of storage

Place four 16MB chunks in four systematic nodes

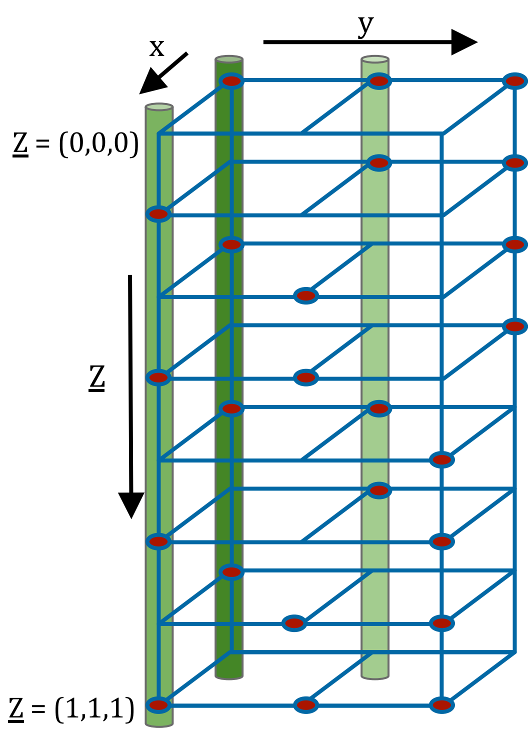


Place four 16MB chunks in four systematic nodes

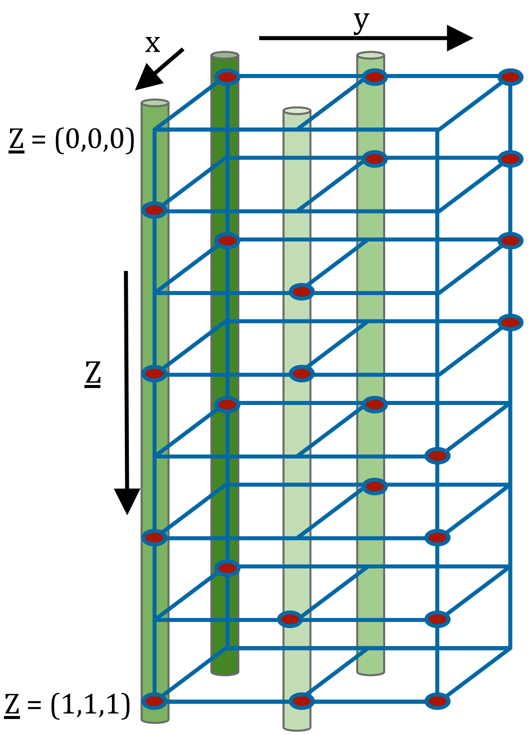


Place four 16MB chunks in four systematic nodes

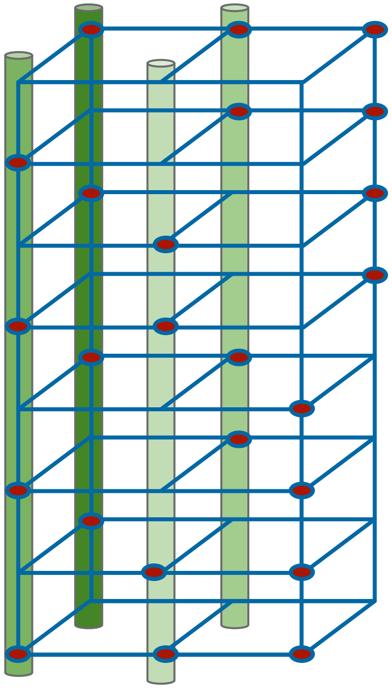
16MB



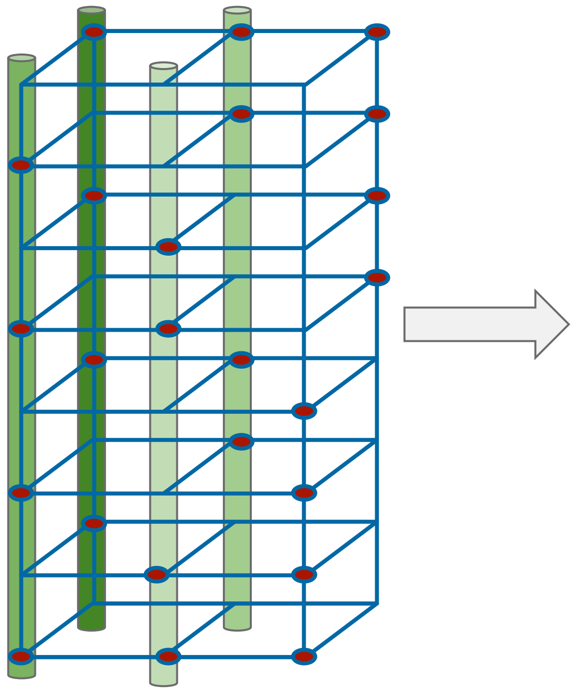
Place four 16MB chunks in four systematic nodes

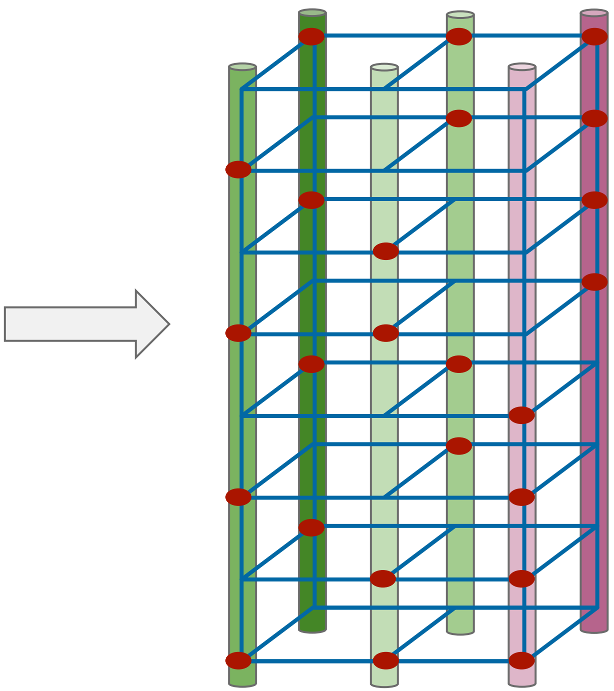


We now have the systematic nodes



We will now compute the parity nodes

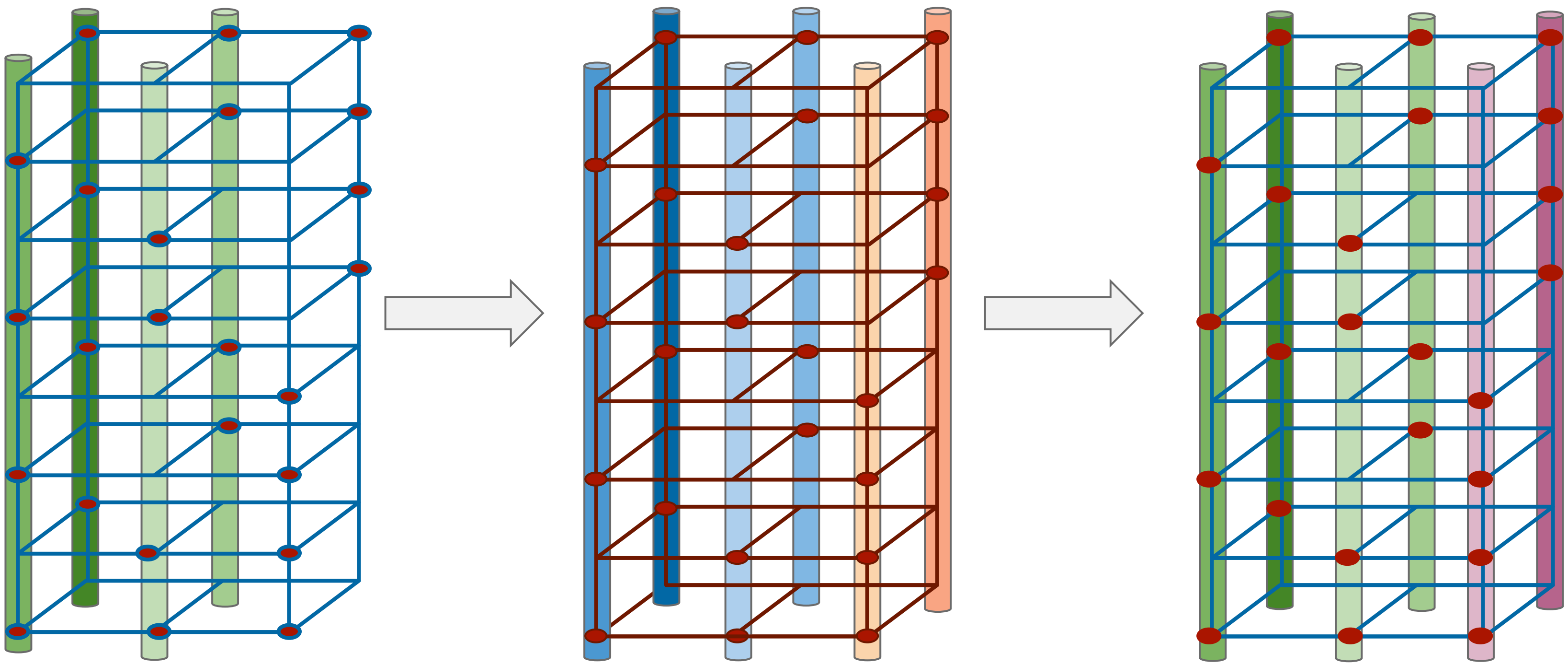




Actual data cube

A

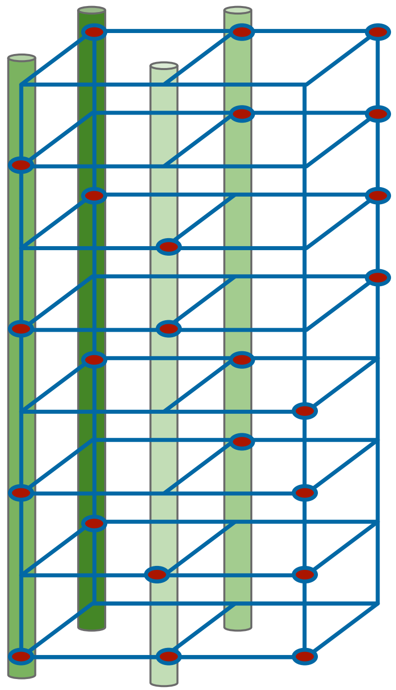
Will get there through an intermediate “Virtual data cube”

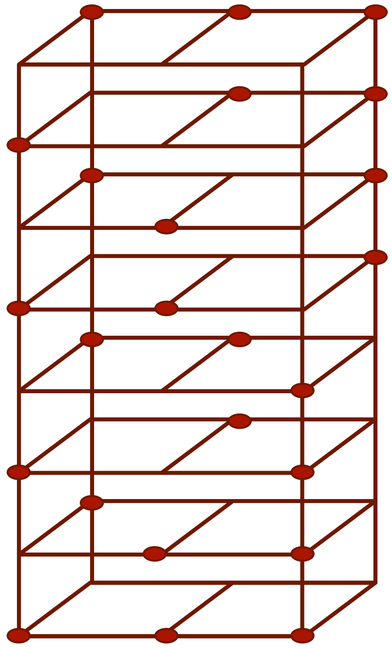


Virtual data cube
B

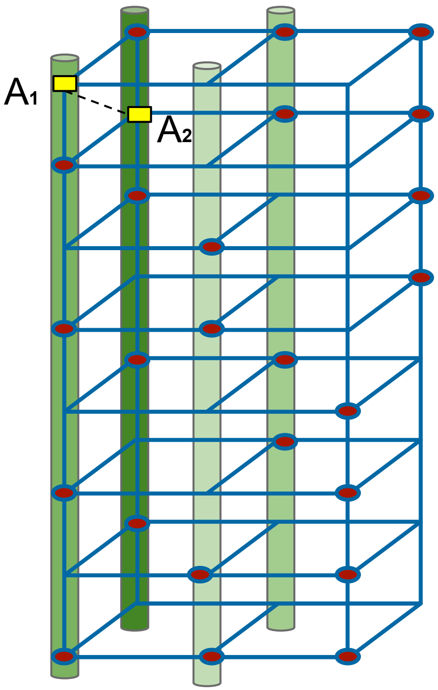
Actual data cube
A

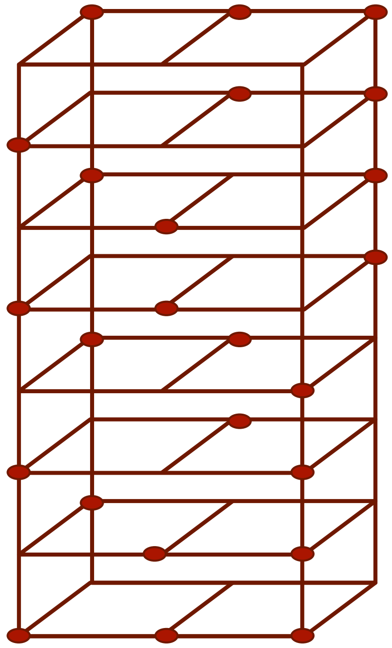
Start filling the virtual data cube on the right as follows



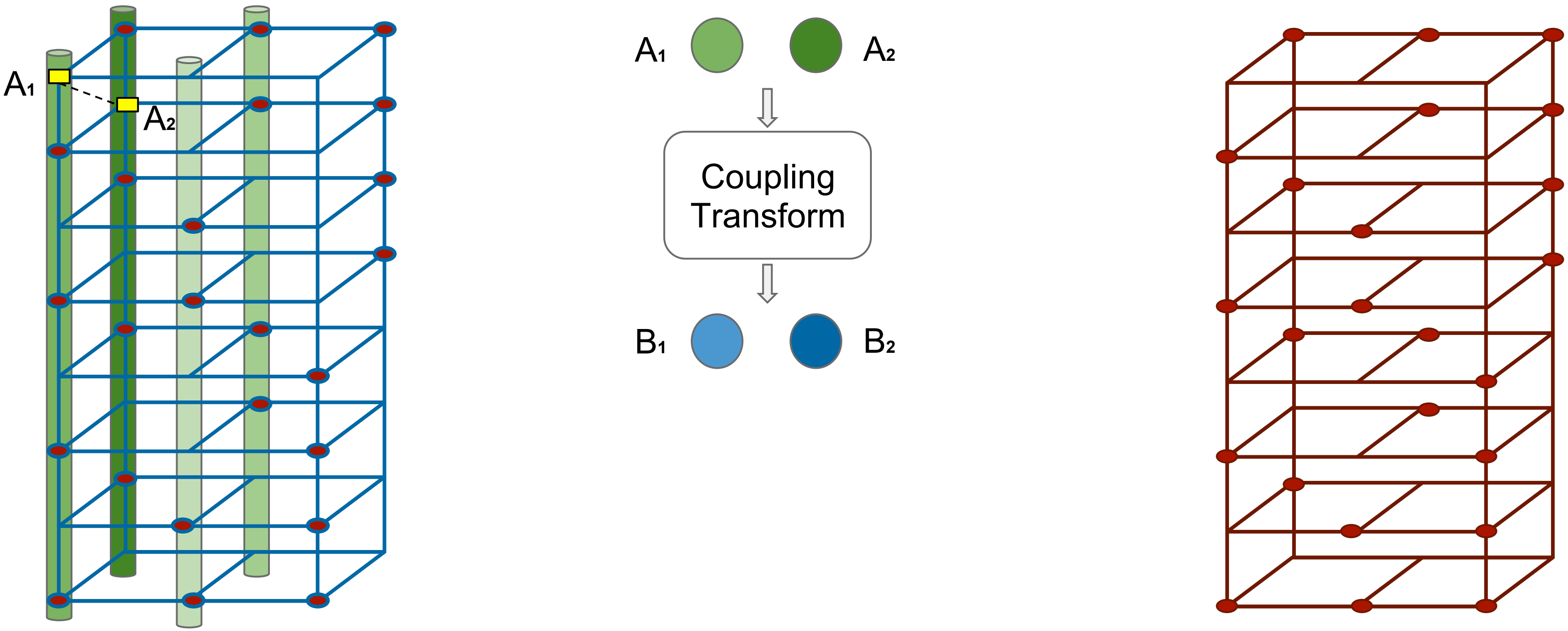


Certain pairs of points in the cube are "coupled"

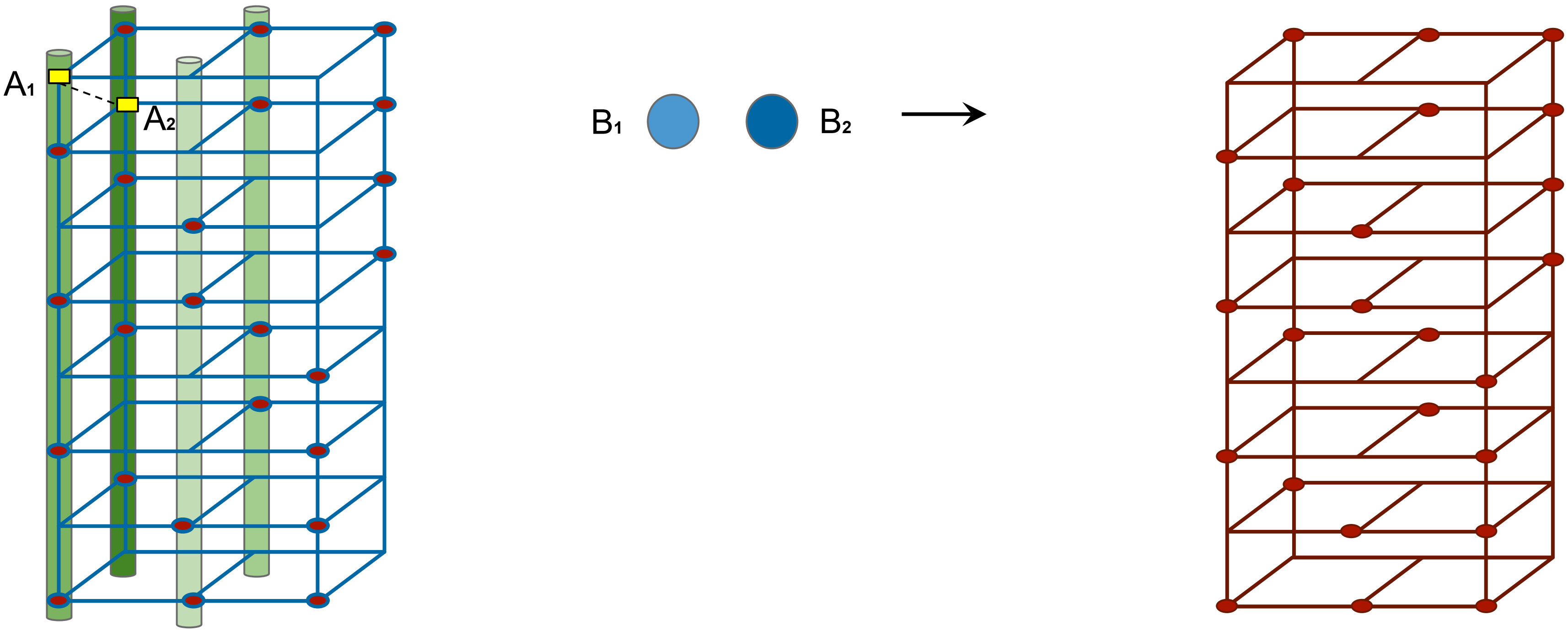




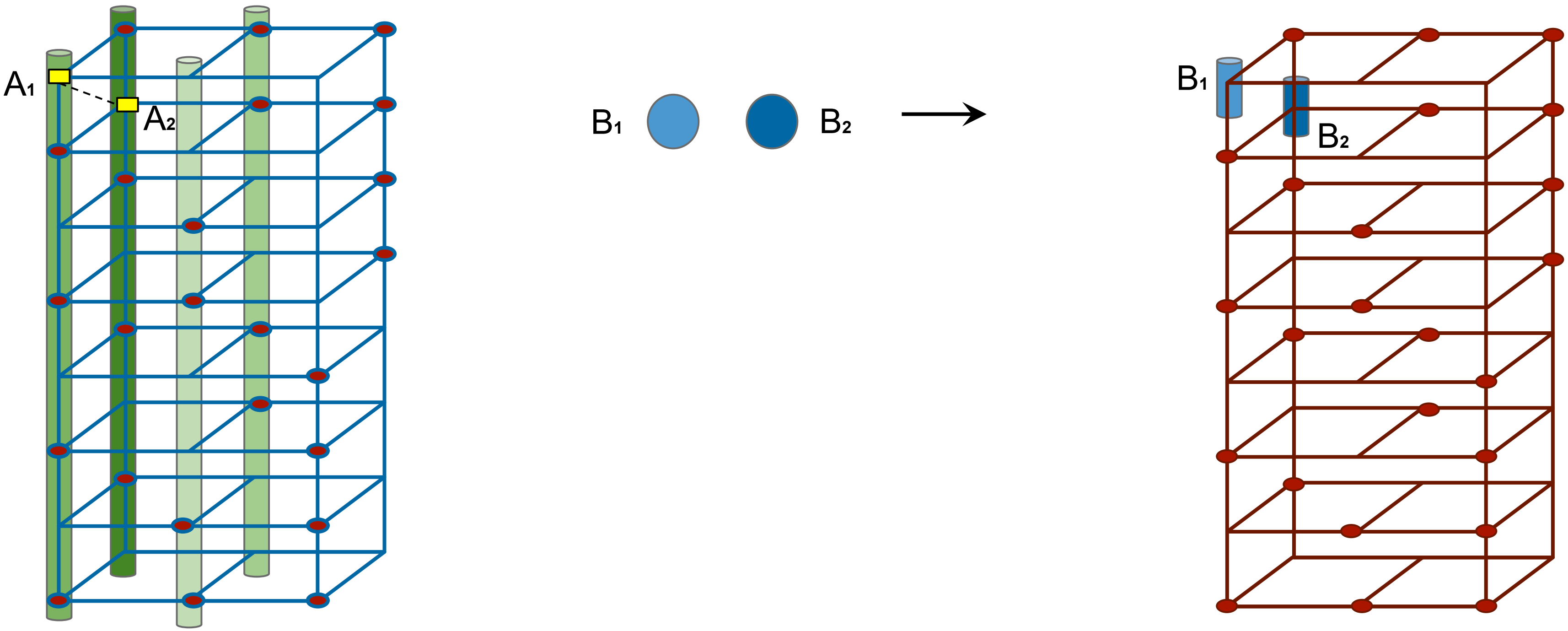
The Coupling Transform is a 2x2 matrix transform



