Sorting on the Cray X1

Helene Kulsrud
CCR-P

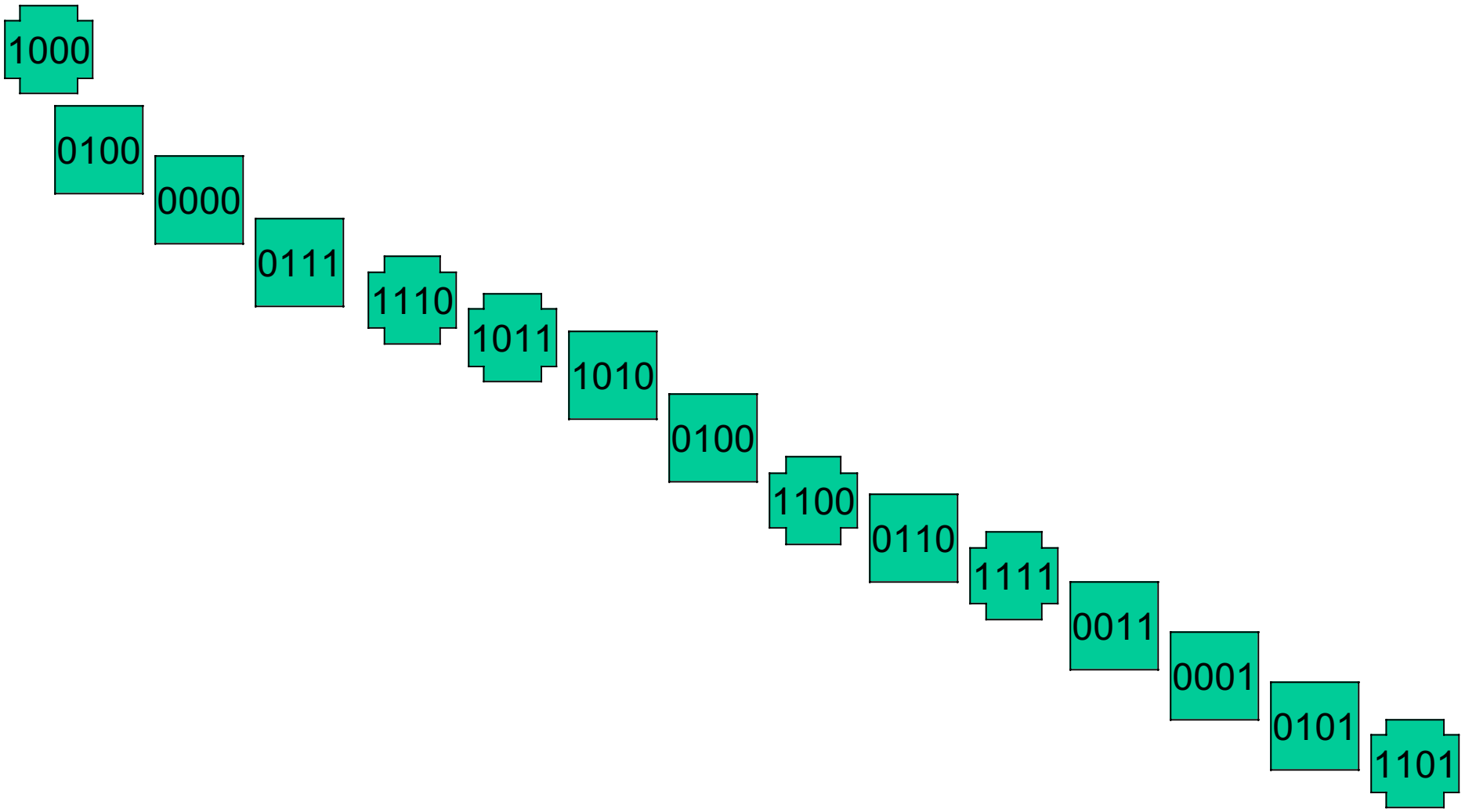
Sorting Methods

- Heap Sort
- Bubble Sort
- Insertion Sort
- Shellsort
- Mergesort
- Quicksort
- **Radix Sort**
- **Radix Exchange Sort**
- **etc…**
- **Parallel Sorts etc..**

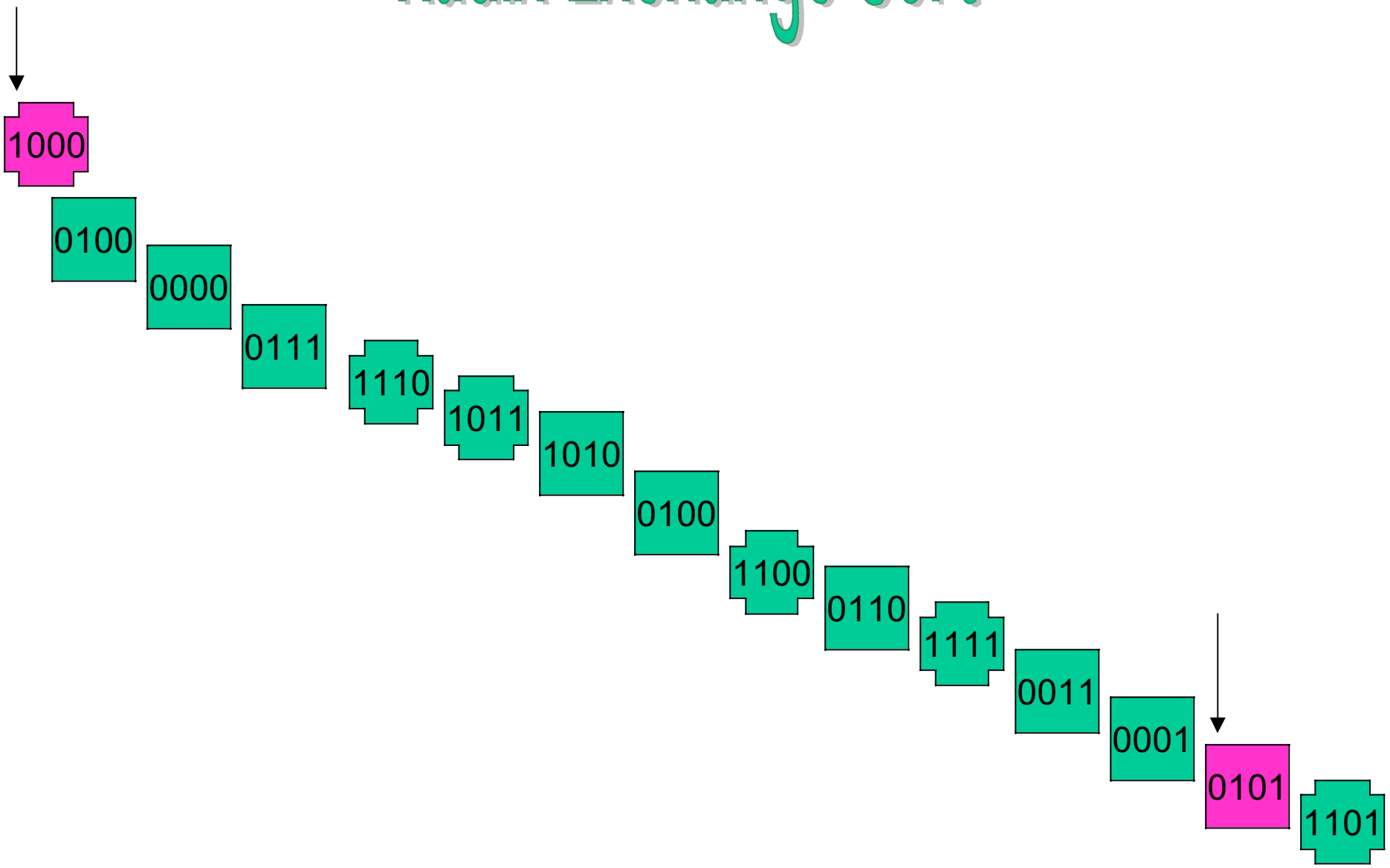
Sorting on the Cray X1

- QKSORT – A scalar sort (radix exchange)
- PSORT - A vectorized sort (radix)
- MSORT - A multiprocessor sort