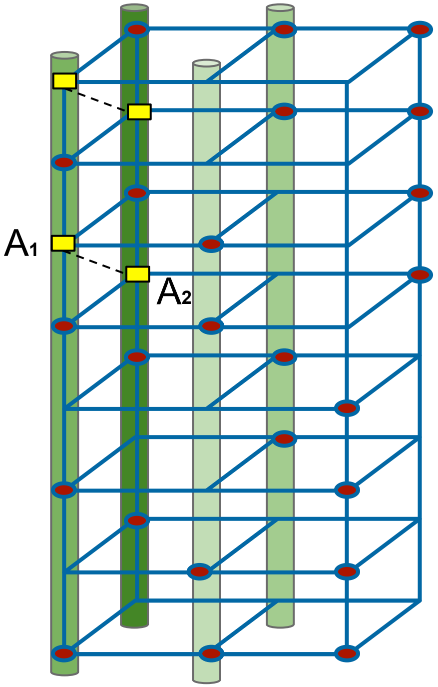
Place the points obtained in the Virtual data cube

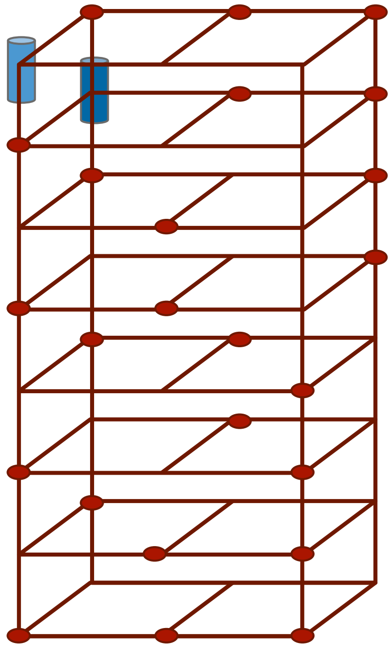


Place the points obtained in the Virtual data cube

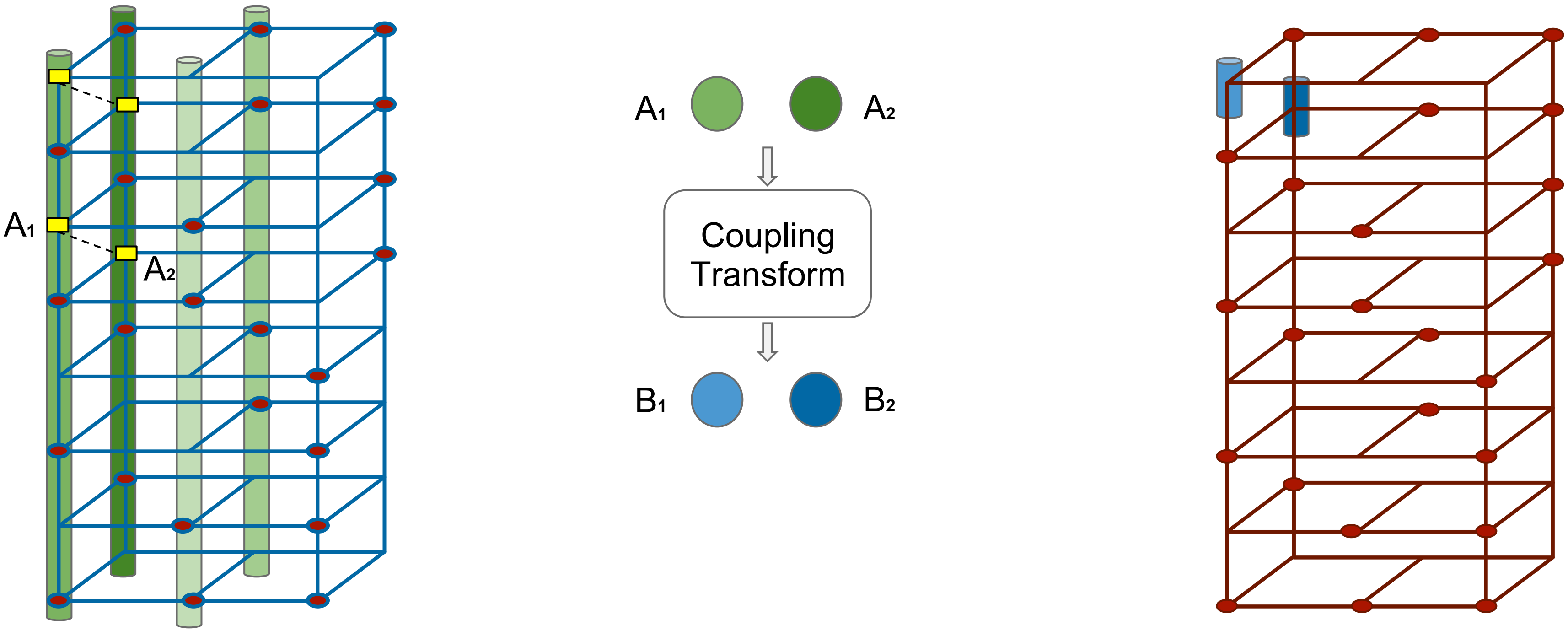


Place the points obtained in the Virtual data cube

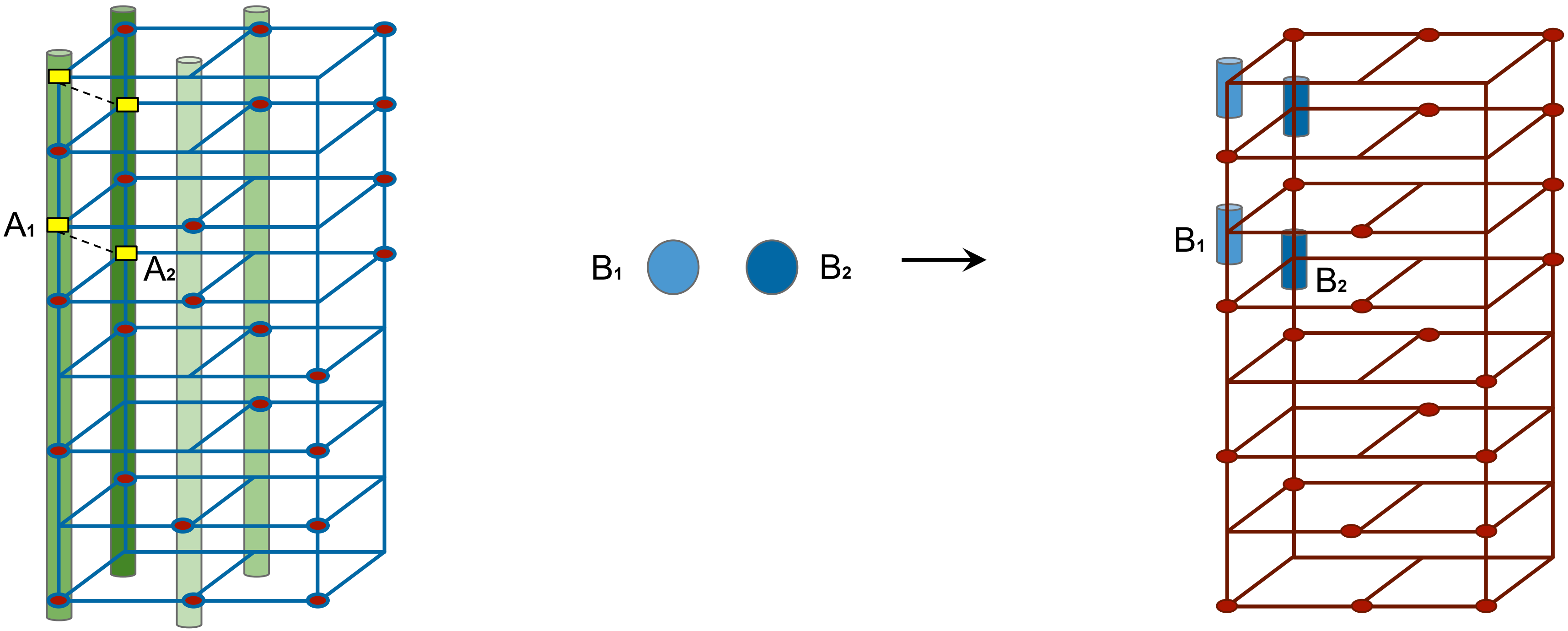




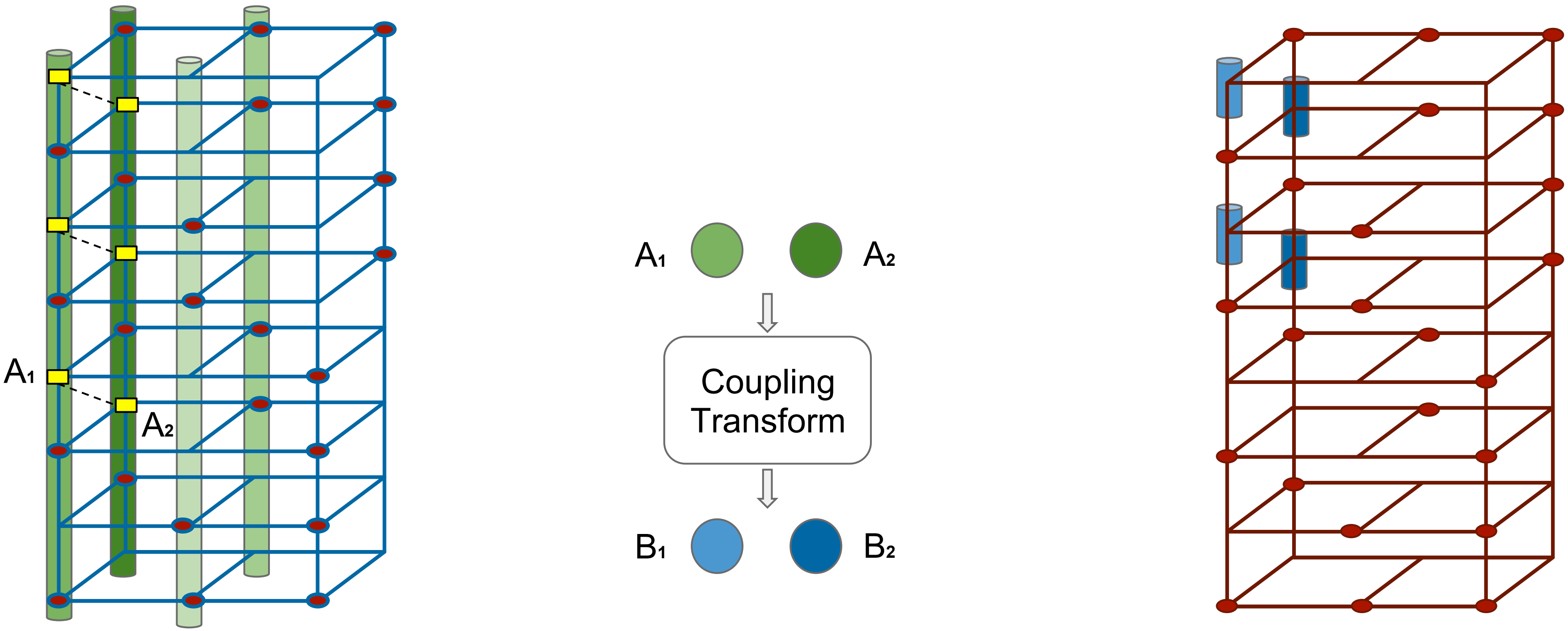
Place the points obtained in the Virtual data cube



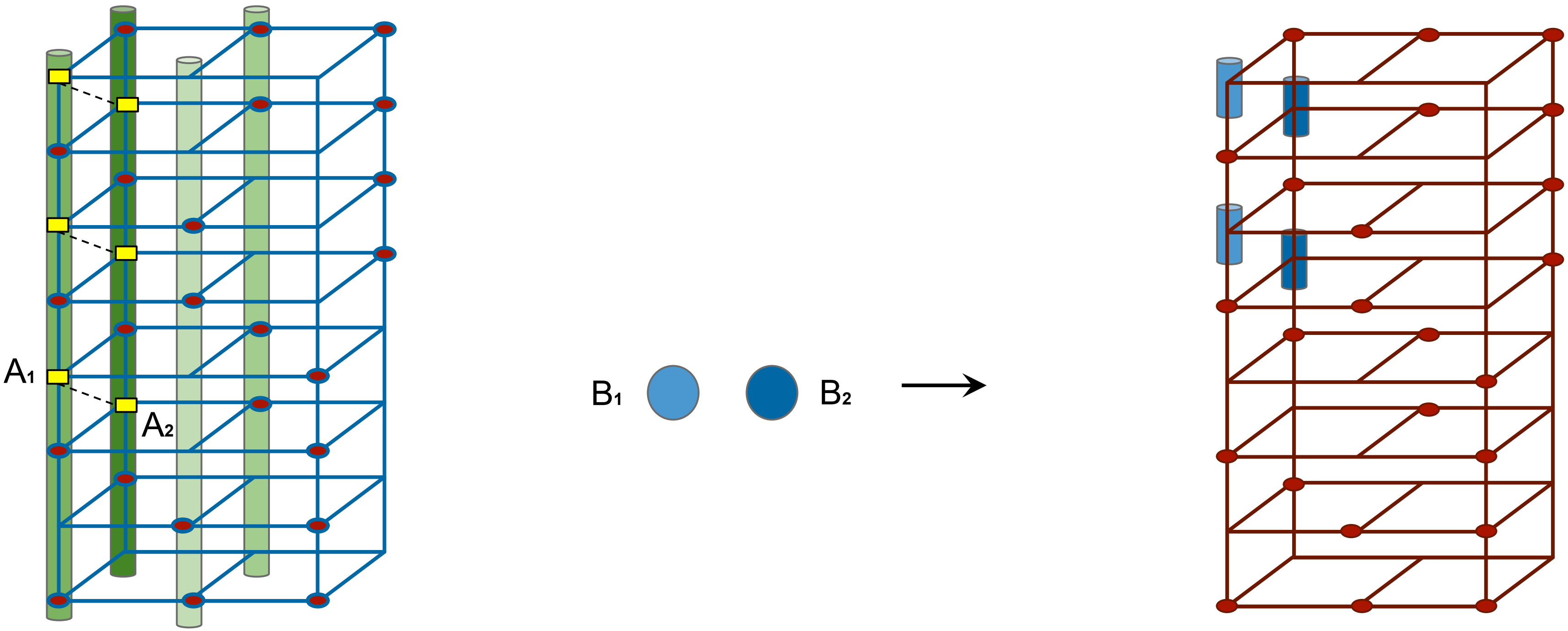
Place the points obtained in the Virtual data cube



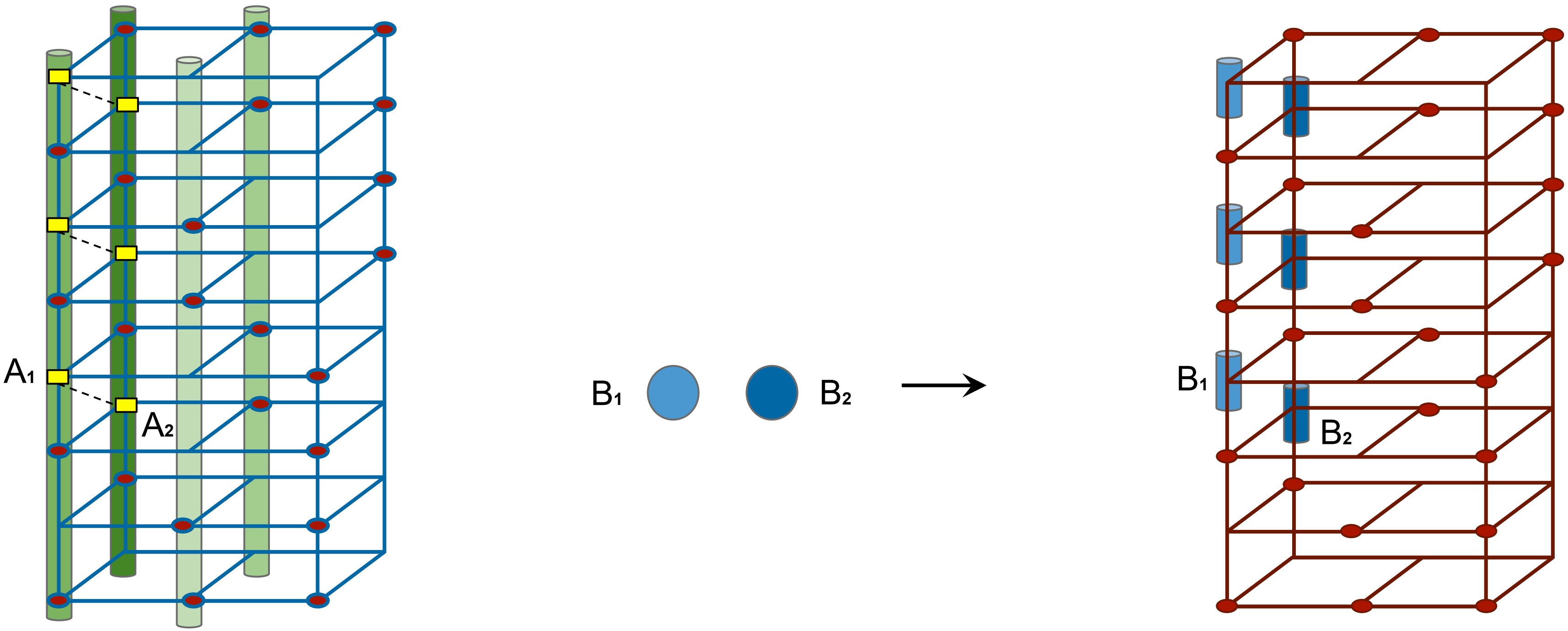
Place the points obtained in the Virtual data cube



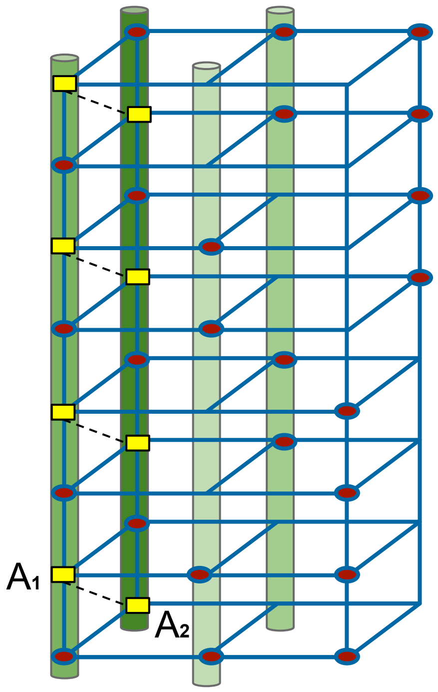
Place the points obtained in the Virtual data cube