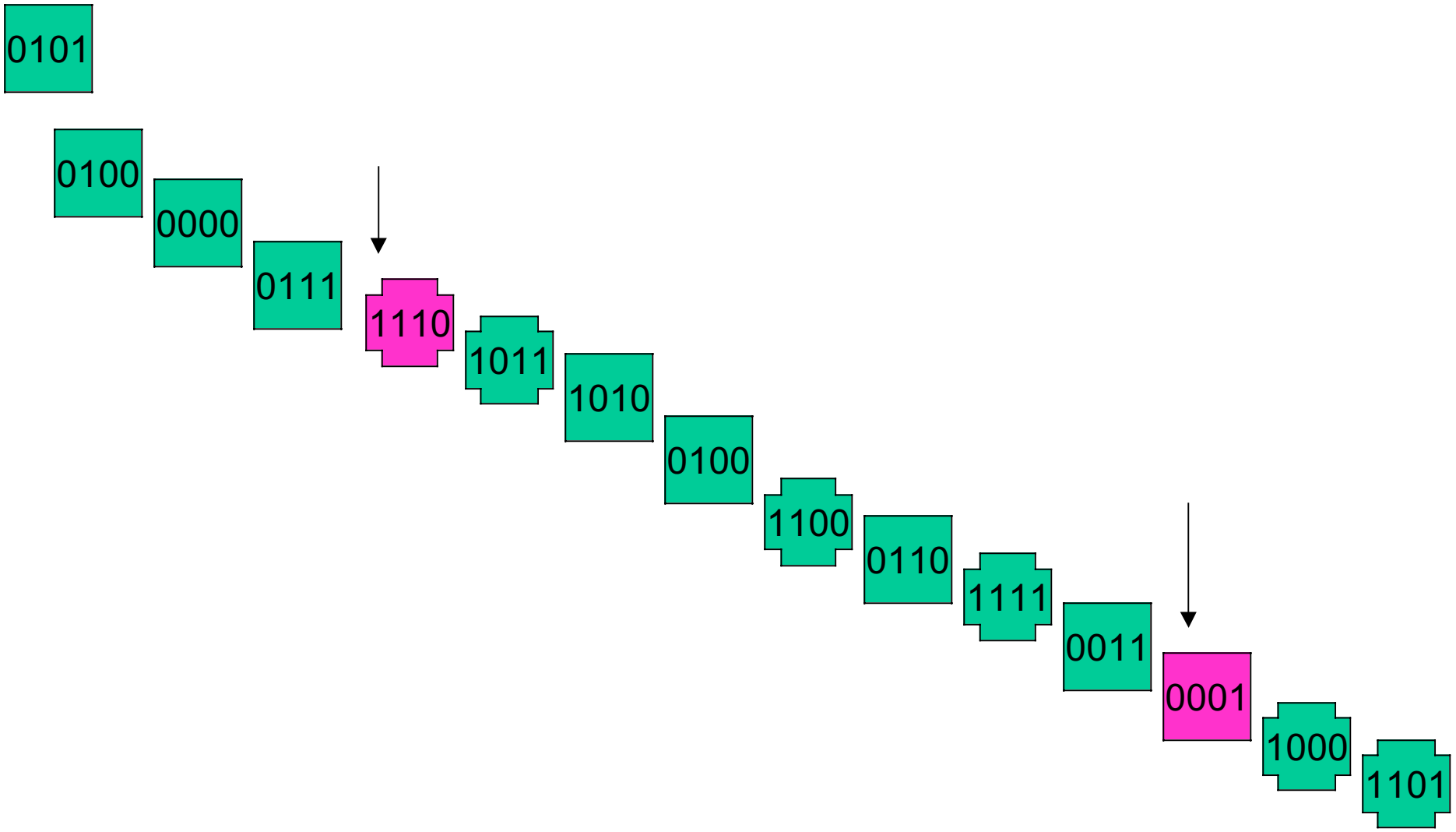
Radix Exchange Sort



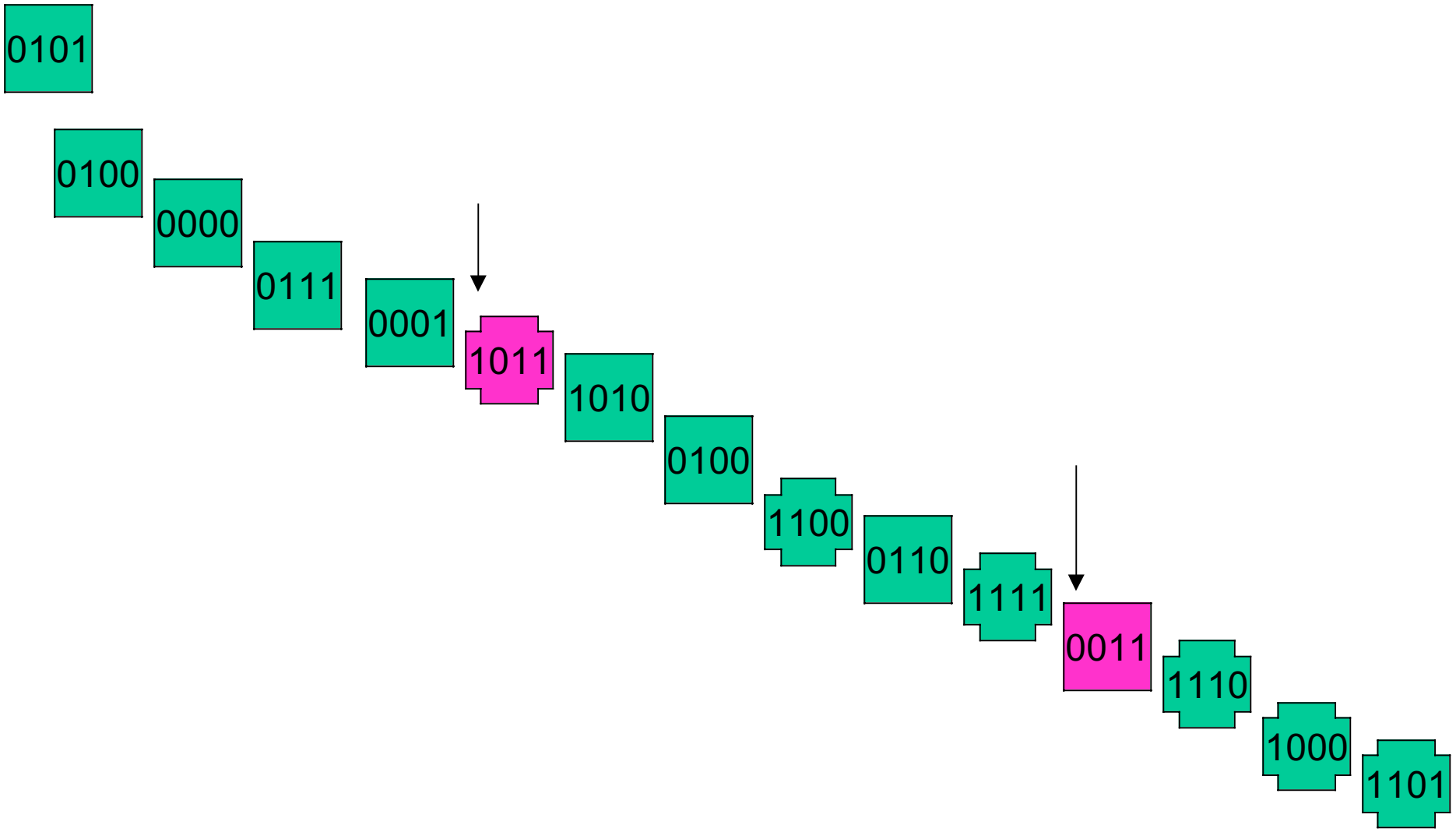
Radix Exchange Sort



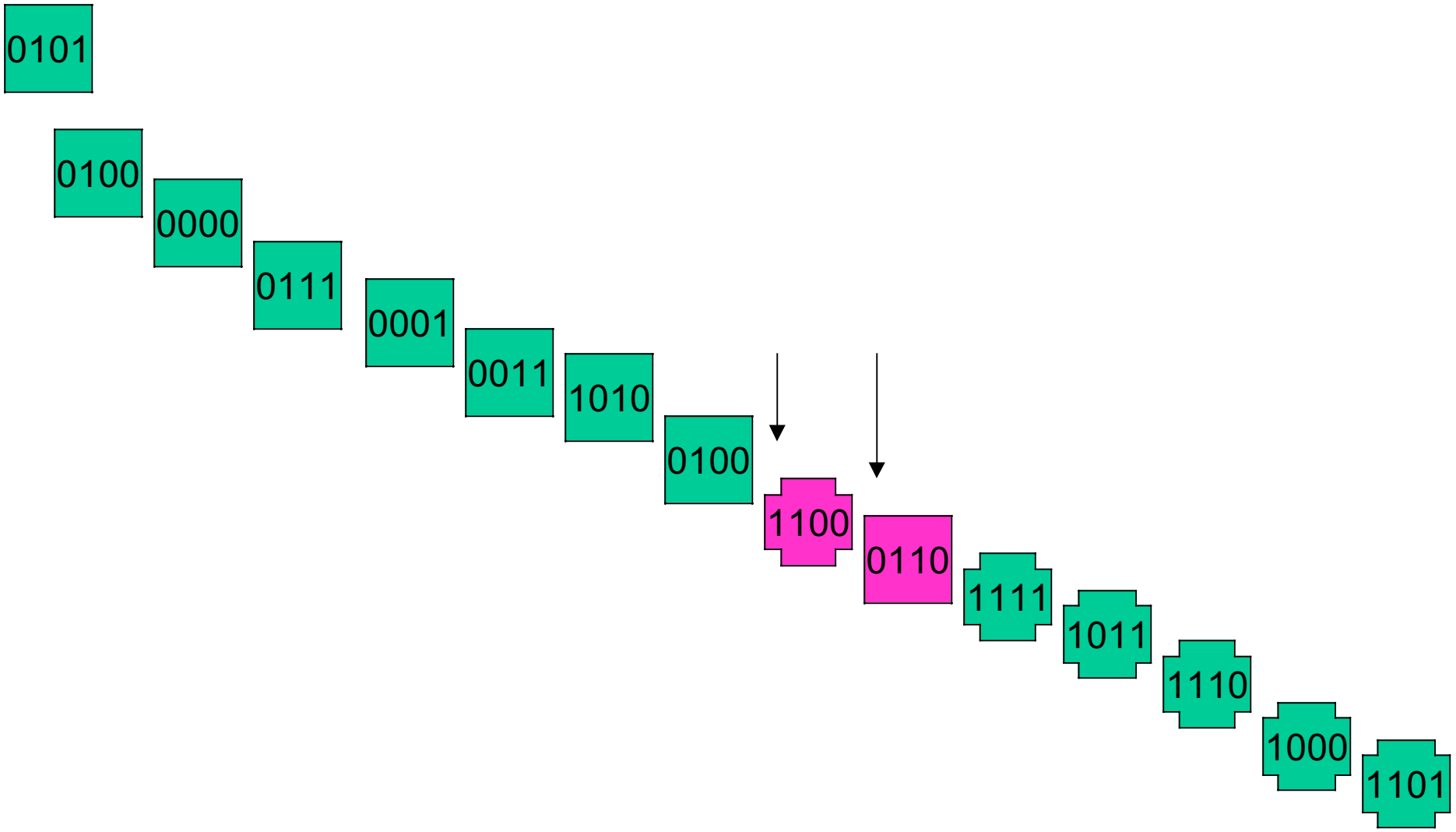
Radix Exchange Sort



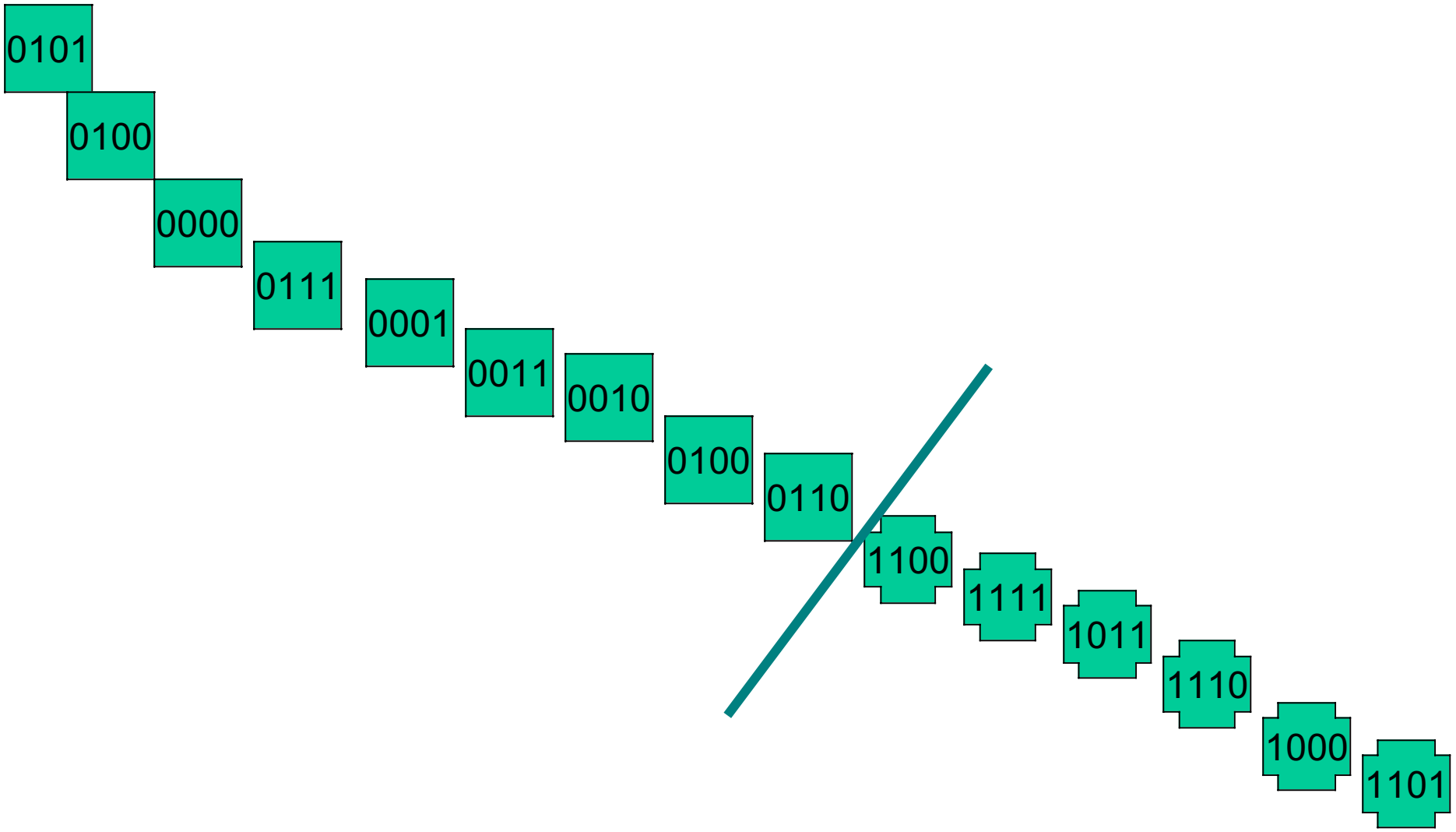
Radix Exchange Sort



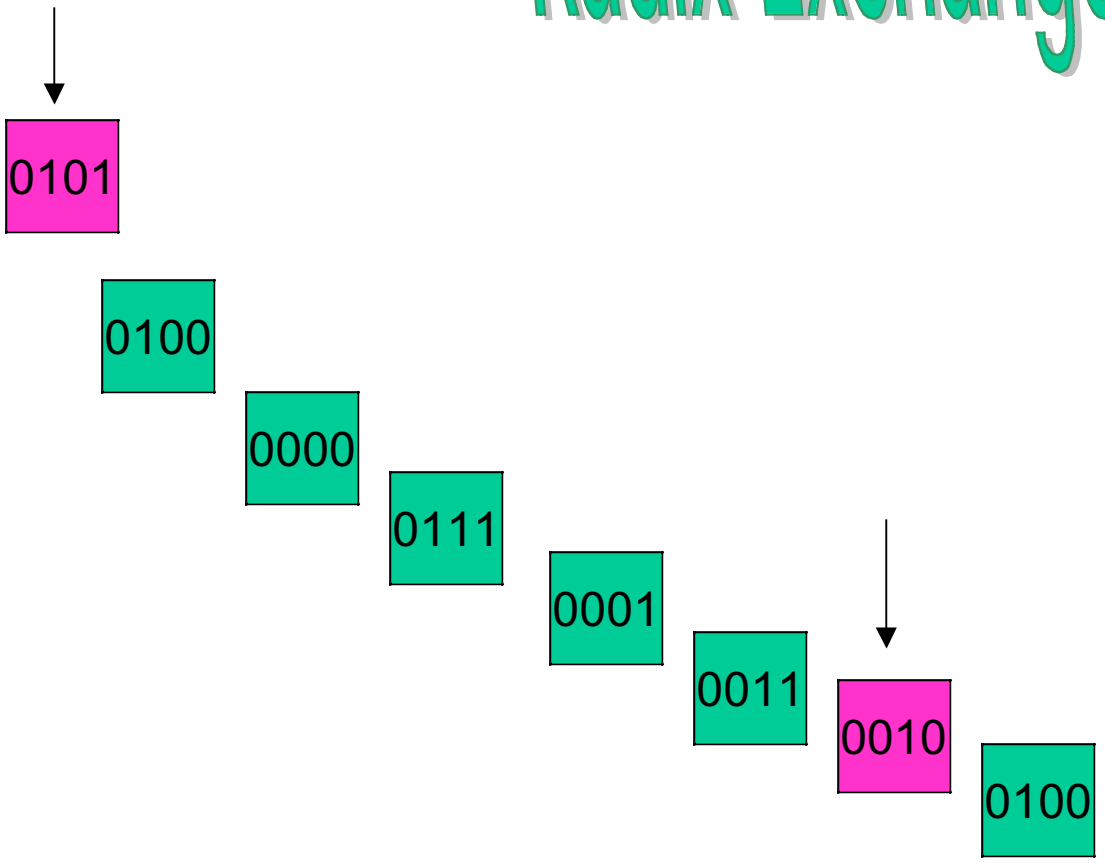