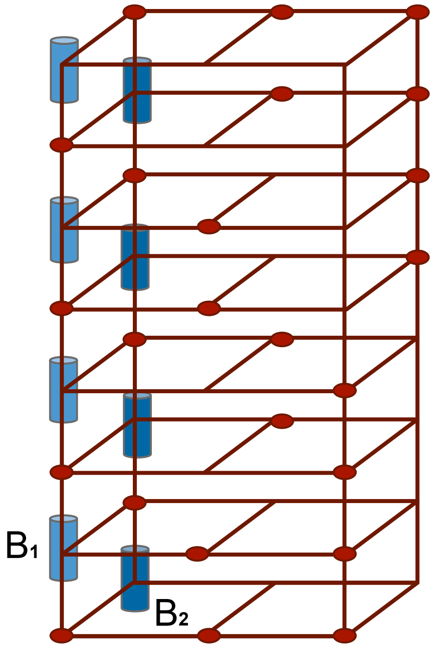


Place the points obtained in the Virtual data cube

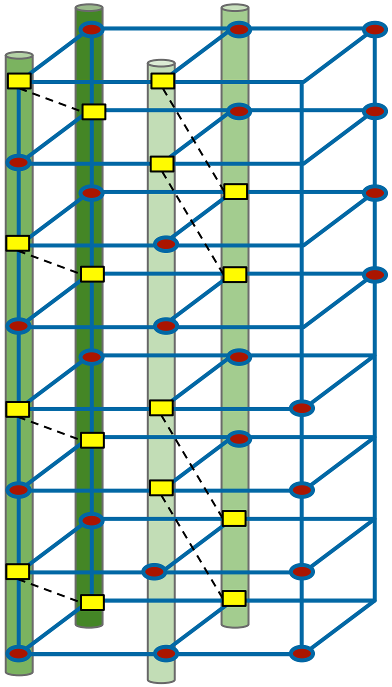


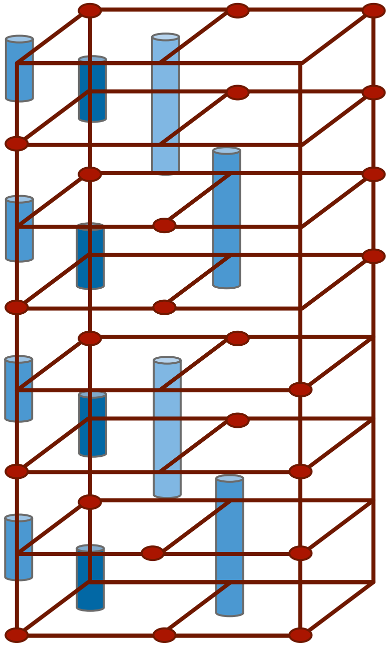
Place the points obtained in the Virtual data cube



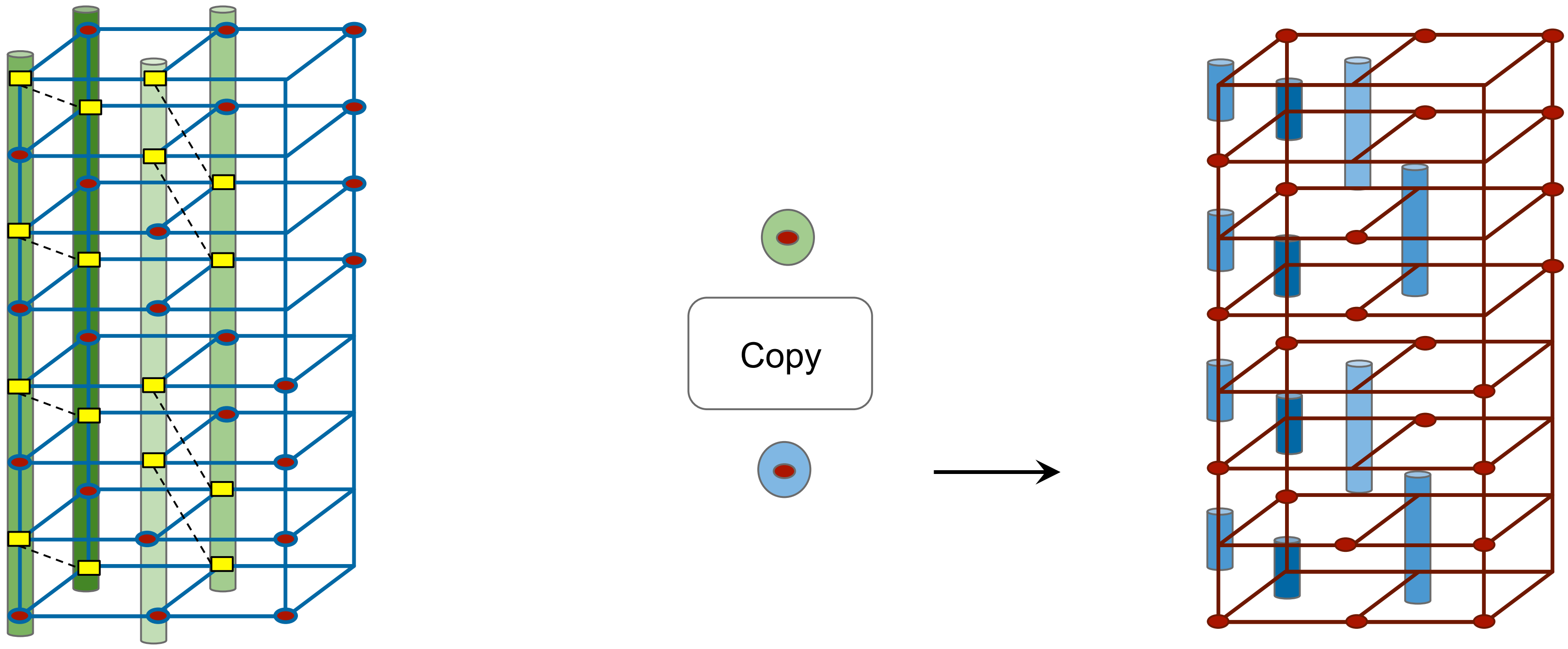


Place the points obtained in the Virtual data cube

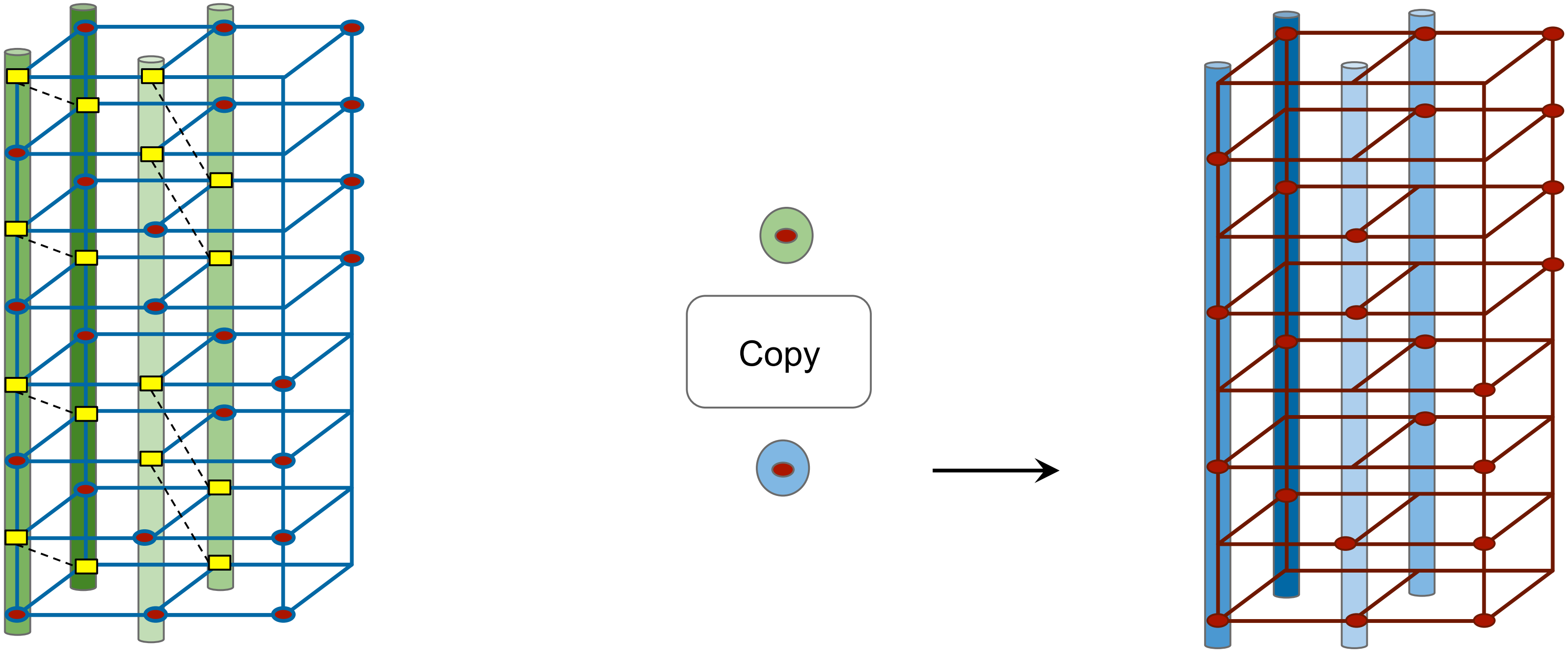




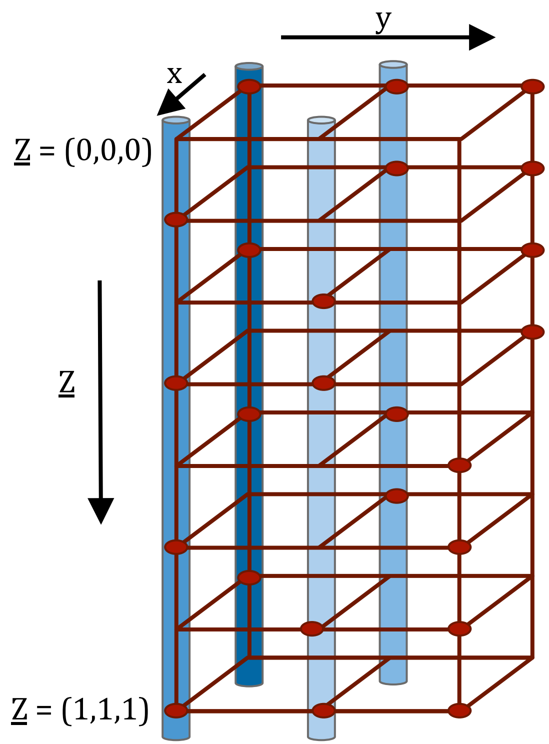
Red dotted points are not paired, they are simply carried over



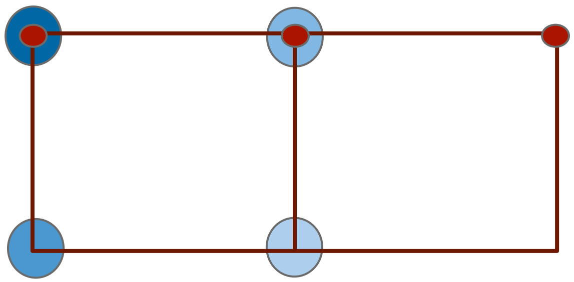
Red dotted points are not paired, they are simply carried over



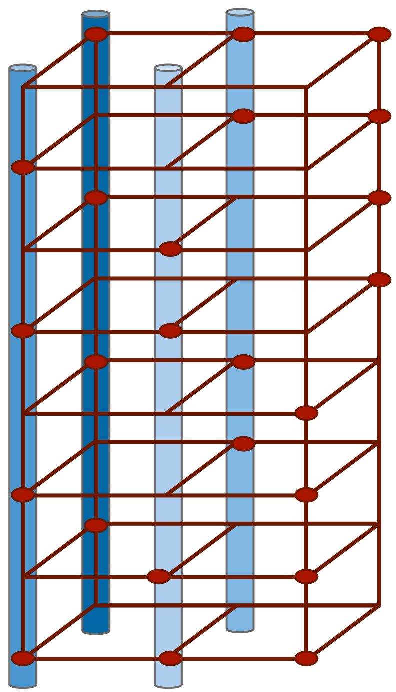
We now have data-part of the Virtual data cube



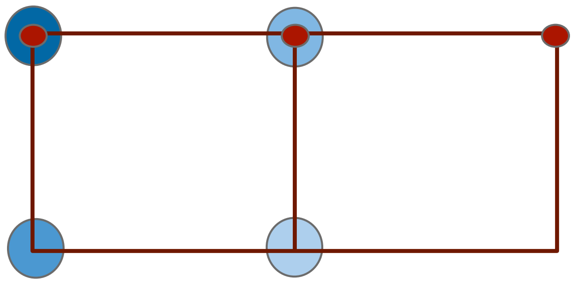
Each plane is Reed-Solomon coded to obtain parity points