Radix Exchange Sort



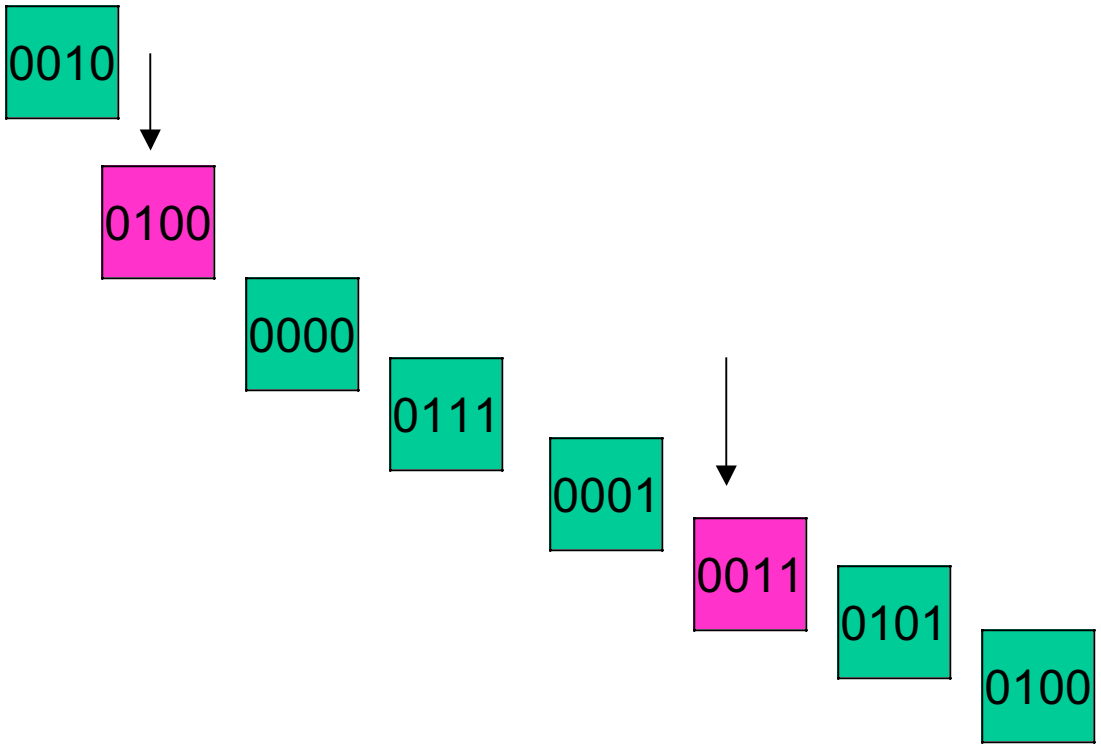
Radix Exchange Sort



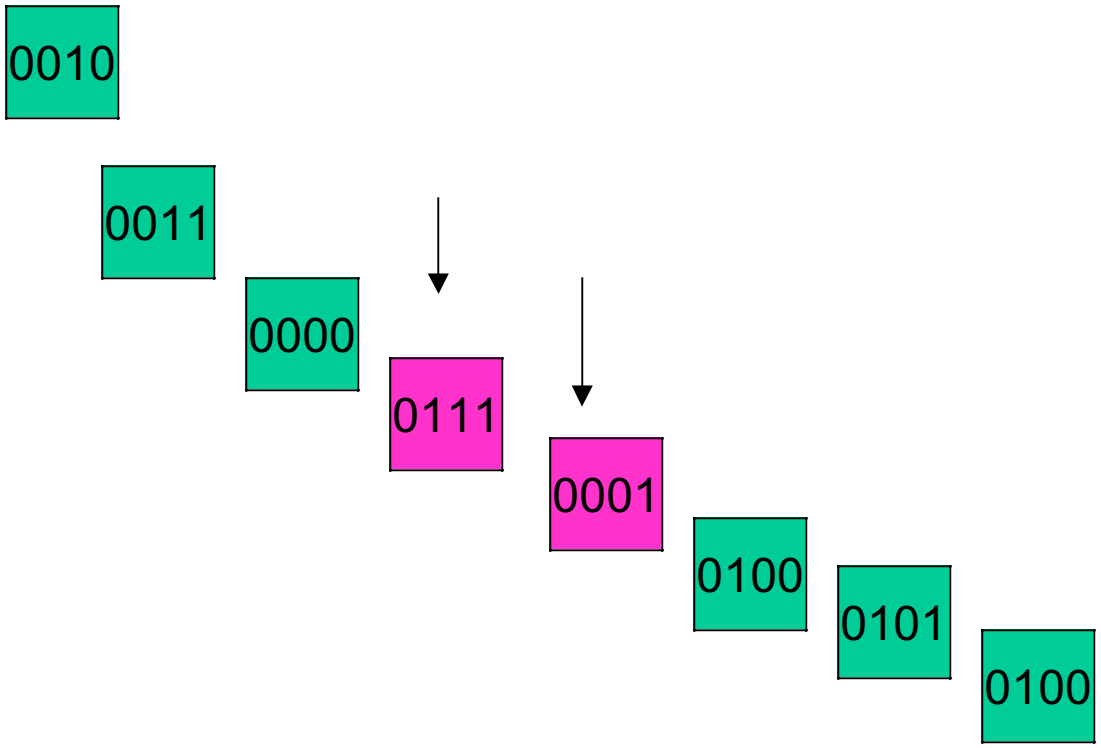
Radix Exchange Sort



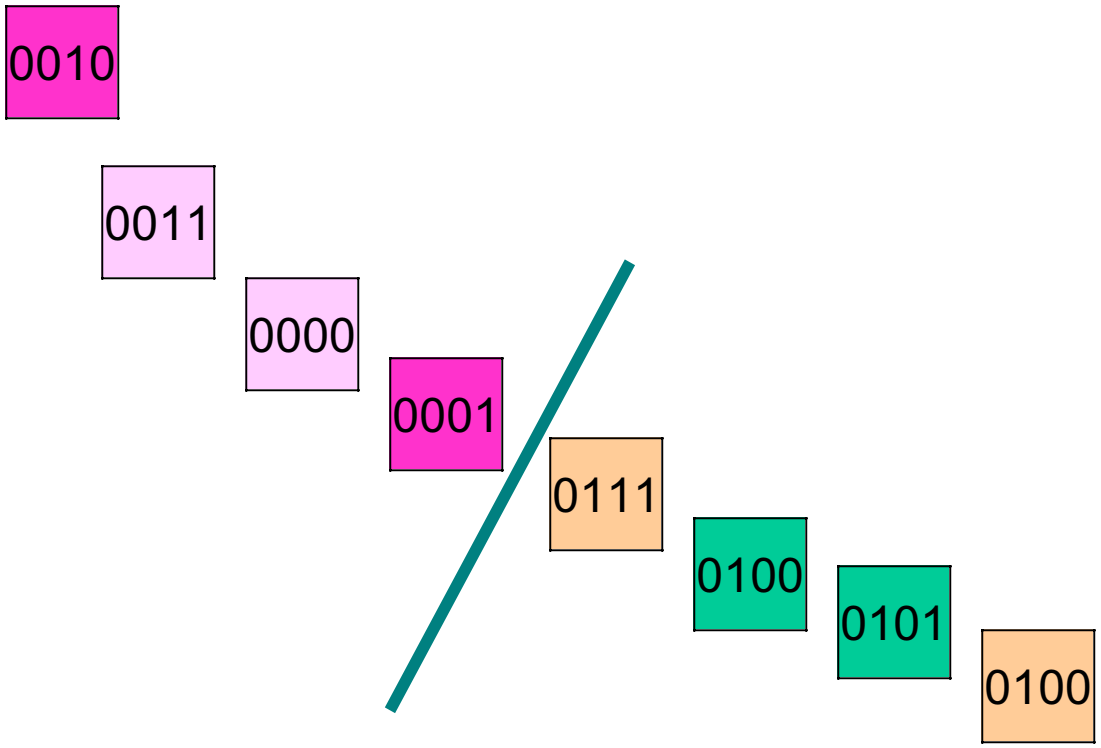
Radix Exchange Sort



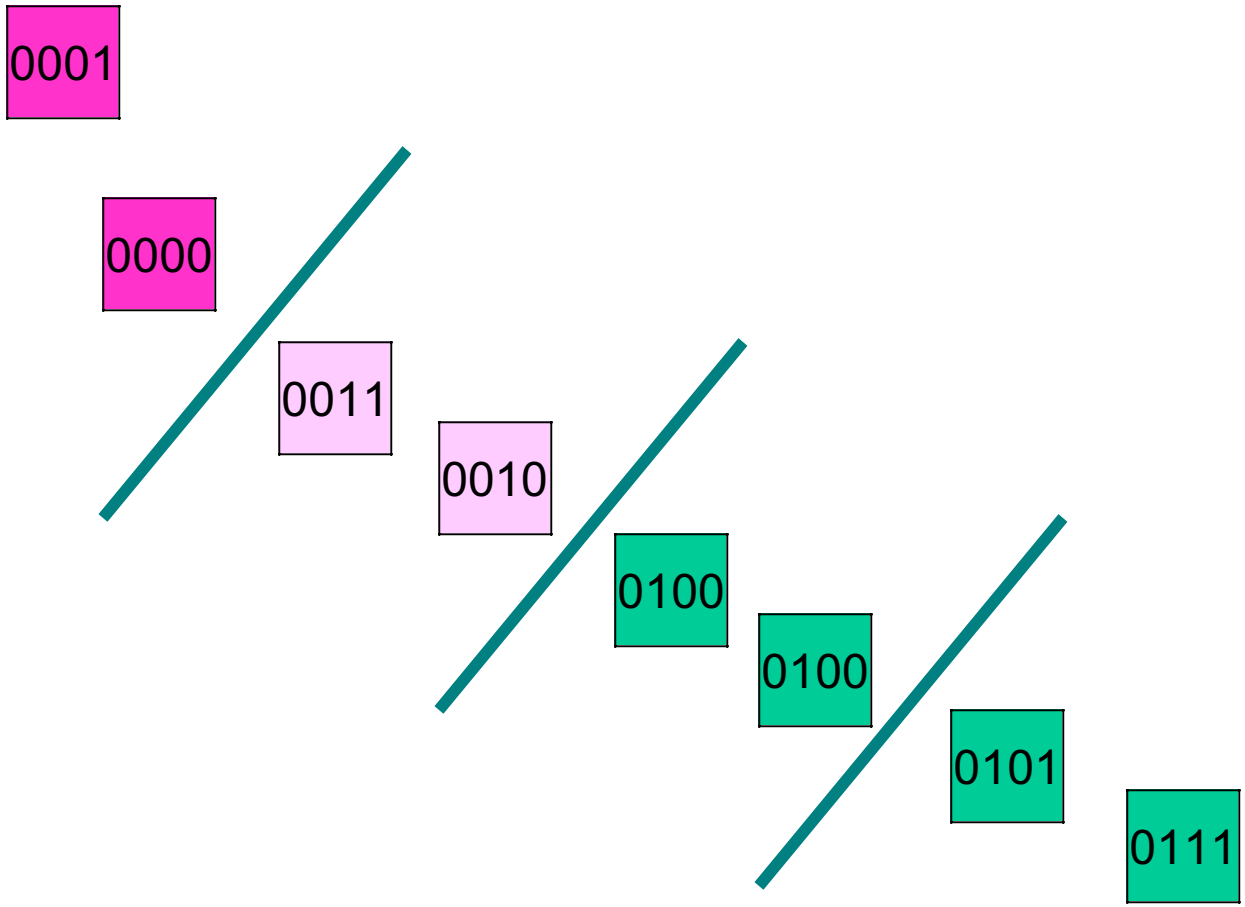
Radix Exchange Sort