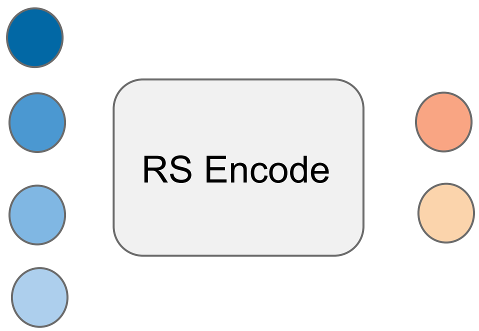
$\underline{Z} = (0,0,0)$

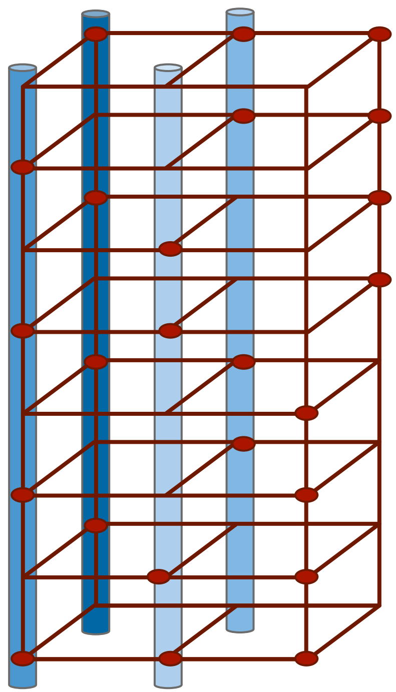


Each plane is Reed-Solomon coded to obtain parity points

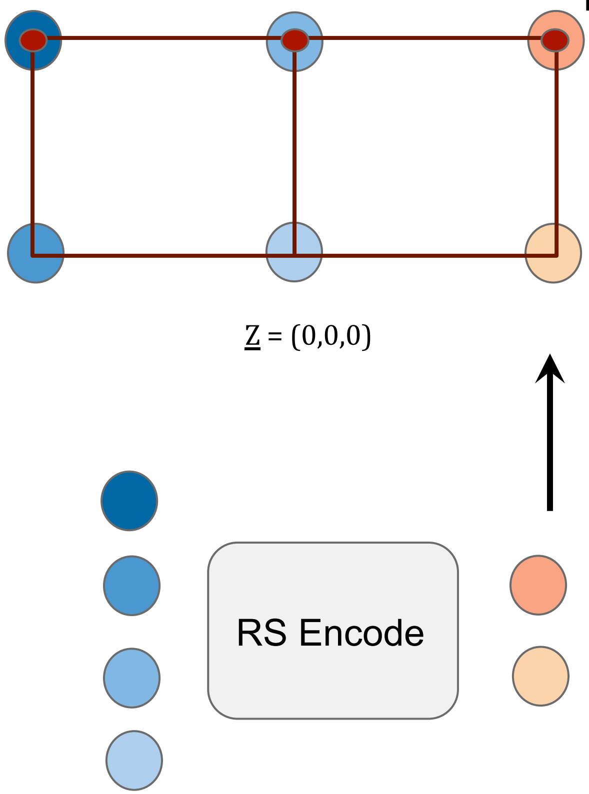


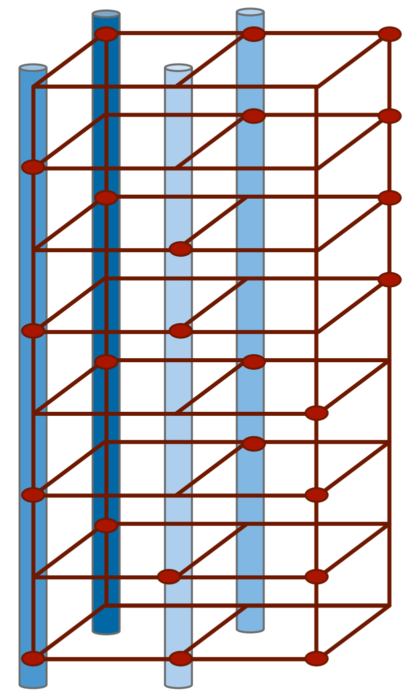
$\underline{Z} = (0,0,0)$



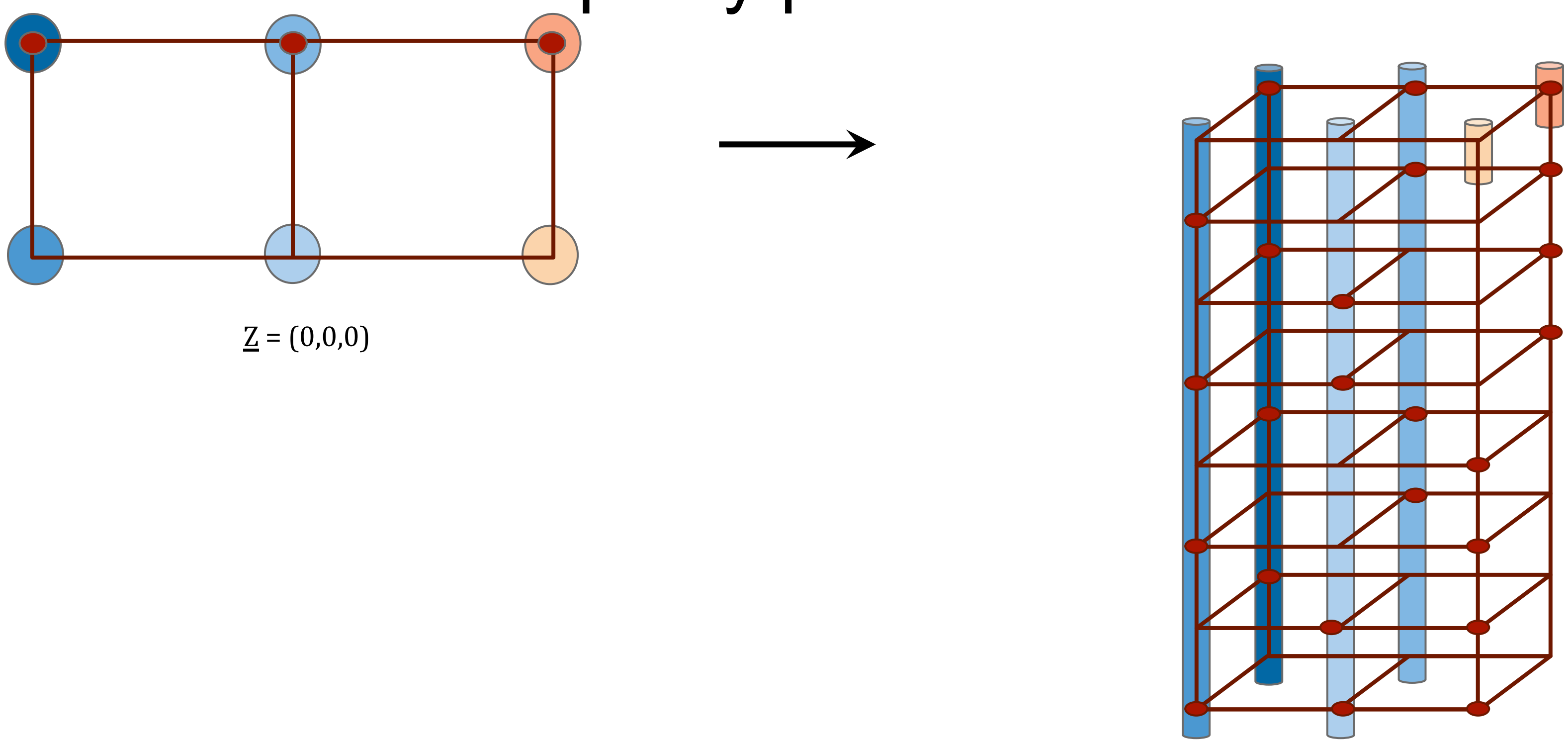


Each plane is Reed-Solomon coded to obtain parity points

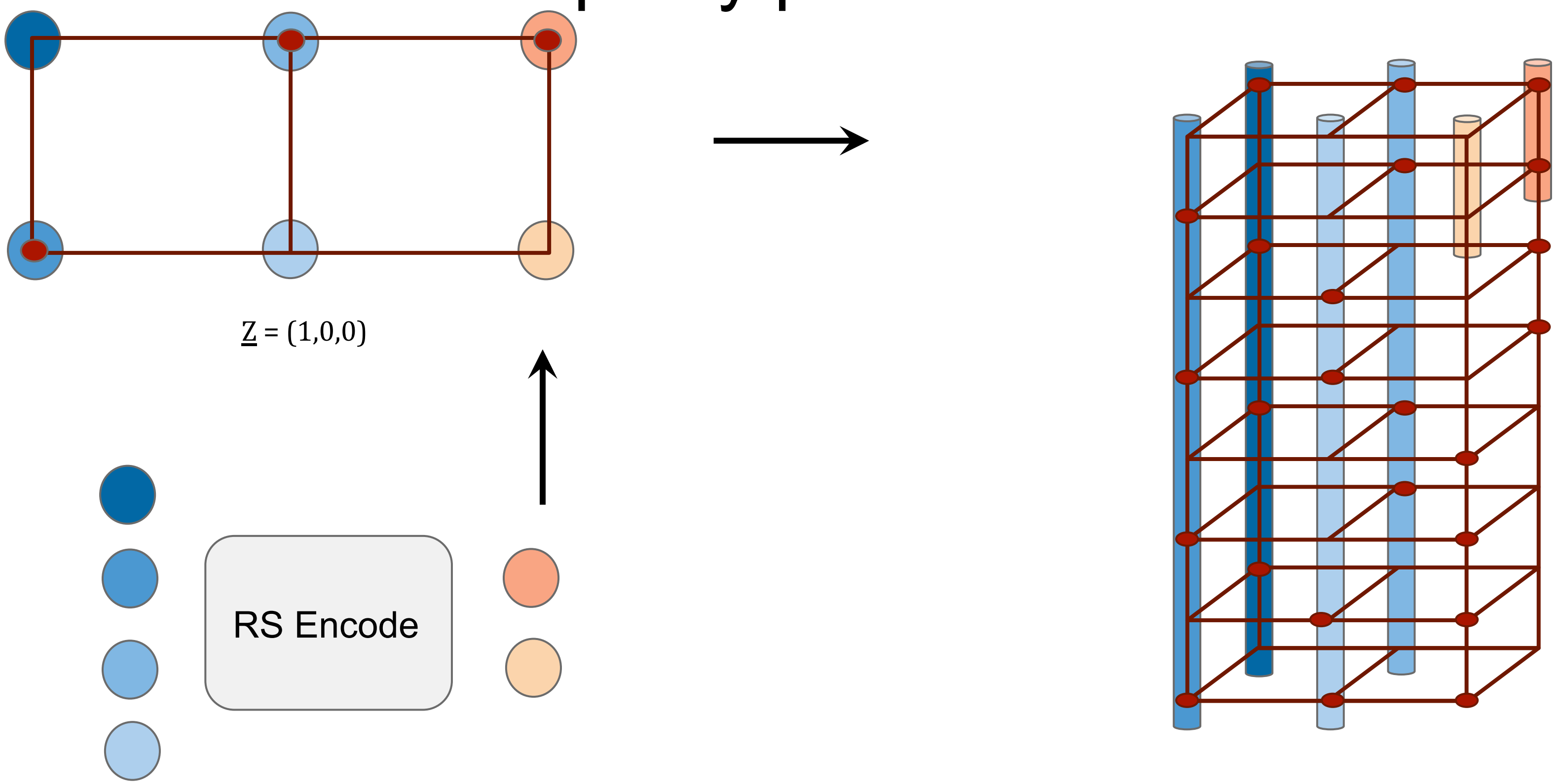




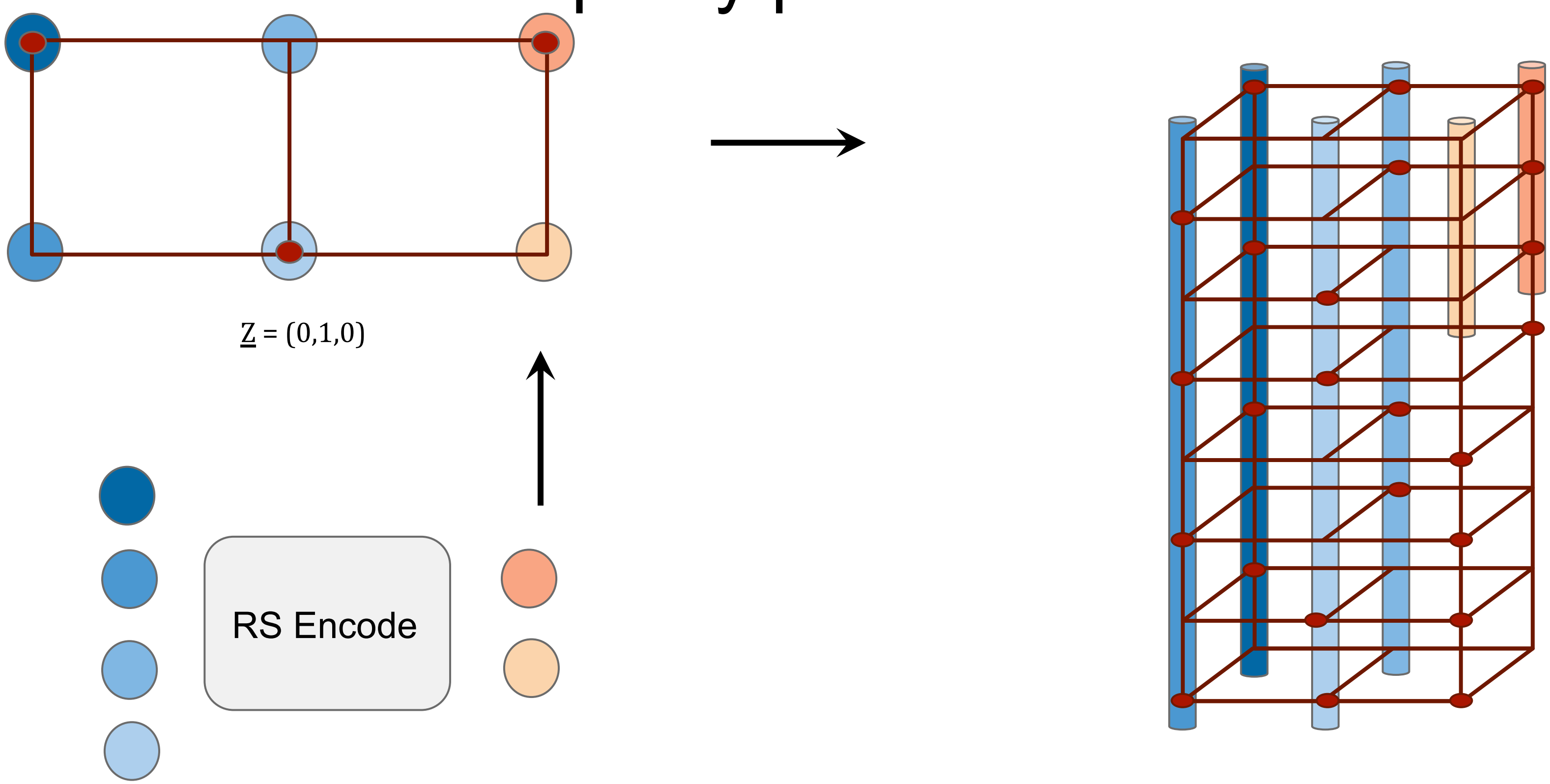
Each plane is Reed-Solomon coded to obtain parity points



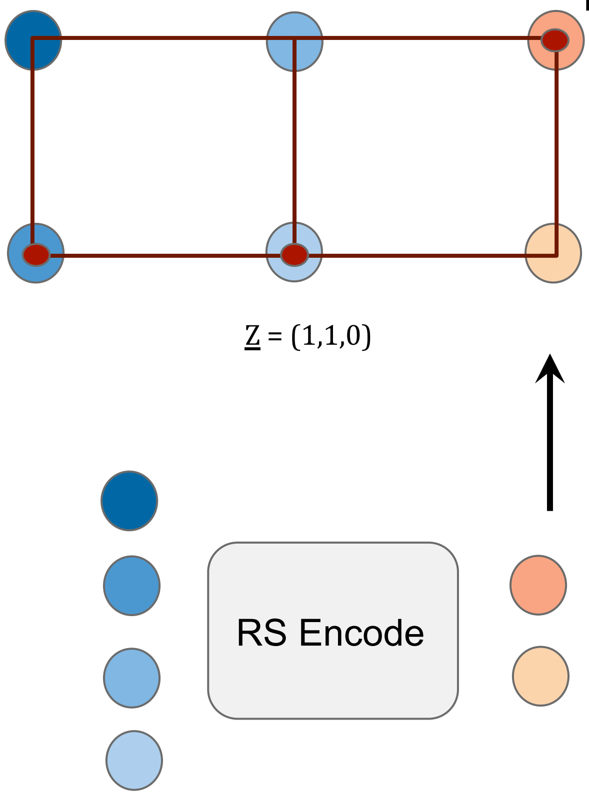
Each plane is Reed-Solomon coded to obtain parity points

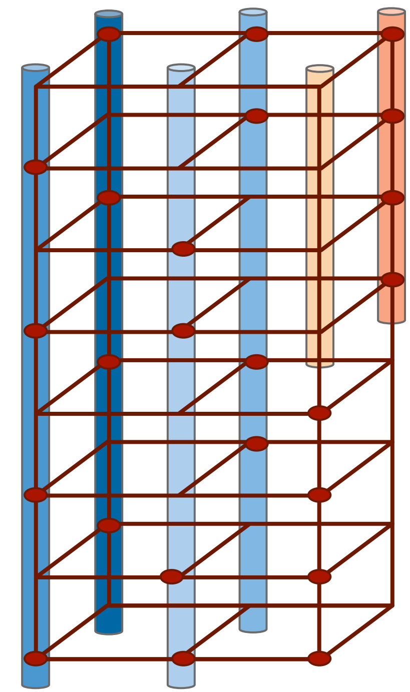


Each plane is Reed-Solomon coded to obtain parity points

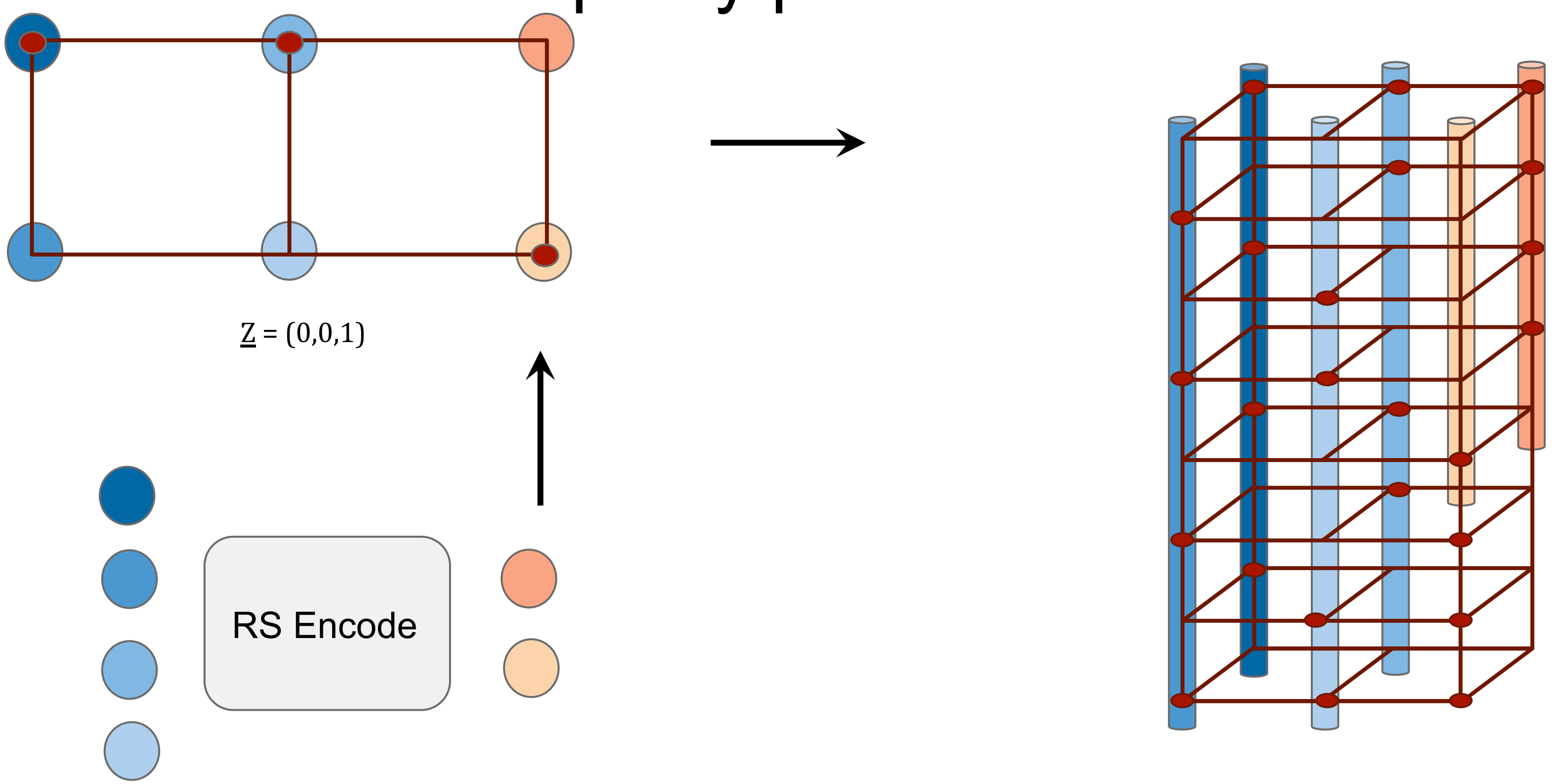


Each plane is Reed-Solomon coded to obtain parity points

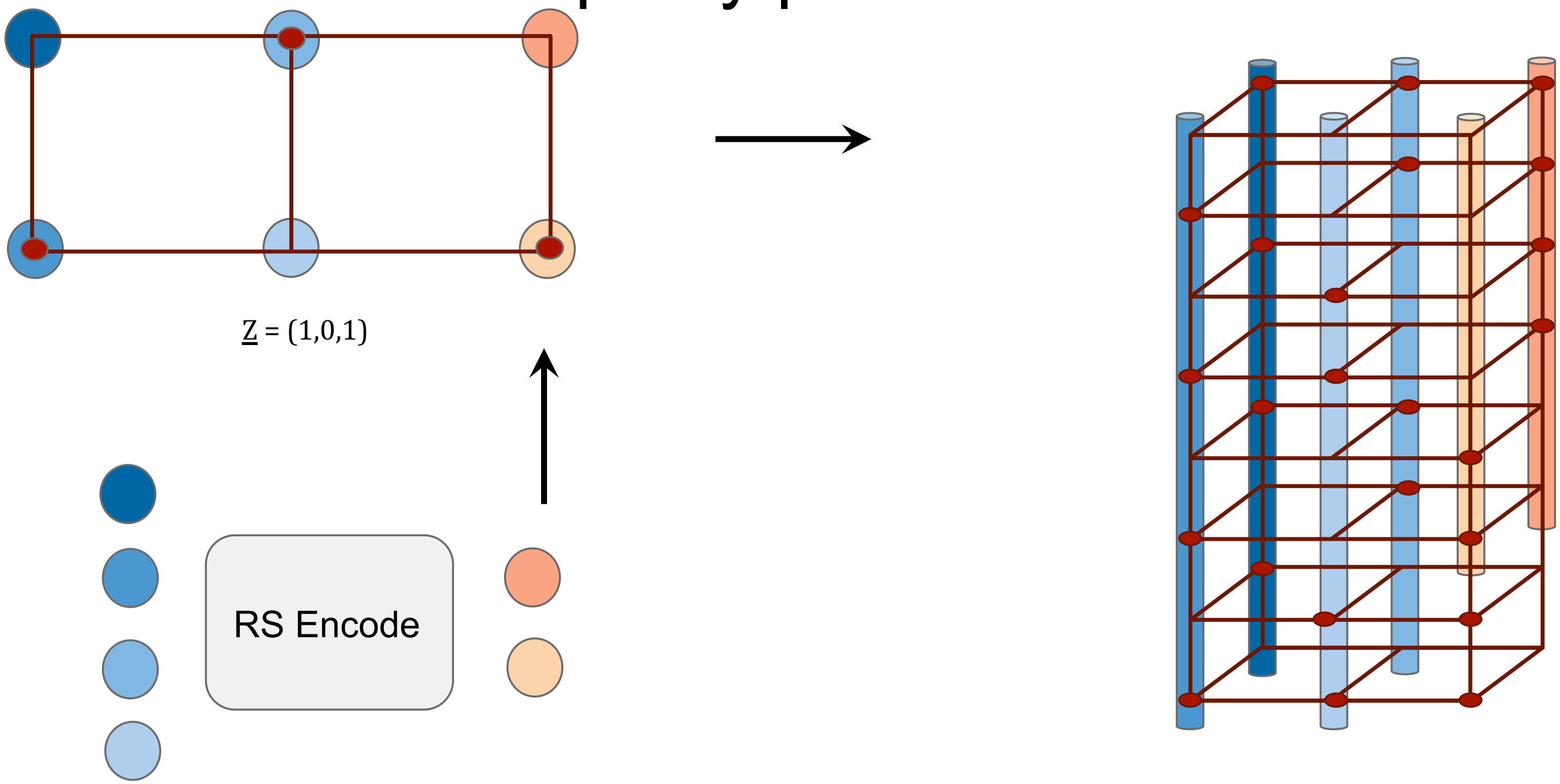




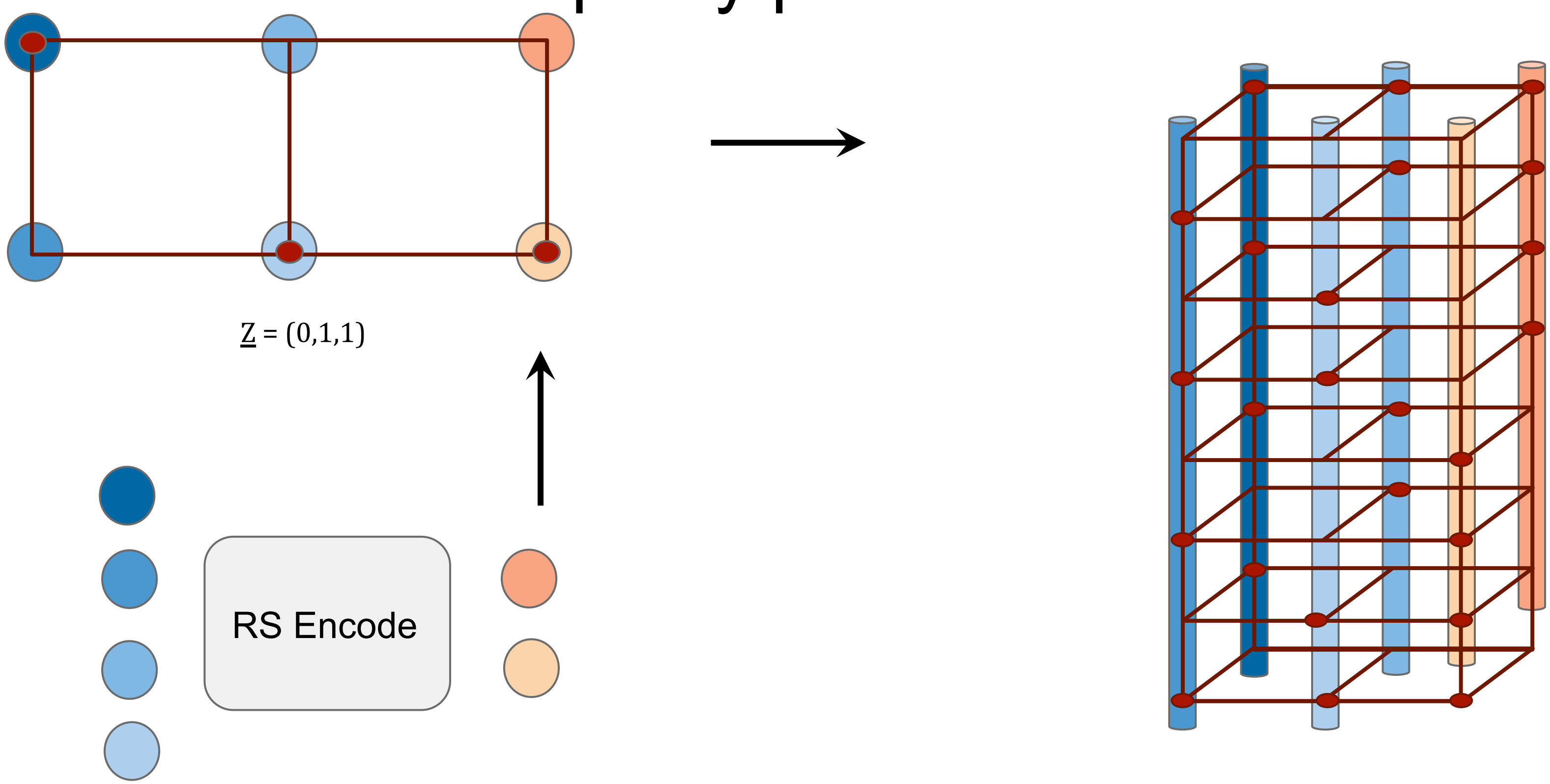
Each plane is Reed-Solomon coded to obtain parity points