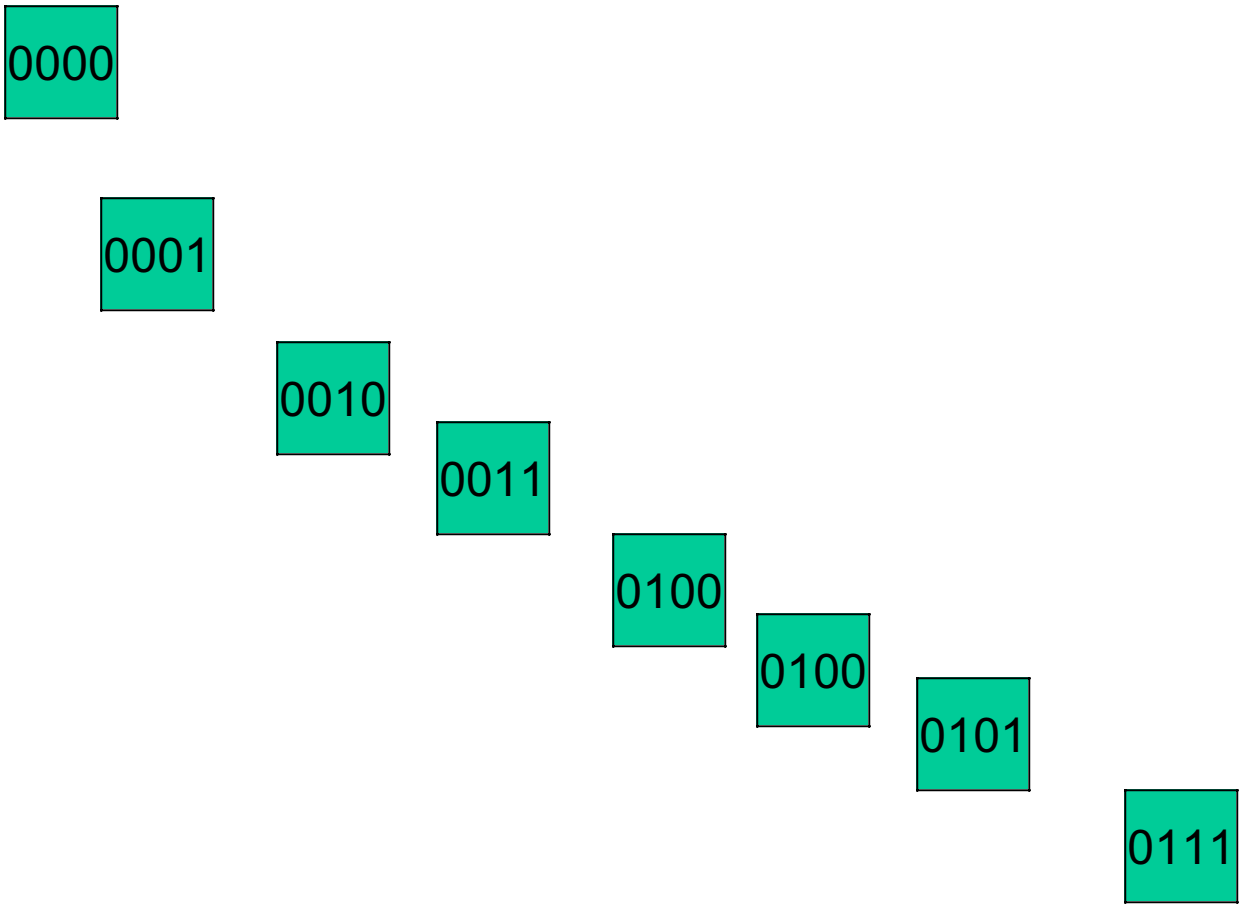
Radix Exchange Sort



Radix Exchange Sort



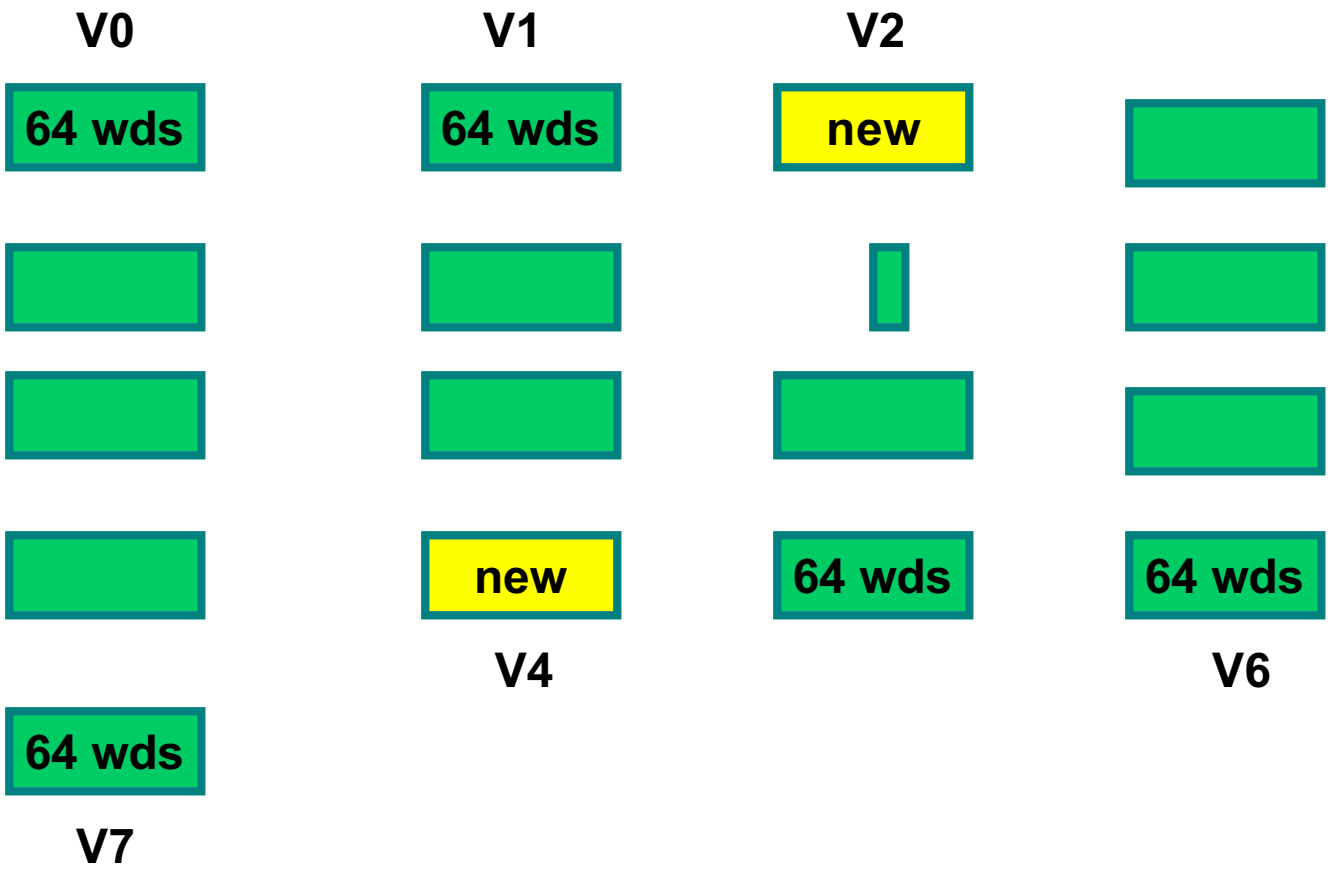
Radix Exchange Sort



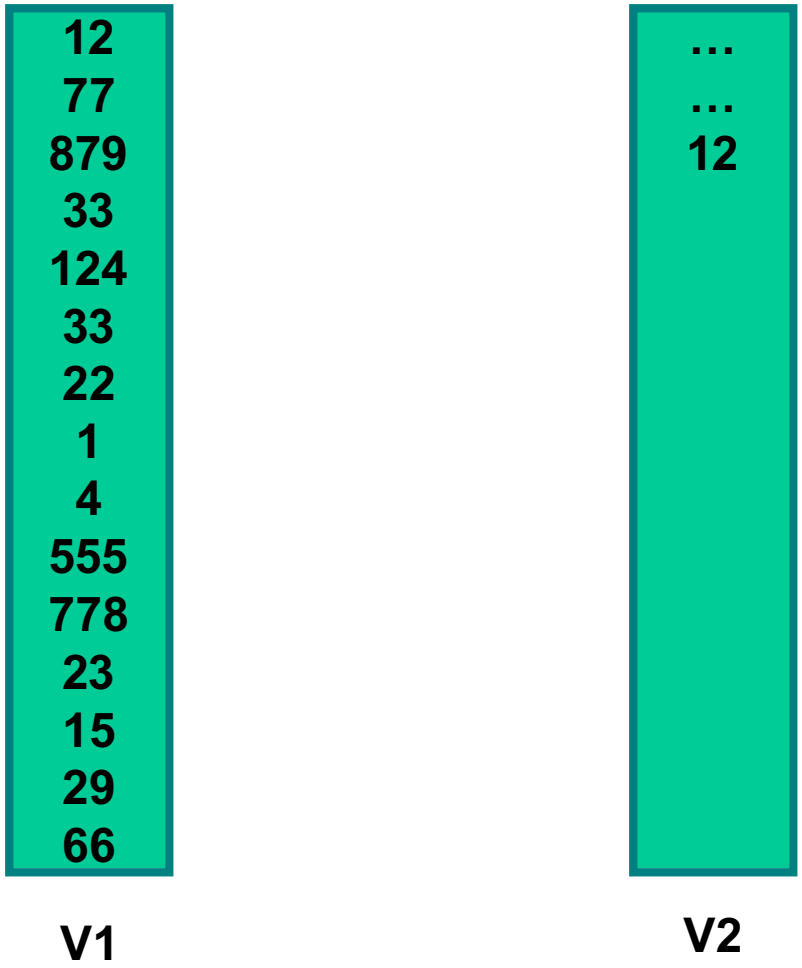
QKSORT on Cray X1

- A version of standard radix sort.
- Basically a scalar code.
- Code has two distinct parts.
- Part 1: Interchange which uses vector registers.
- Part 2 -Short sort in vector registers.
- Two routines in assembly language.

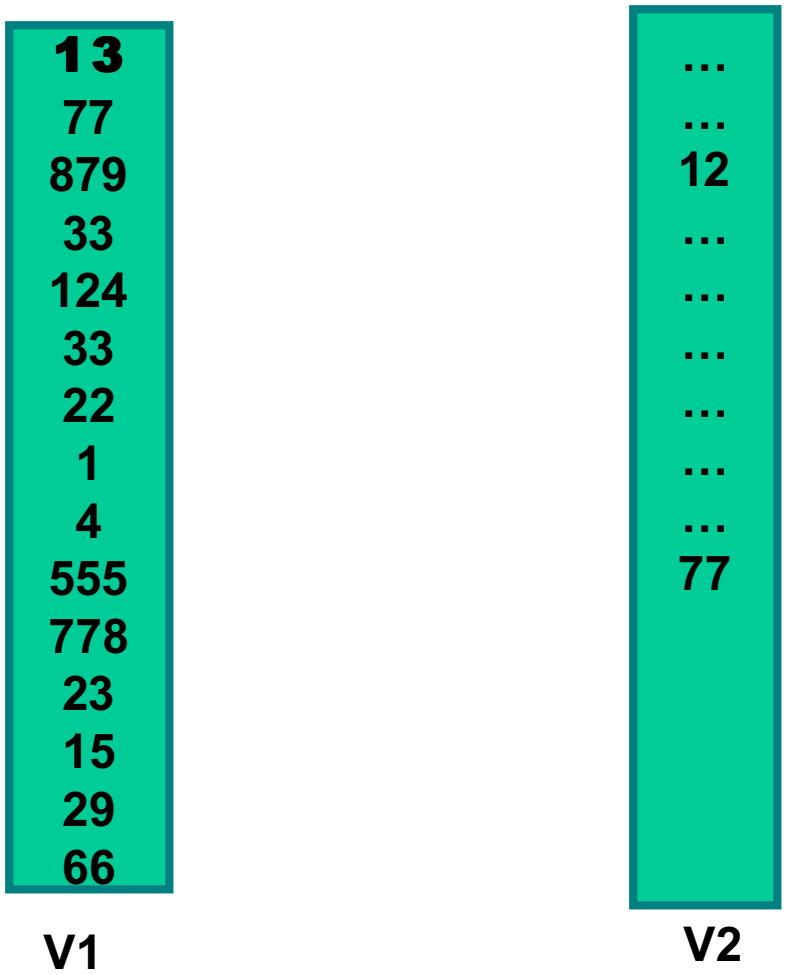
Partitioning in the Vector Registers



Short Sort



Short Sort



Improving QKSORT

Count	5000	10000	50000	100000	500000	1000000	4000000