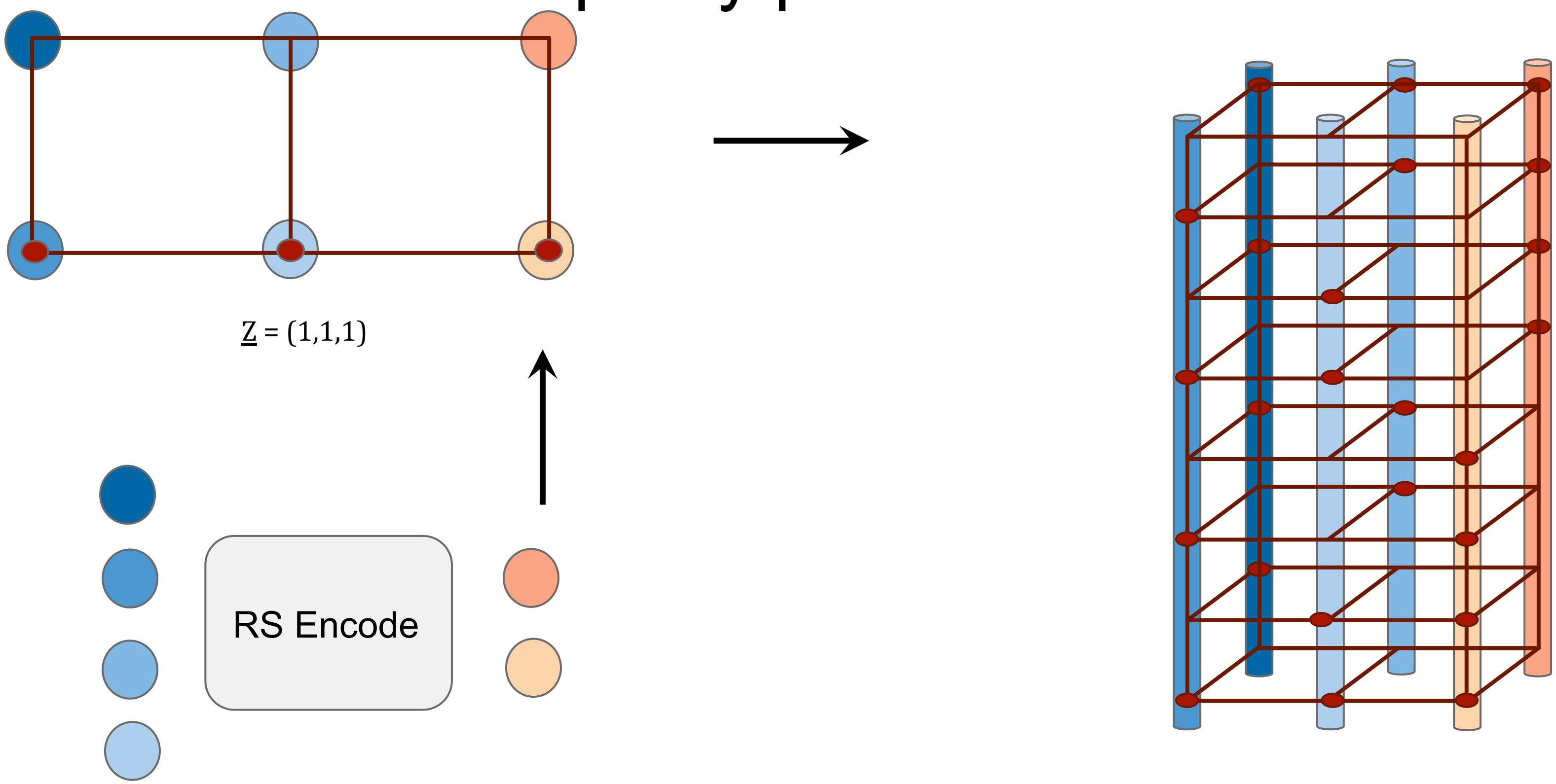
Each plane is Reed-Solomon coded to obtain parity points



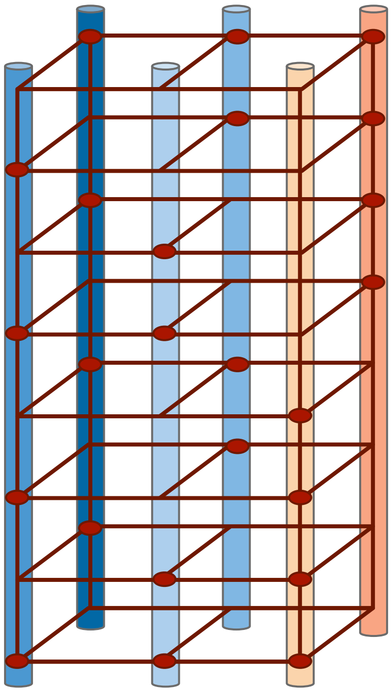
Each plane is Reed-Solomon coded to obtain parity points



Each plane is Reed-Solomon coded to obtain parity points



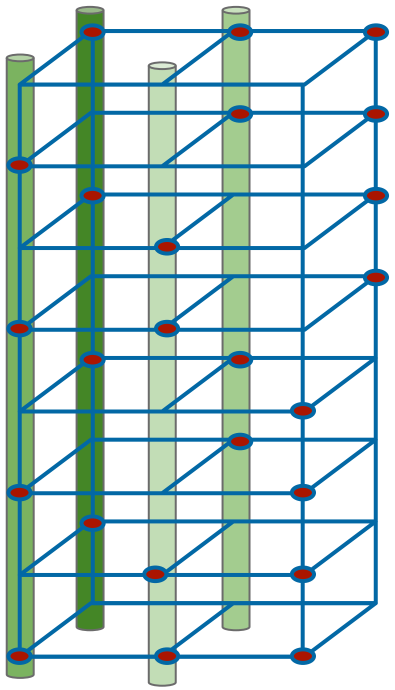
Now we have the complete Virtual data cube

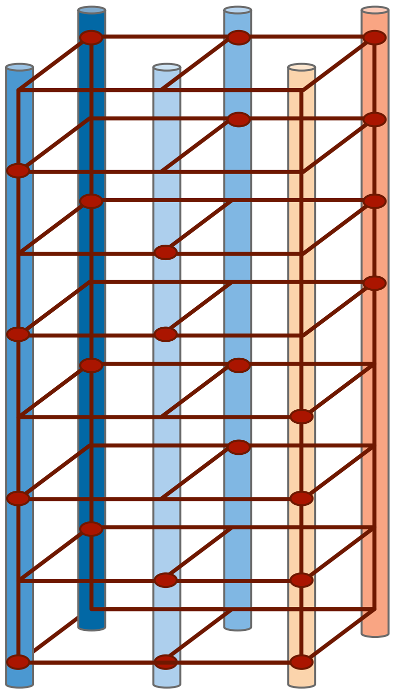


Virtual data cube

B

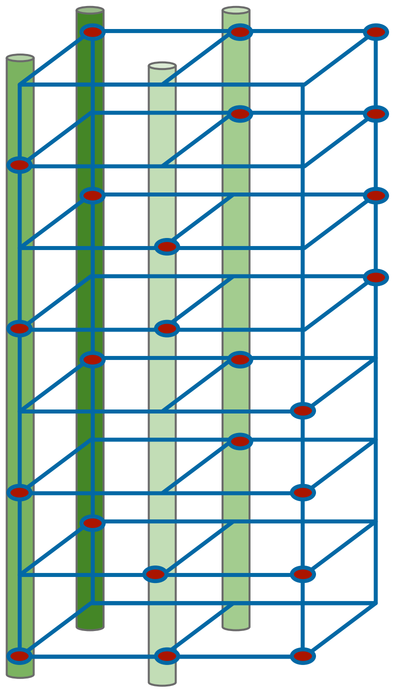
Parity points of Actual data cube can now be computed