Version							
Port	.016	.037	.217	.467	2.784	5.987	27.208

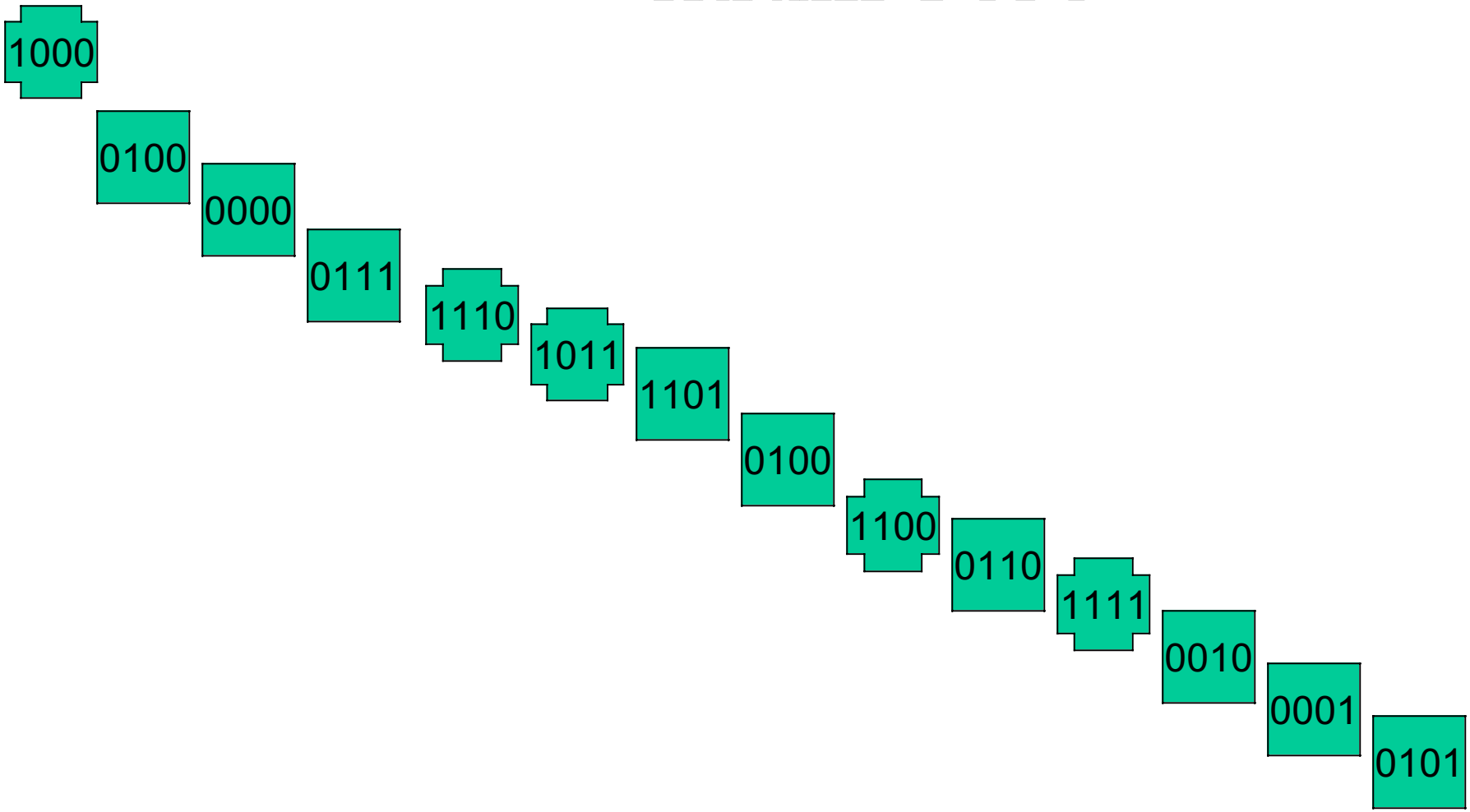
Improving QKSORT

Count	5000	10000	50000	100000	500000	1000000	4000000
Version							
Port	.016	.037	.217	.467	2.784	5.987	27.208
New	.002	.003	.018	. 038	.205	.425	1.824