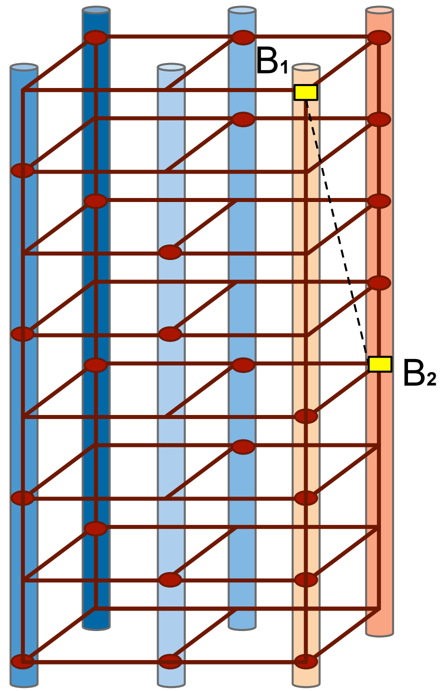




Virtual data cube
B

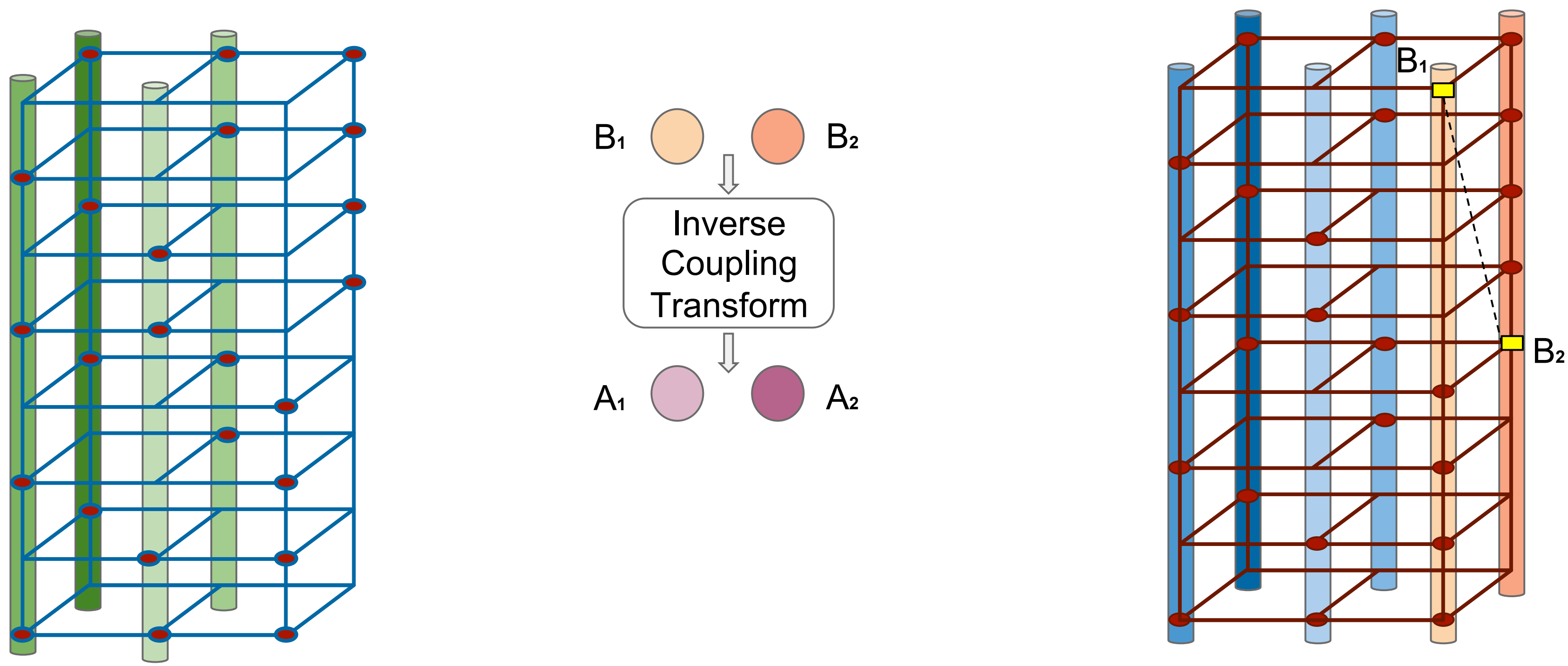
Perform decoupling





Virtual data cube
B

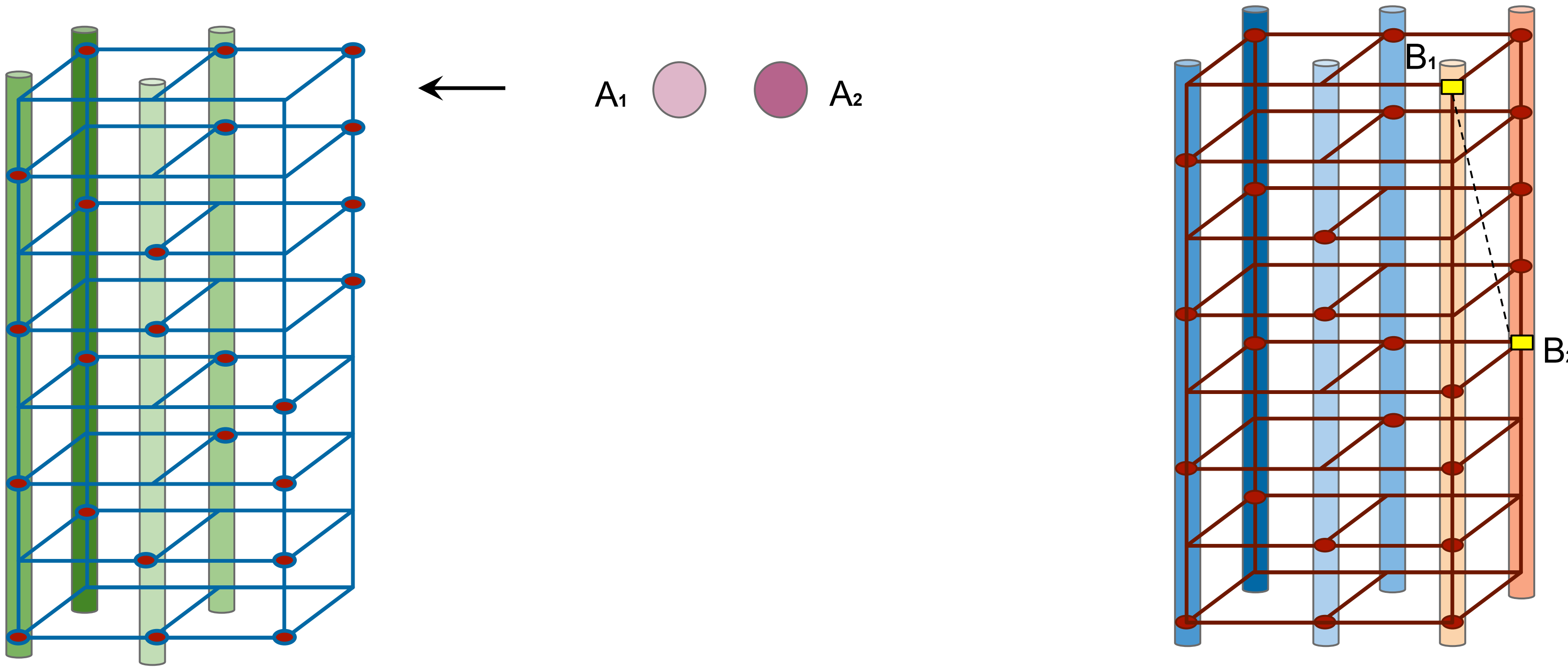
Perform decoupling



Virtual data cube

B

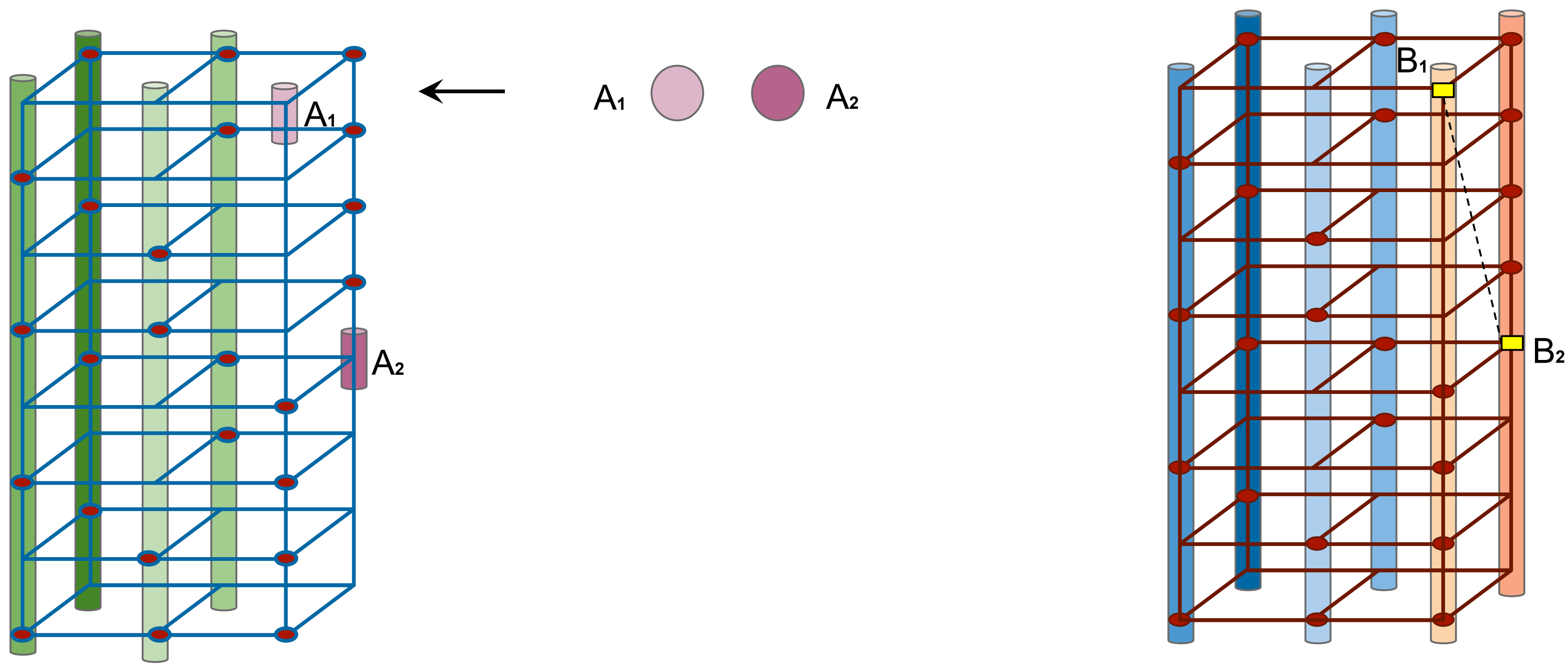
Perform decoupling



Virtual data cube

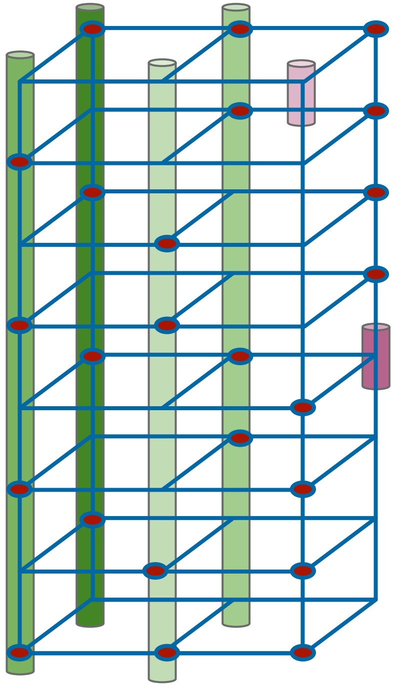
B

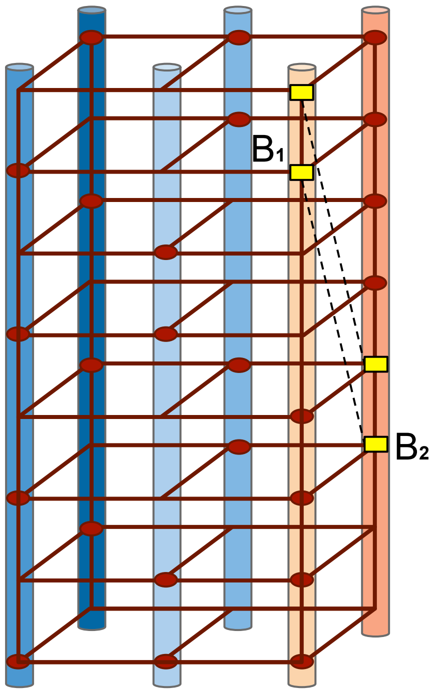
Perform decoupling



Virtual data cube
B

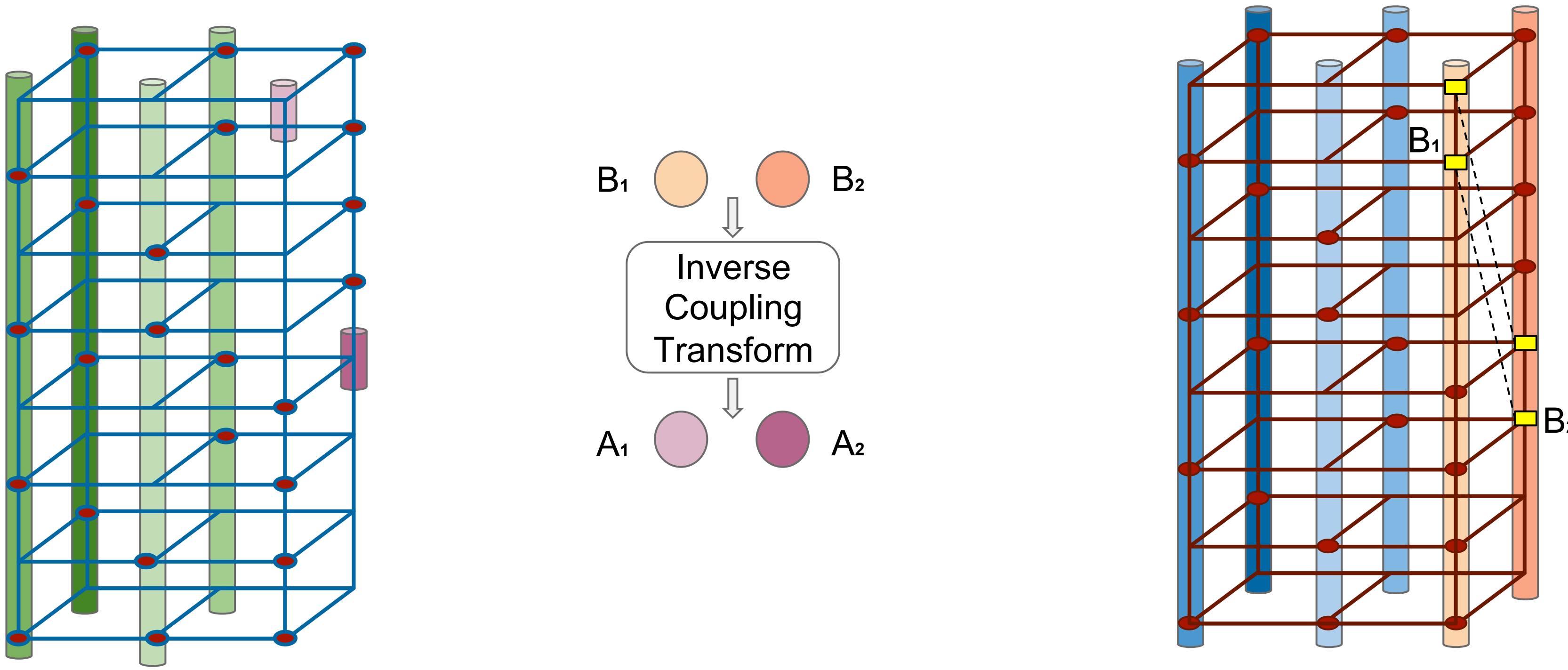
Perform decoupling





Virtual data cube
B

Perform decoupling



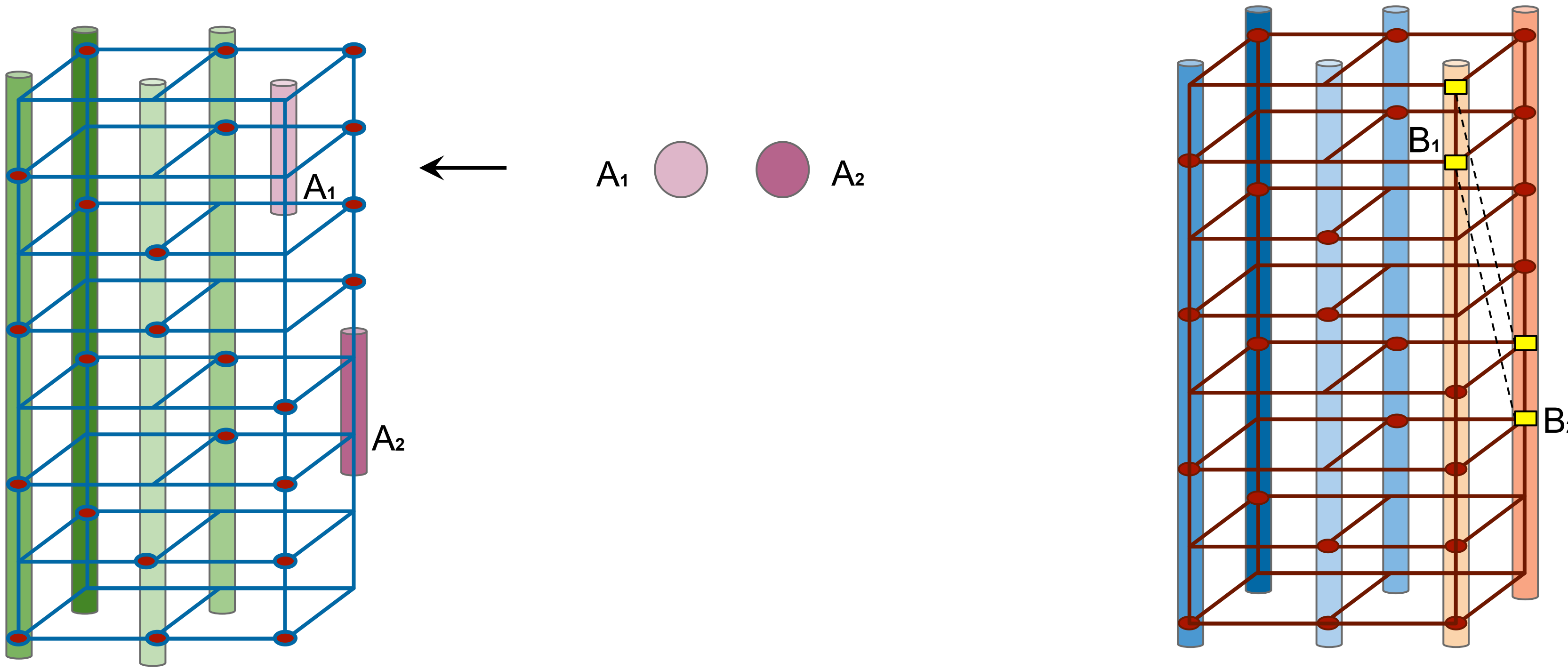
Virtual data cube

B

Perform decoupling

Virtual data cube
B

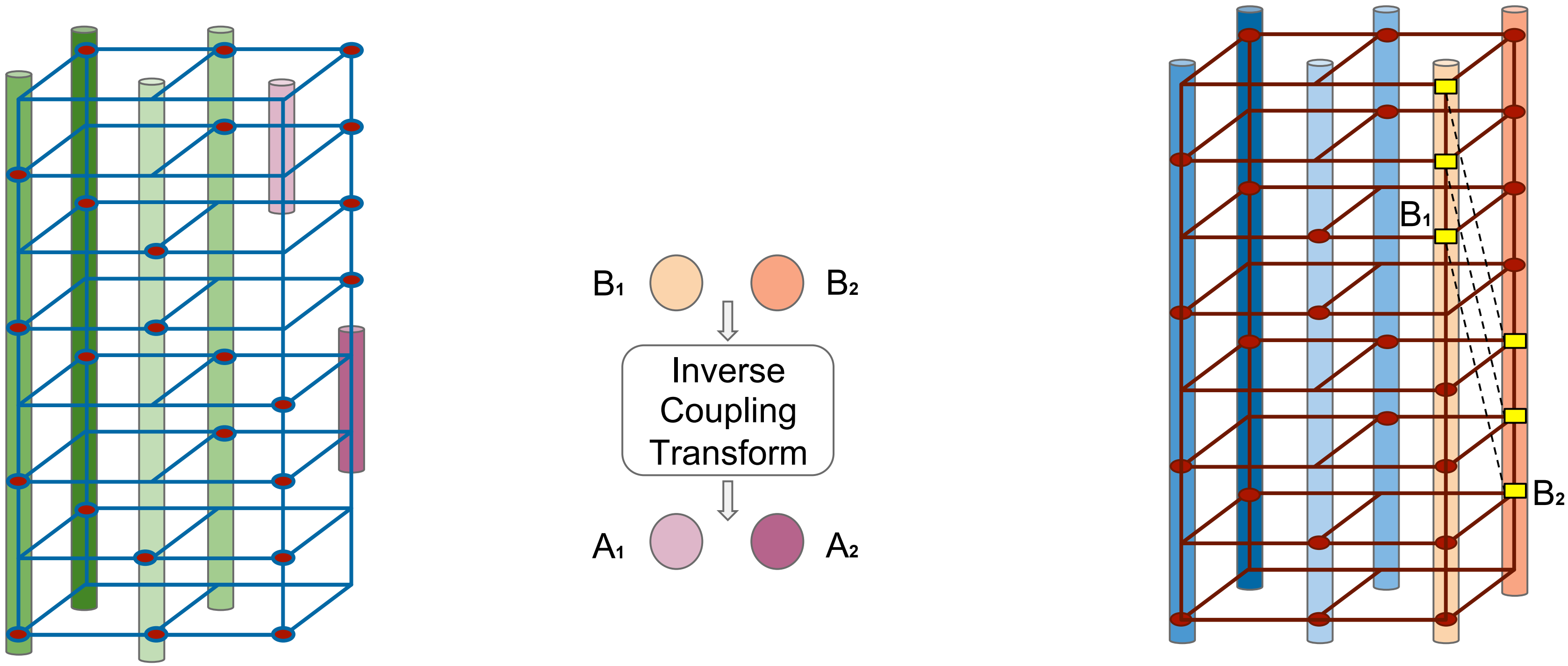
Perform decoupling



Virtual data cube

B

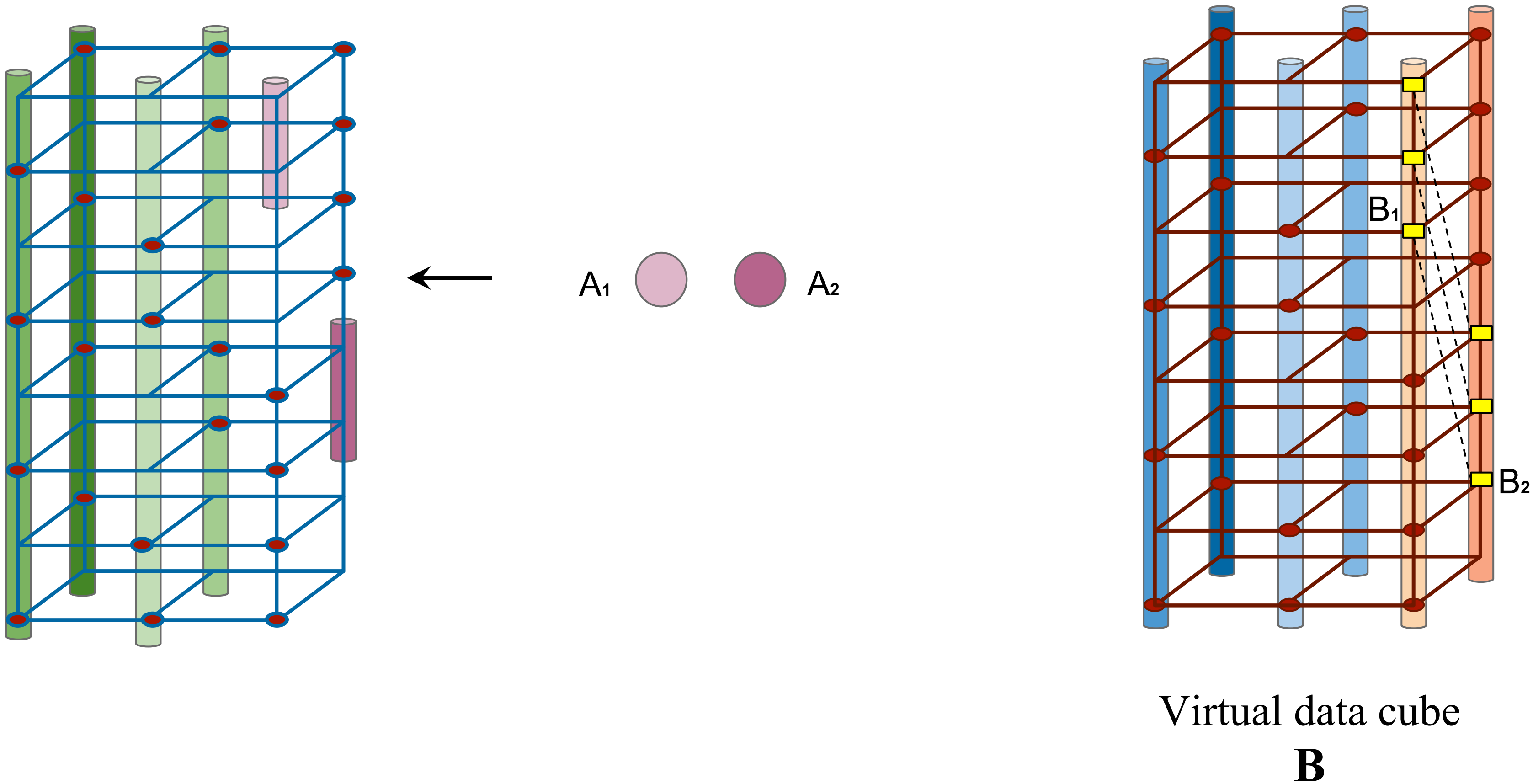
Perform decoupling



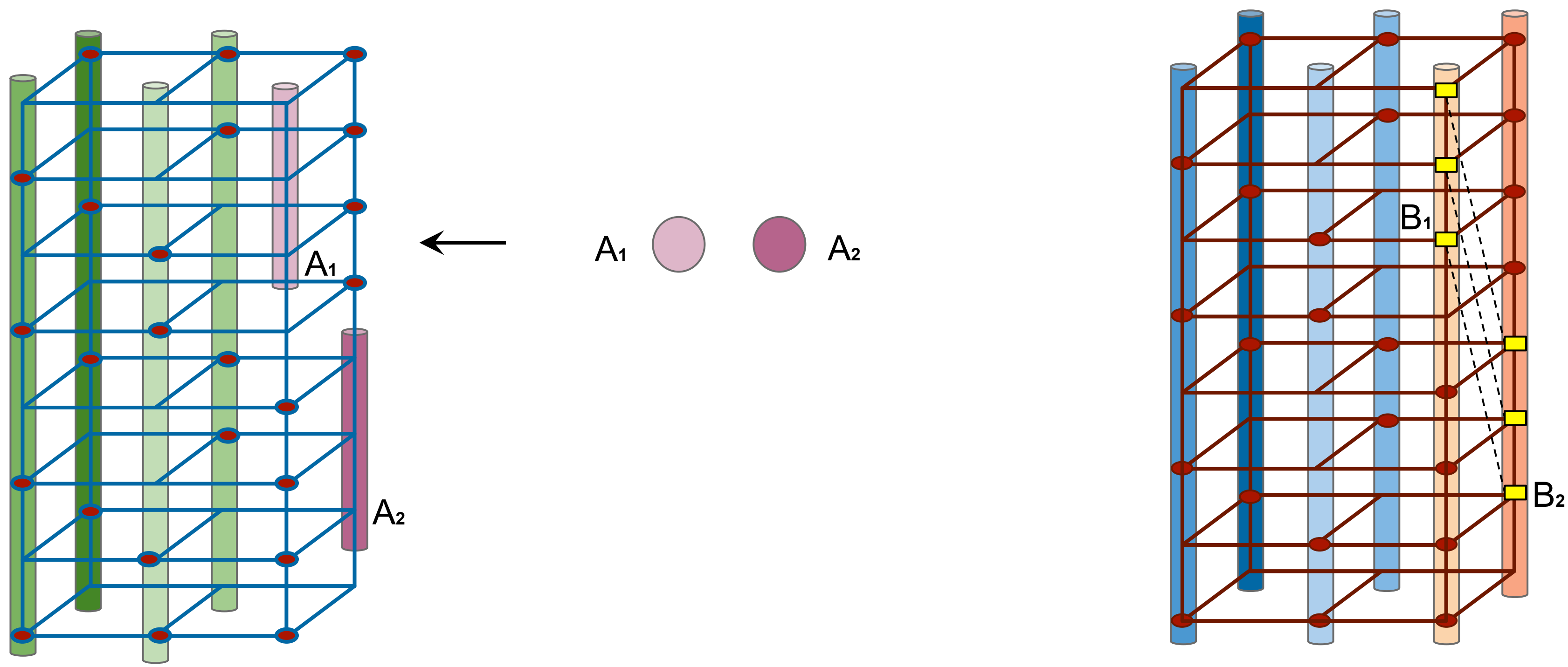
Virtual data cube

B

Perform decoupling



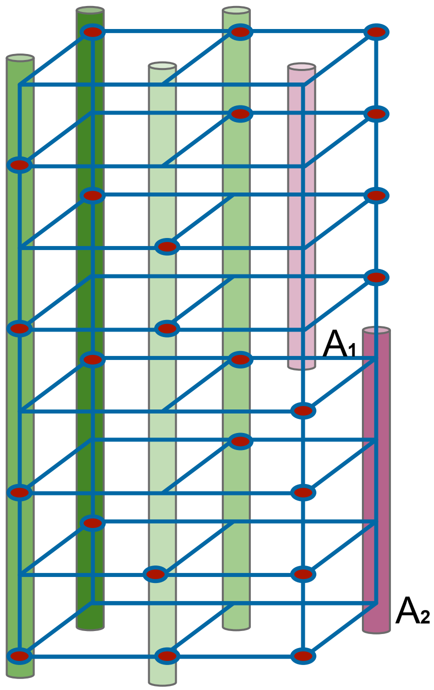
Perform decoupling