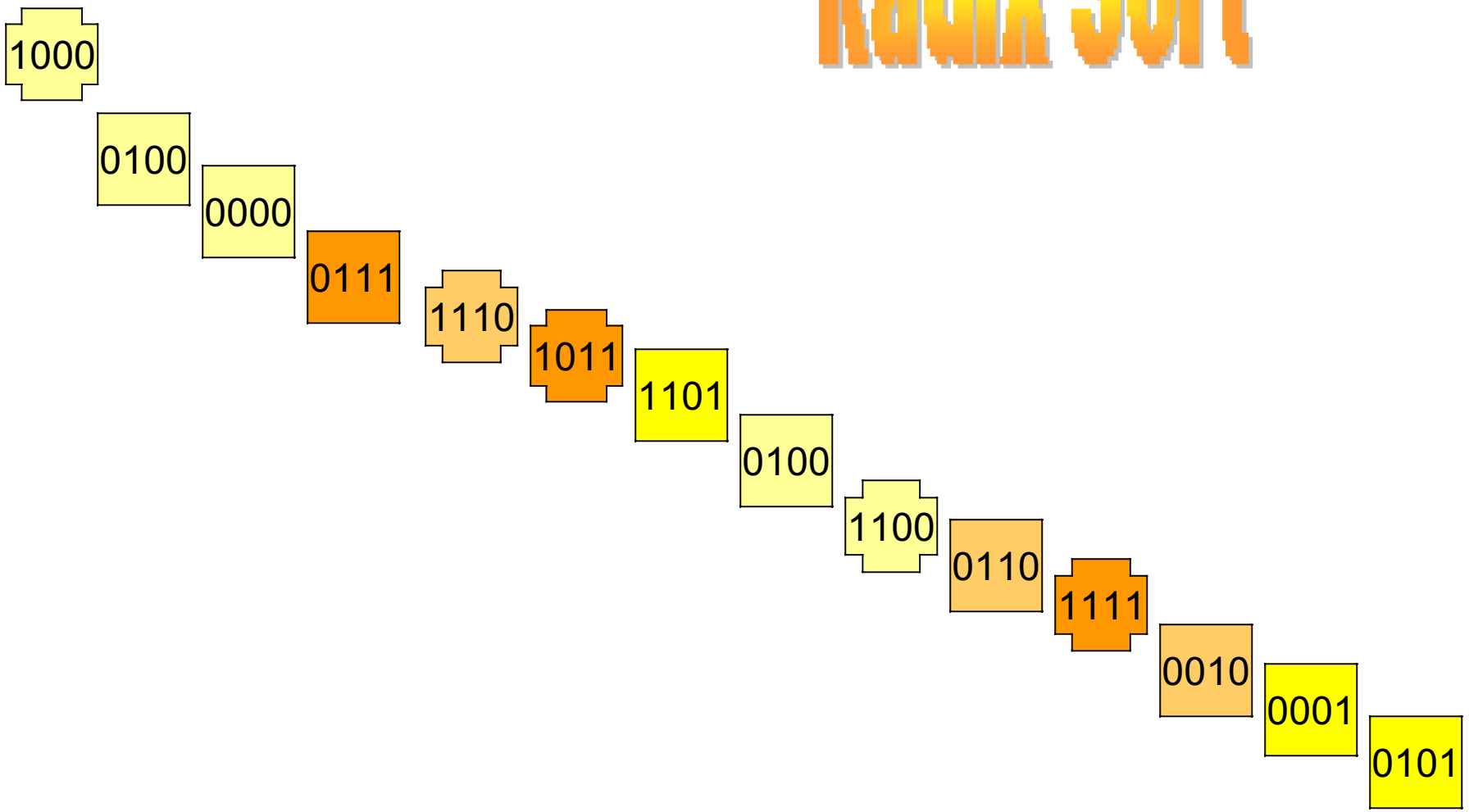
QKSORT in 2003

Size / Machine		10000	50000	100000	500000	1000000	5000000	10000000
T3E		.006	.037	.079	. 510	1.024		
T90		.008	.041	.083	.427	.855	4.367	9.507
Alpha EV6.7		.002	.020	.025	.153	.342	2.242	4.756
Alpha EV6.8		**.001**	**.009**	**.016**	**.085**	**.180**	**1.273**	**2.888**
X1		.003	.018	.038	. 204	.425	2.331	4.805