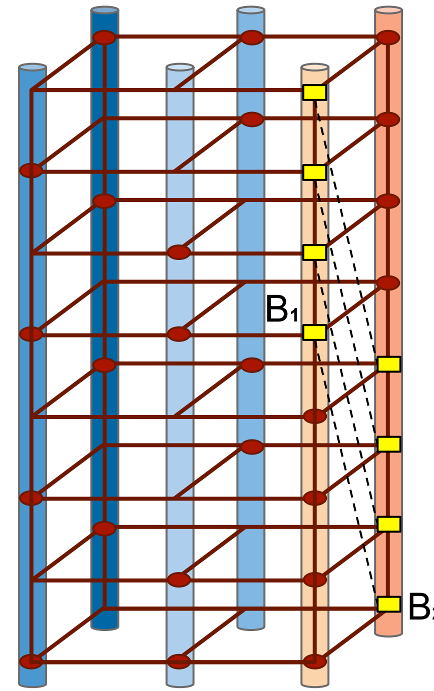


Virtual data cube

B

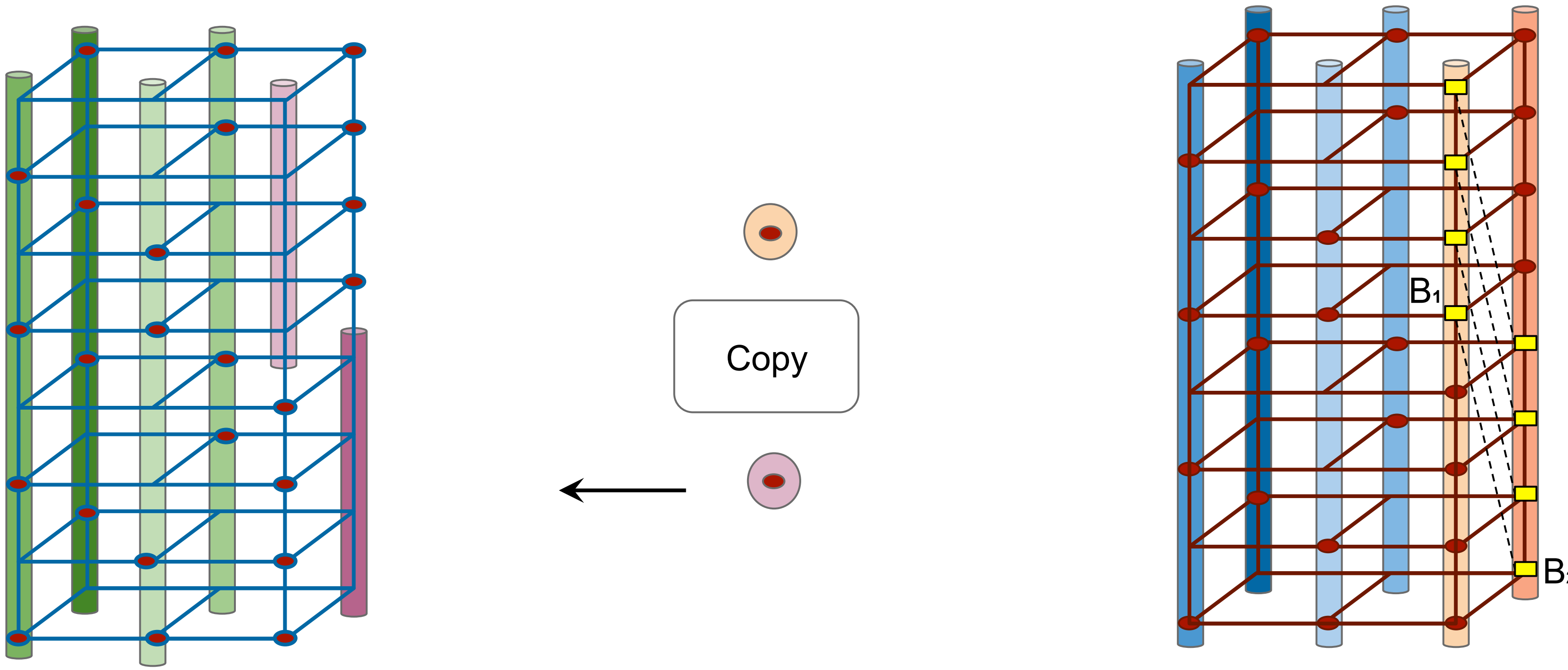
Perform decoupling





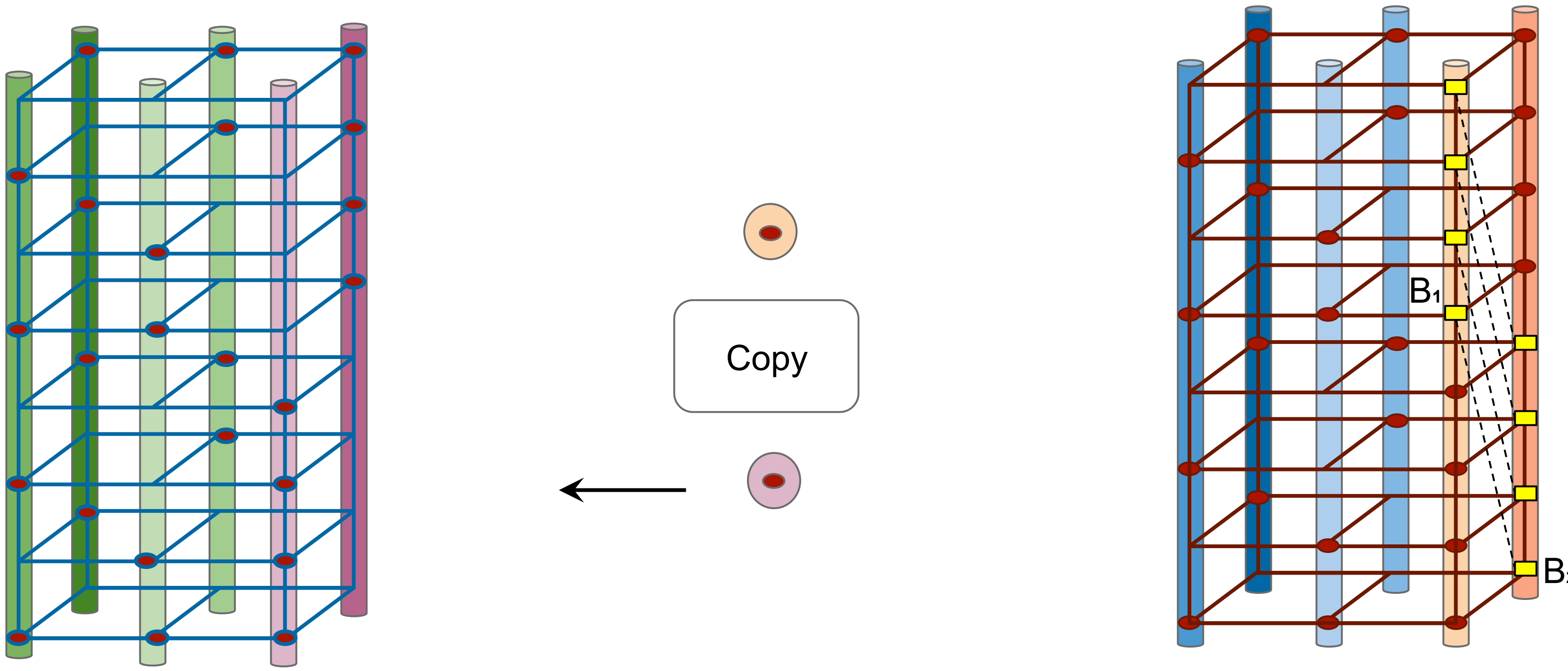
Virtual data cube
B

Red dotted points are simply carried over



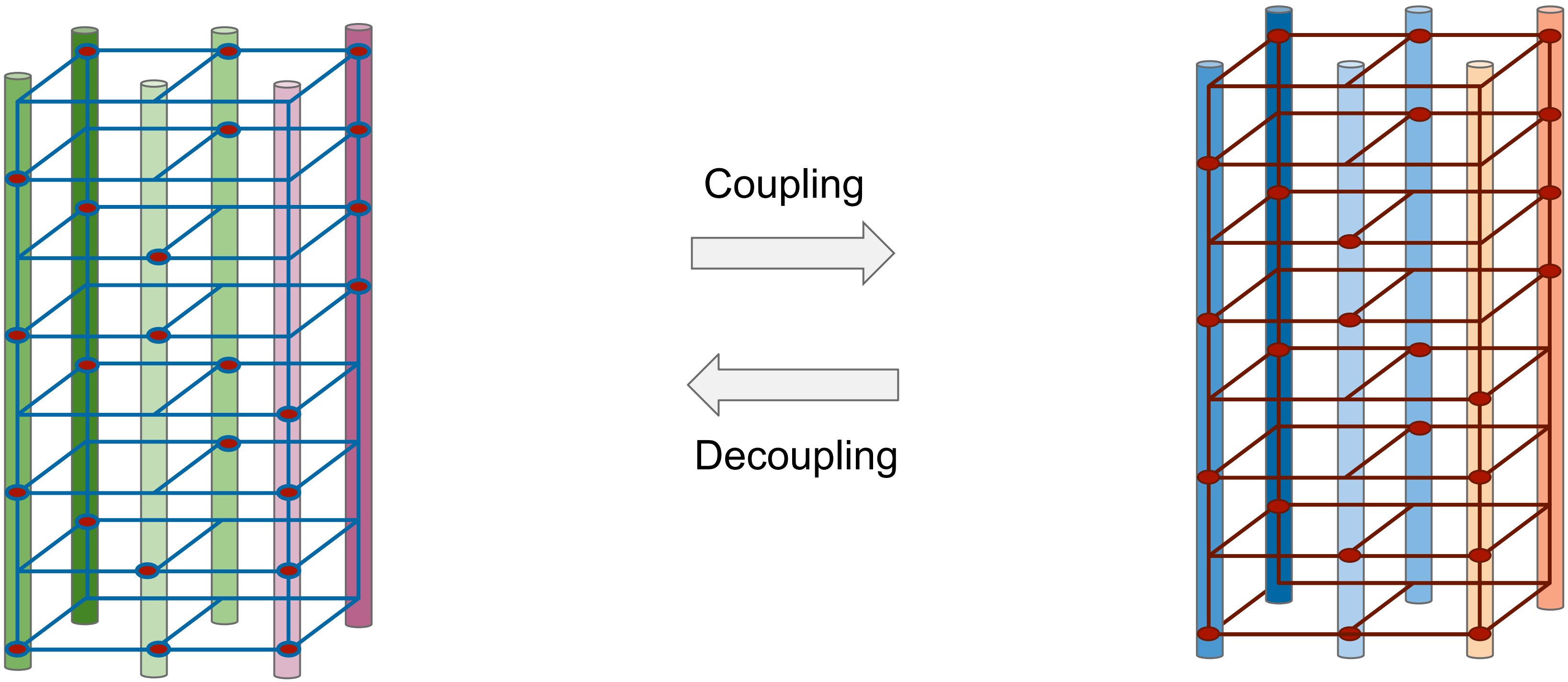
Virtual data cube
B

Red dotted points are simply carried over



Virtual data cube
B

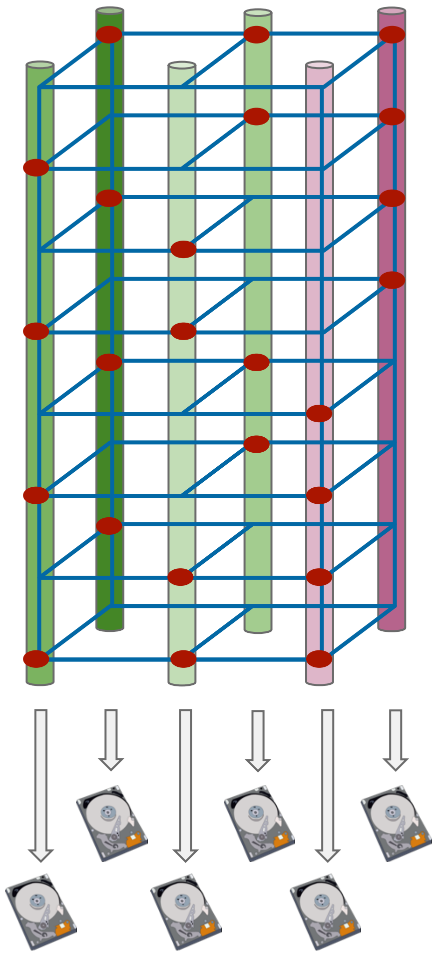
Actual and Virtual data cubes



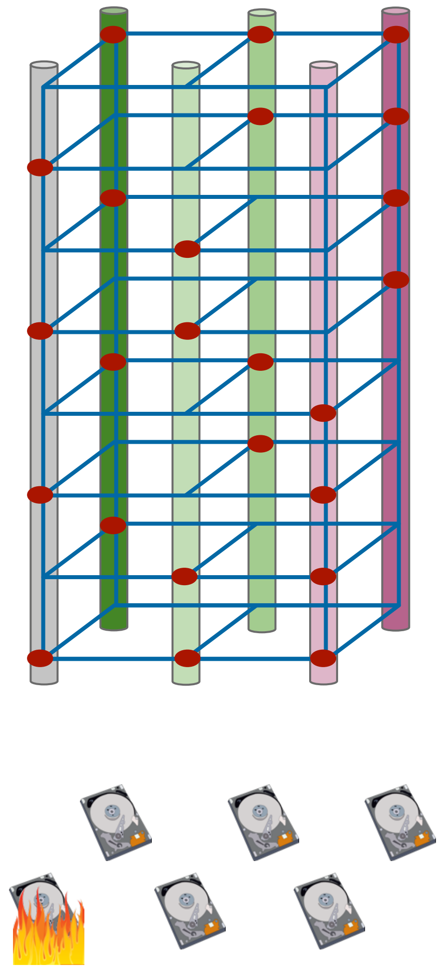
Virtual data cube
A

Virtual data cube
B

The encoding is now completed!

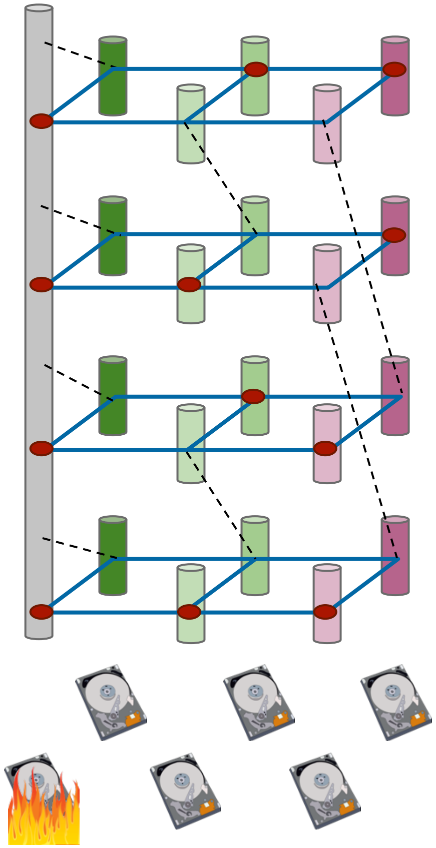


Problem of Node Repair: One node fails



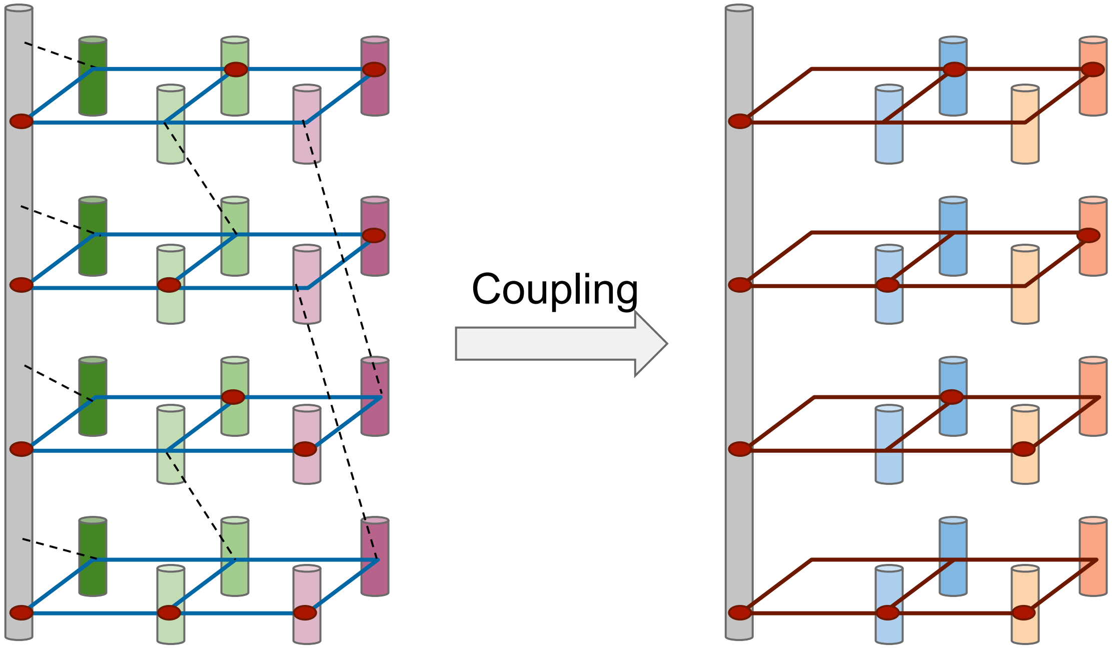
Problem of Node Repair: One node fails

For this example, only half of the planes participate in repair

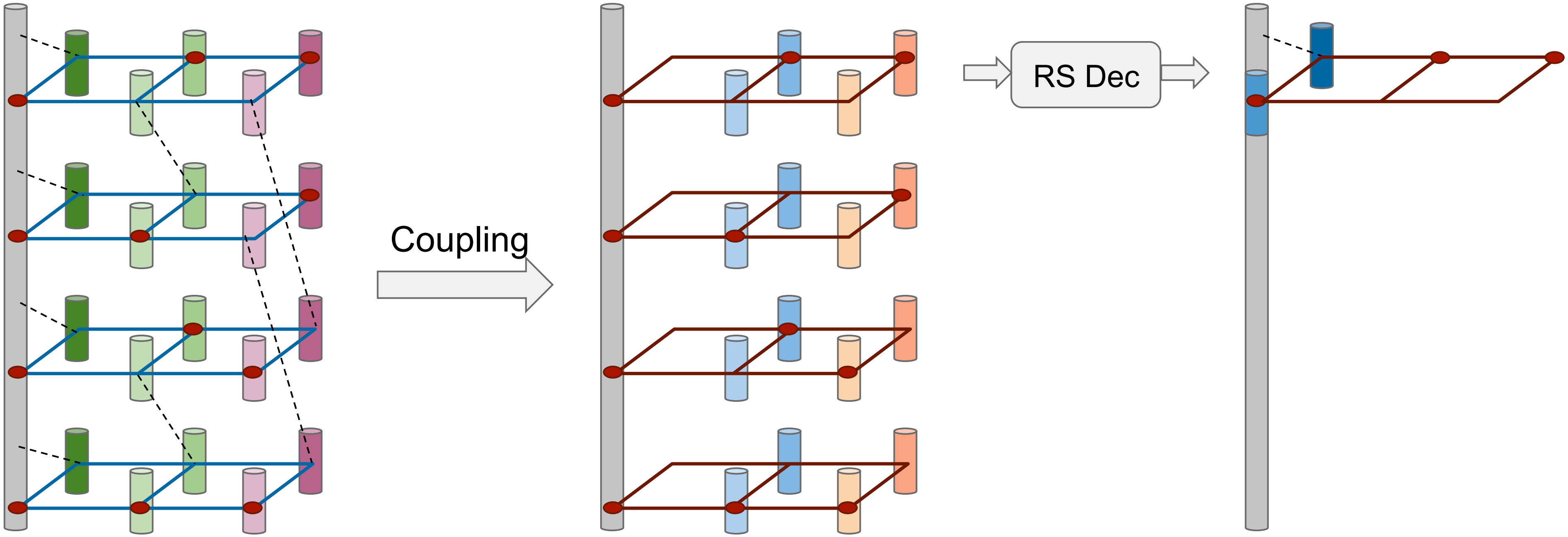


- Total Helper Data = 2MB X 4 X 5 = 40MB
- Opposed to RS code = 16MB X 4 = 64MB
- Much larger savings seen for m > 2

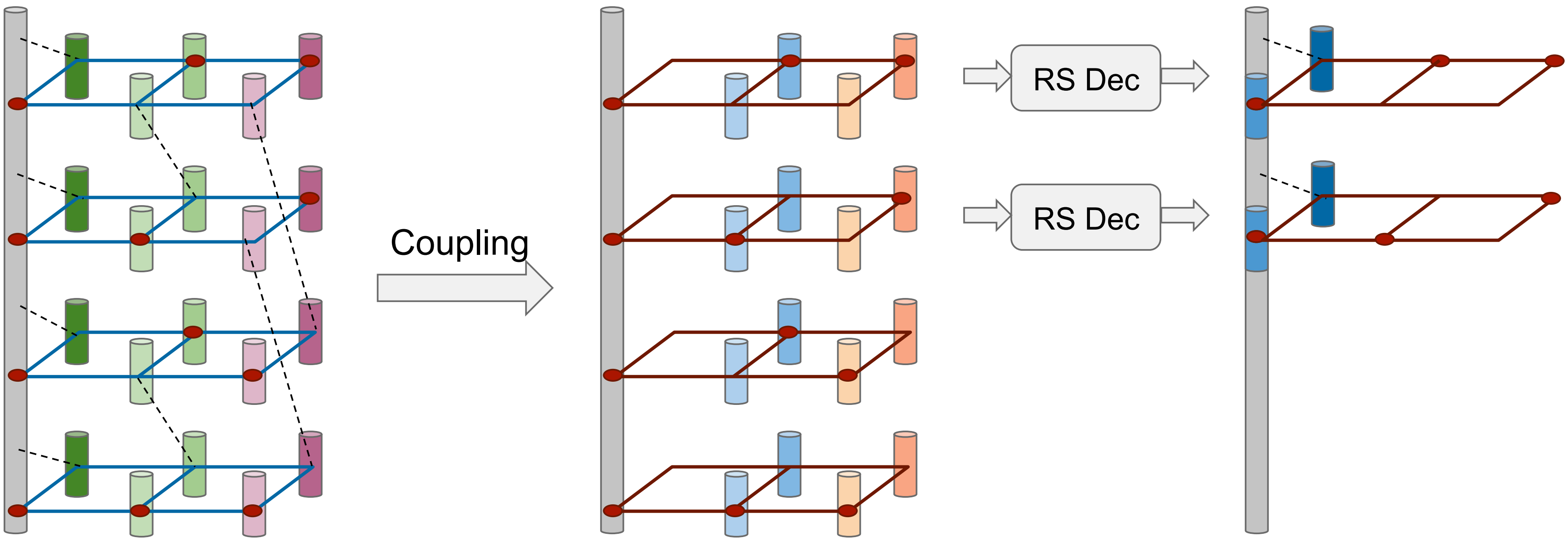
Couple points



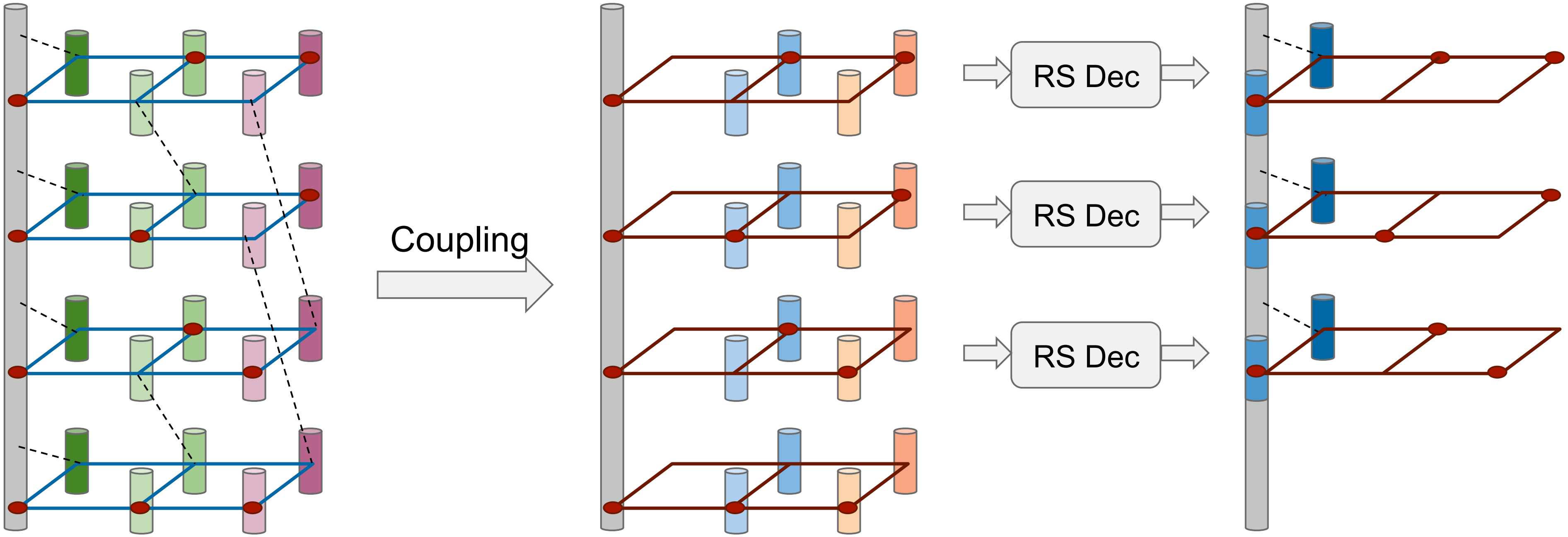
Run RS decoding on each of the selected planes



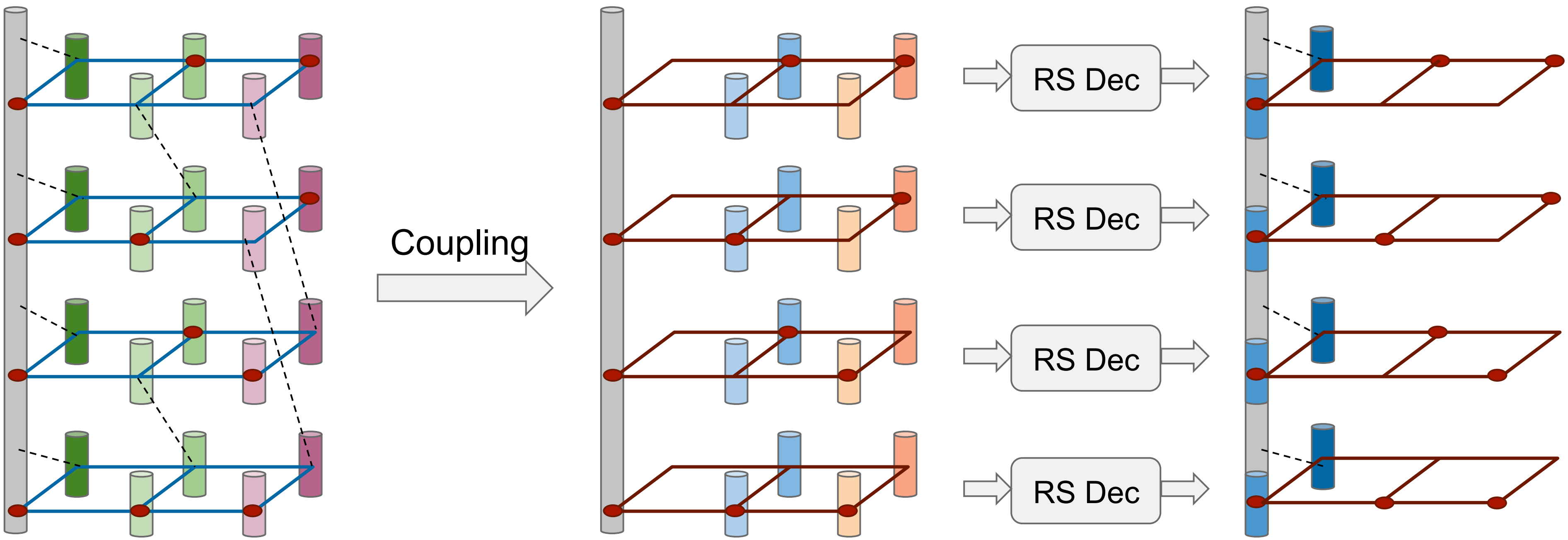
Run RS decoding on each of the selected planes



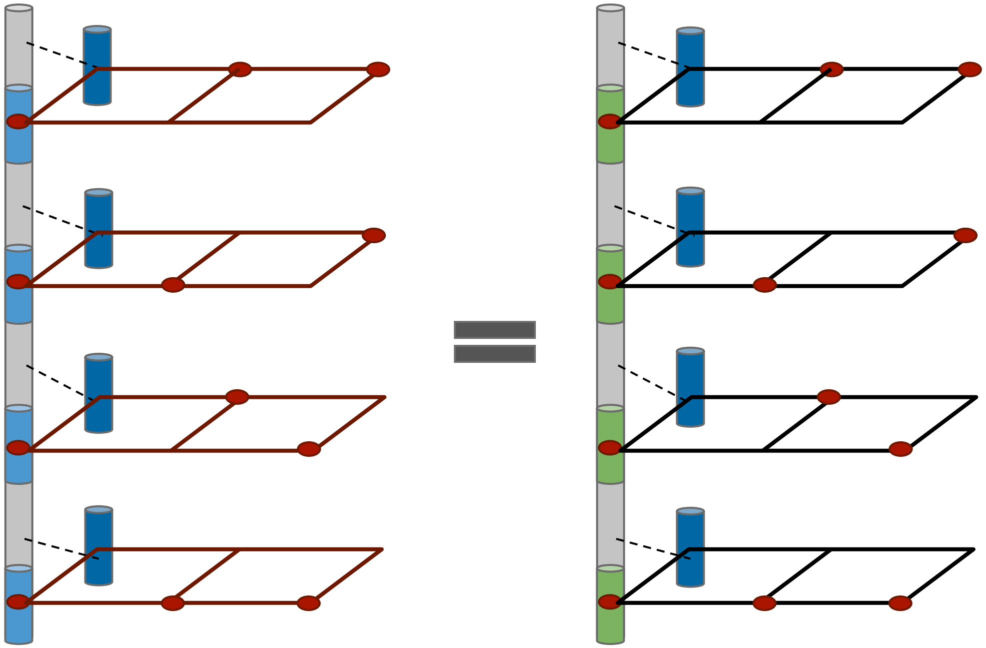
Run RS decoding on each of the selected planes



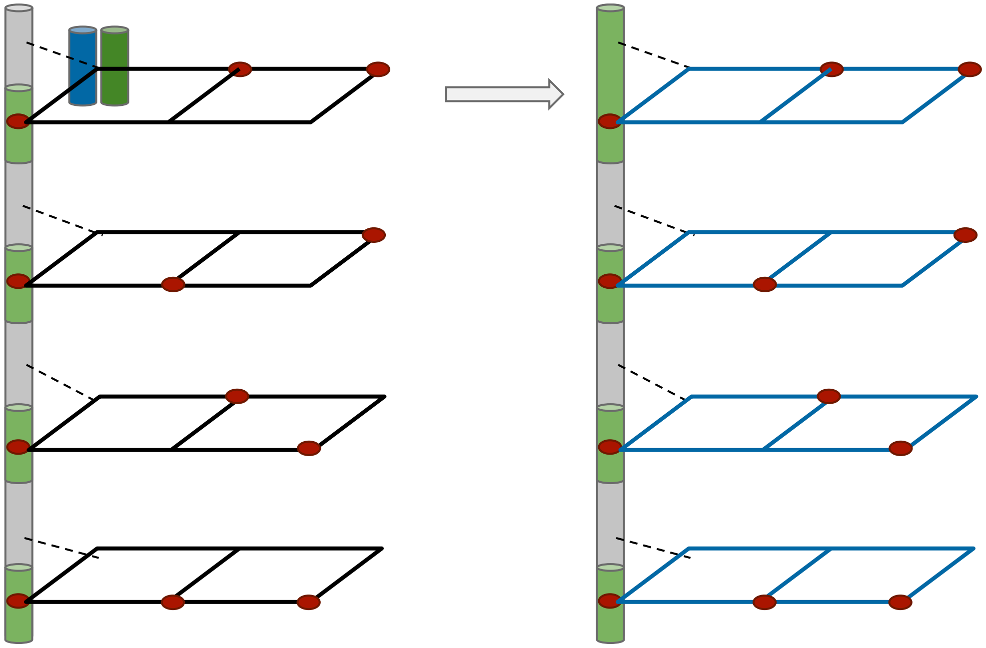
Run RS decoding on each of the selected planes



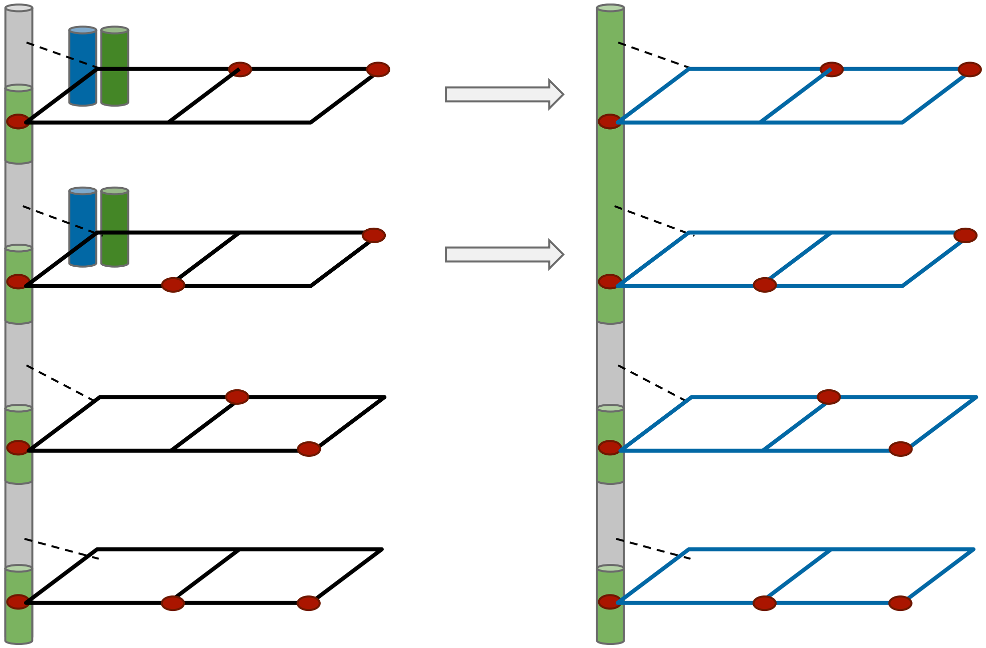
Half the number of required points are now already computed



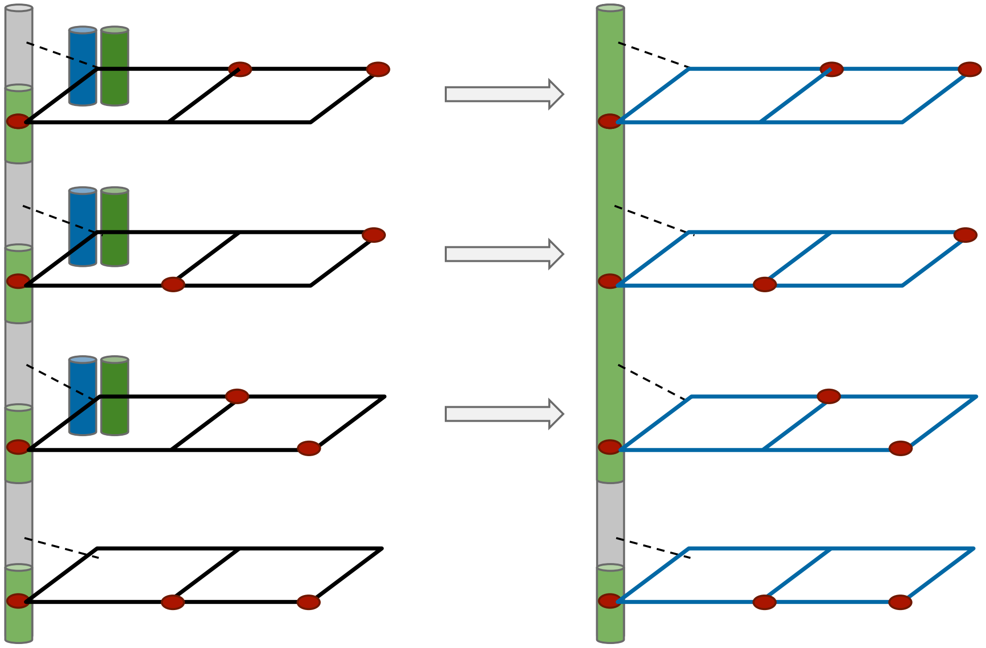
Remaining points are computed by coupling transform



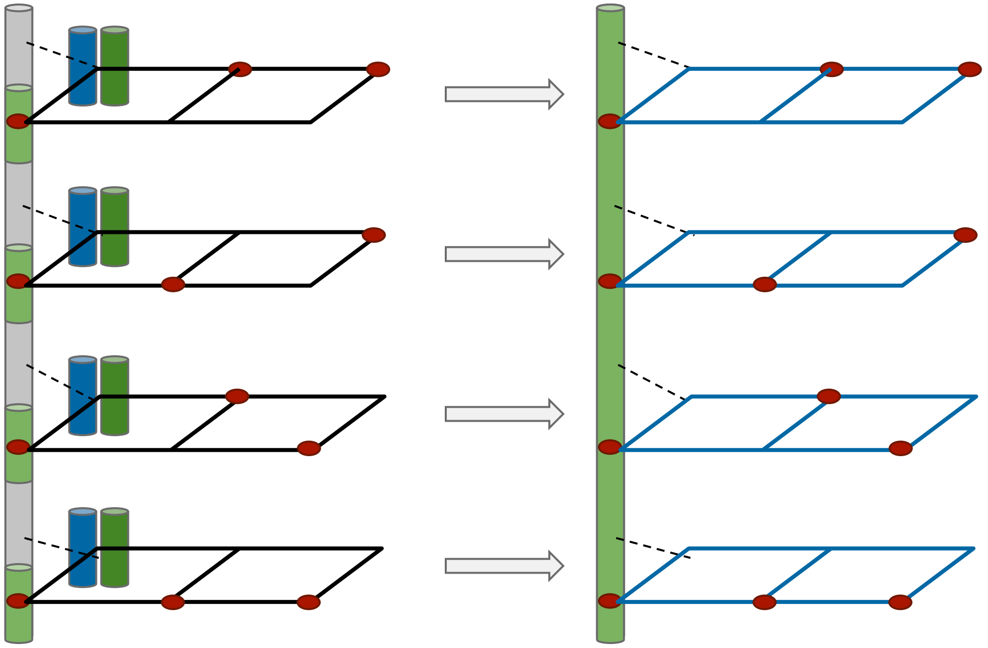
Remaining points are computed by coupling transform



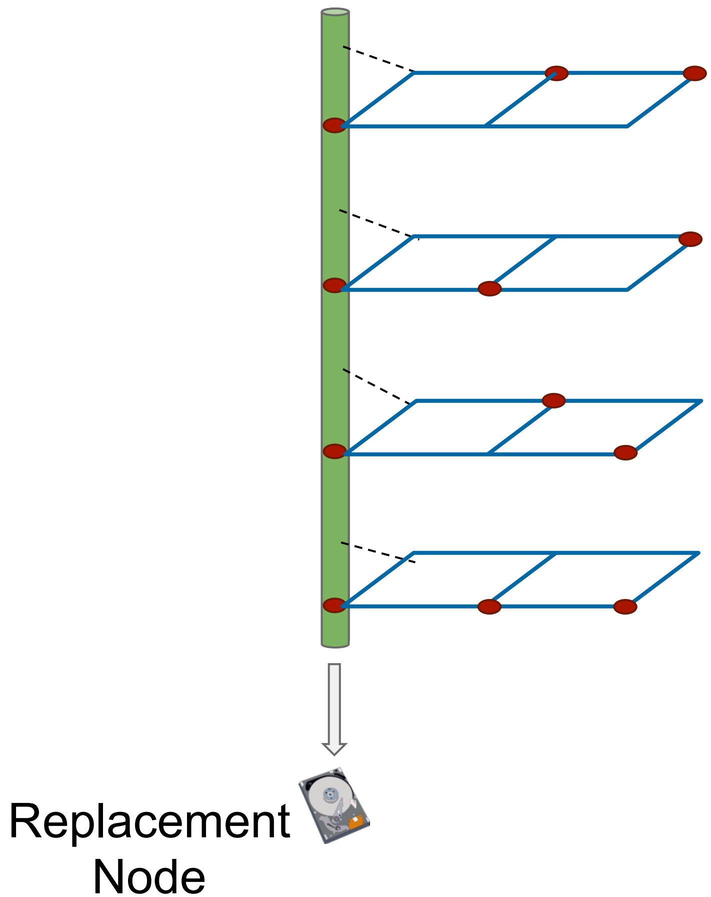
Remaining points are computed by coupling transform



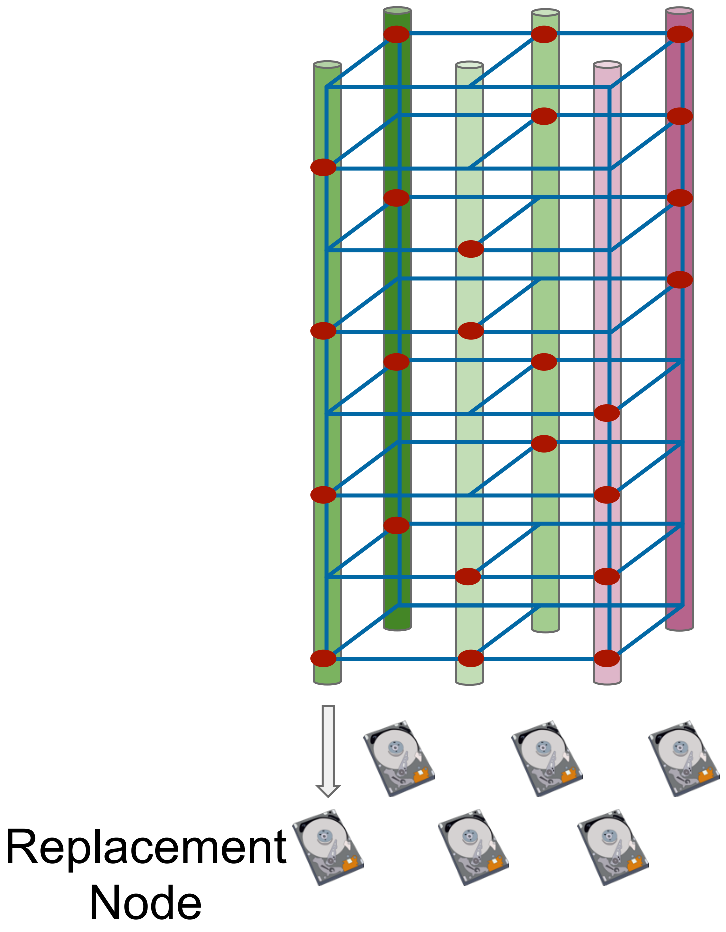
Remaining points are computed by coupling transform



Contents of the failed node are now completely recovered



Node Repair done: system back to original state!



Thanks!