Radix Sort


1000
0100
0000
0111
1110
1011
1101
0100
1100
0110
1111
0010
0001
0101

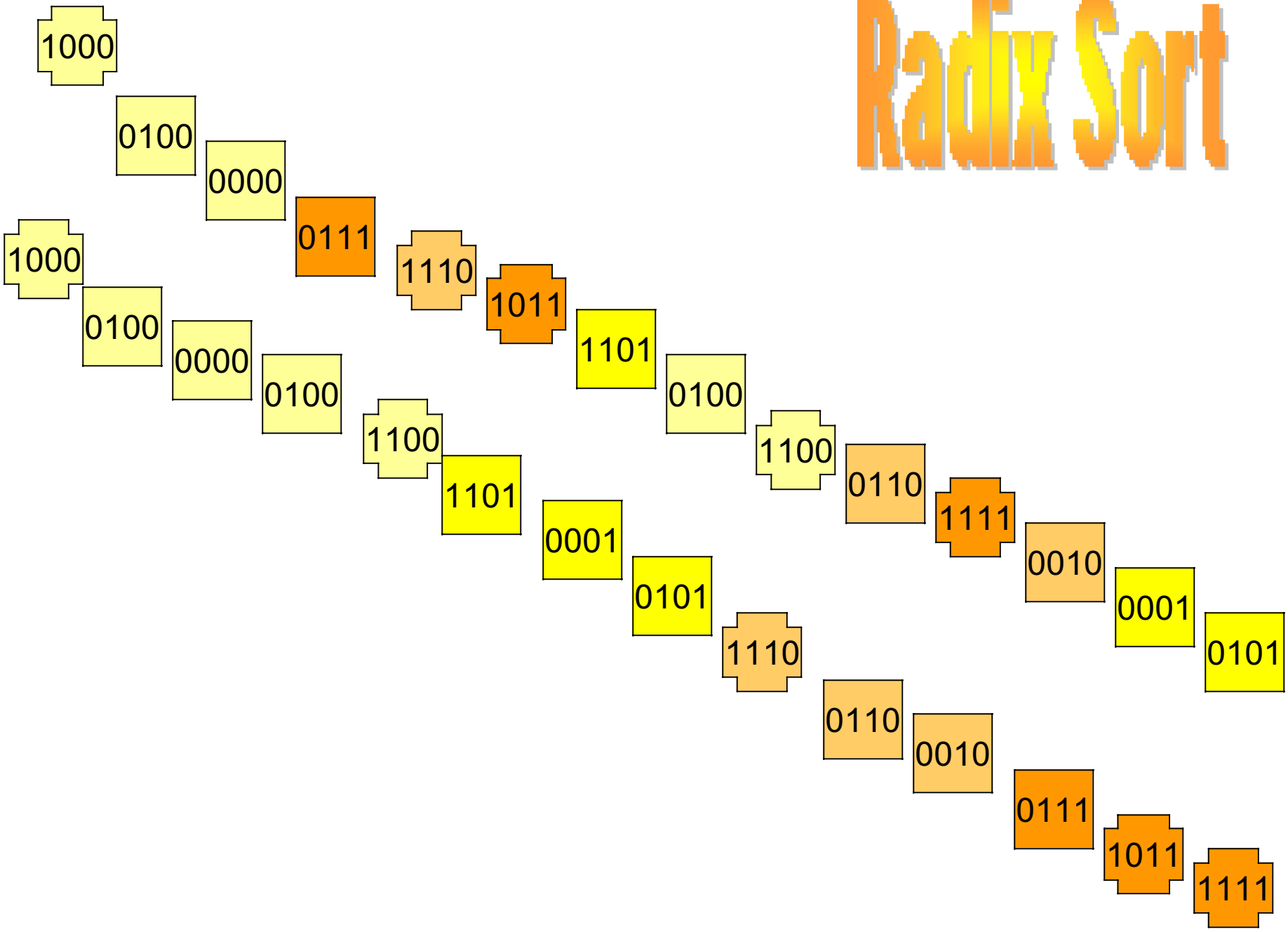
Radix Sort


1000
0100
0000
0111
1110
1011
1101
0100
1100
0110
1111
0010
0001
0101

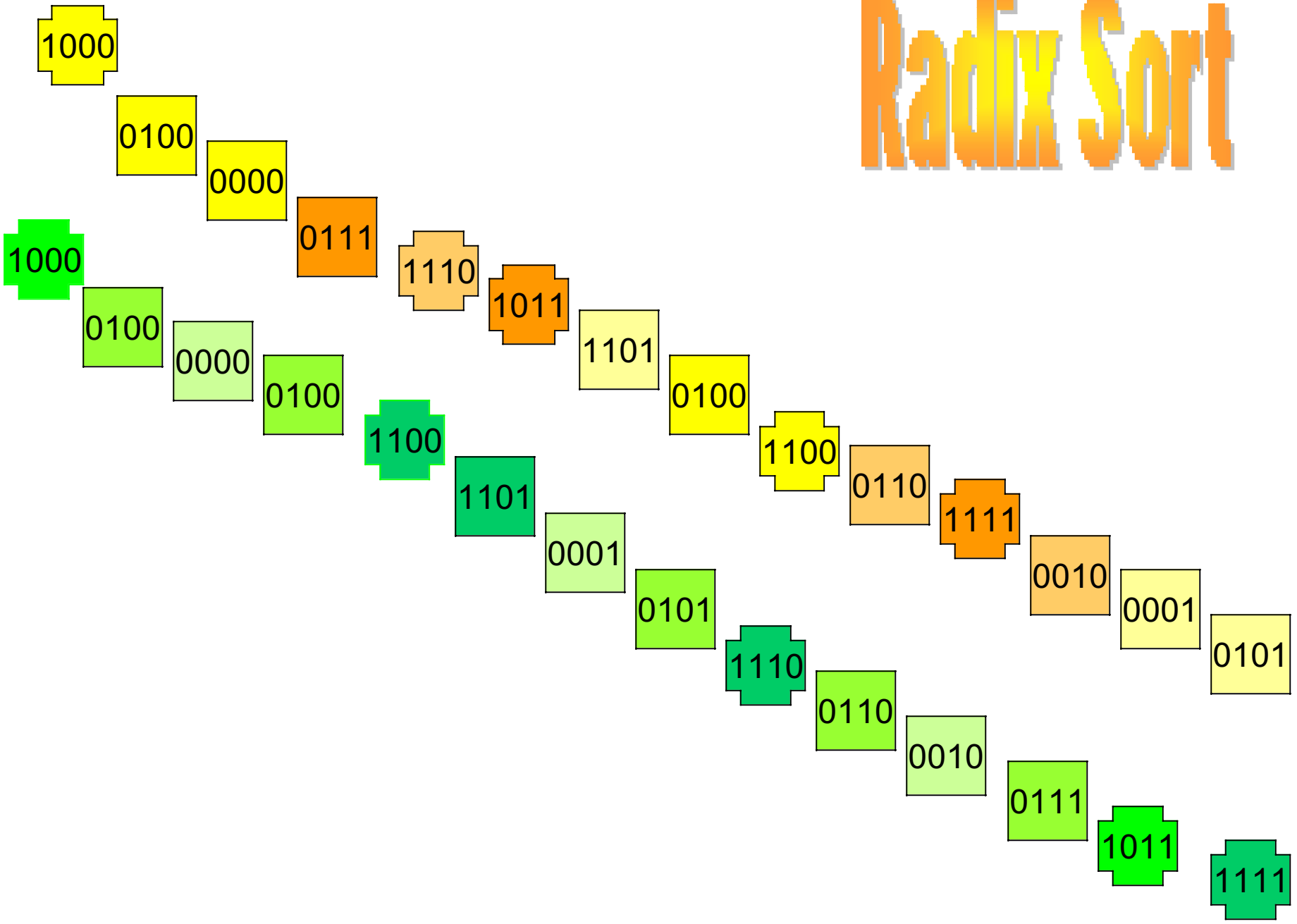
Counts in Right 2 bit pocket

Value	Count	Address
00	5	0
01	3	5
10	3	8
11	3	11

Radix Sort

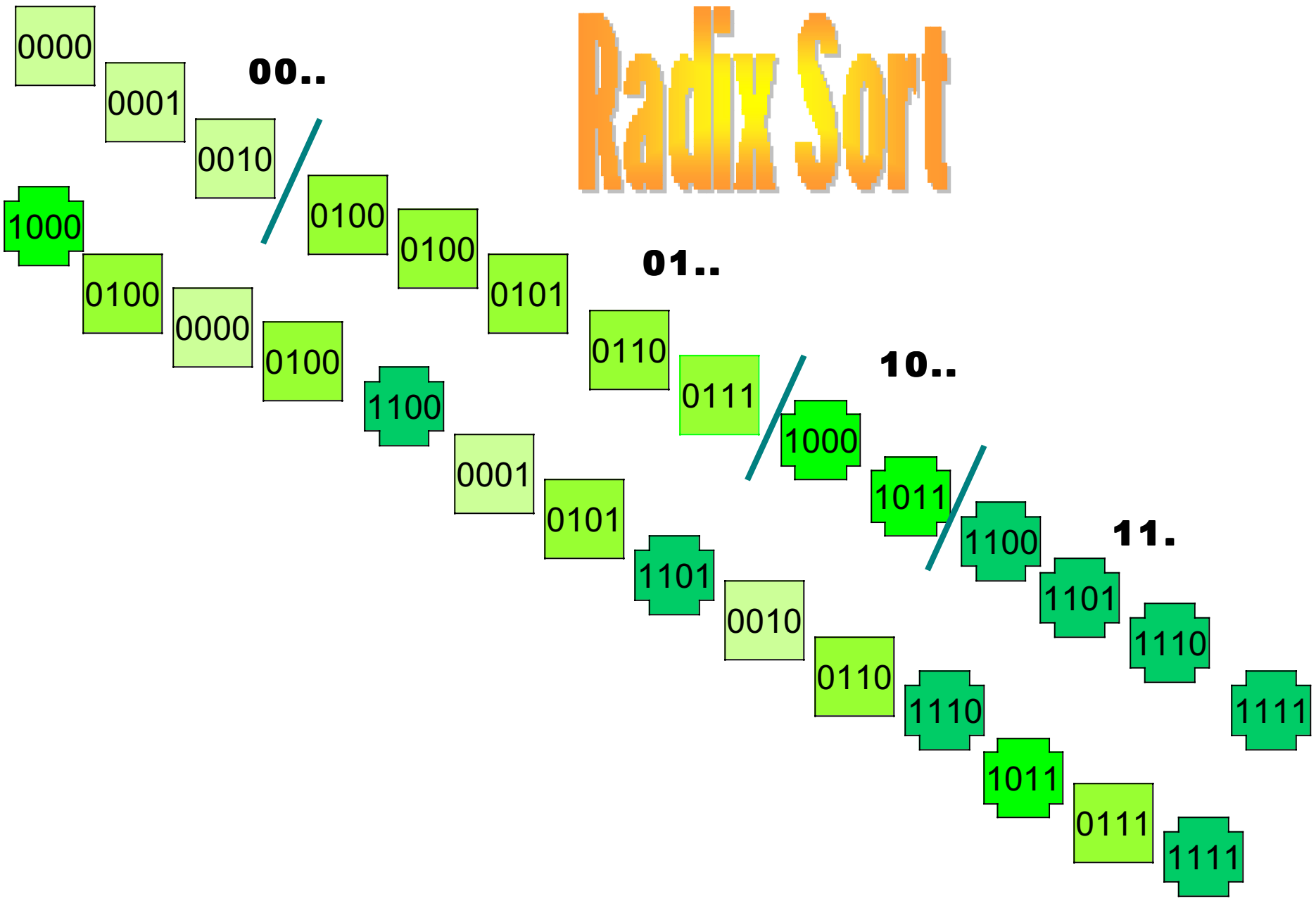

1000
0100
0000
0111
1110
1011
1101
0100
1100
0110
1111
0010
0001
0101
1000
0100
0000
0100
1100
1101
0001
0101
1110
0110
0010
0111
1011
1111

Radix Sort


1000
0100
0000
0111
1110
1011
1101
0100
1100
0110
1111
0010
0001
0101
1000
0100
0000
0100
1100
1101
0001
0101
1110
0110
0010
0111
1011
1111

Counts in left 2 bit pocket

Value	Count	Address
00	3	0
01	5	3
10	2	8
11	4	10


Radix Sort
0000
0001
0010
00..
1000
0100
0000
0100
0100
0100
0101
01..
1100
0001
0101
0110
0111
10..
1000
1011
1100
11.
1101
0010
0110
1101
1110
1110
1011
0111
1111
1111

PSORT on the Cray X1

- A radix sort .
- It requires double memory.
- Three major parts to the code.
- A vector code.
- The address calculation used most time.

  word[j+1] = word[j]+ word[j+1]
- Write address calculation in assembly language.
- Two versions – scalar and 2 vectors

Two vectors Address Update

k=(n-1)/63+1

```
for(j=1;j<k;j++) for(i=0;i<63;i++) word[k*i+j+1] = word[k*i+j+1] +word[k*i+j]

for(i=1;i<63;i++) for(j=1;j<k;j++) word[k*i+j] = word[k*i] + word[k*i+j}
```

Improving PSORT

Count		5000	10000	50000	100000	500000	1000000	4000000
Version								
Ported		.0049	.0047	.0360	.0722	.4142	.4972	1.2252

Improving PSORT

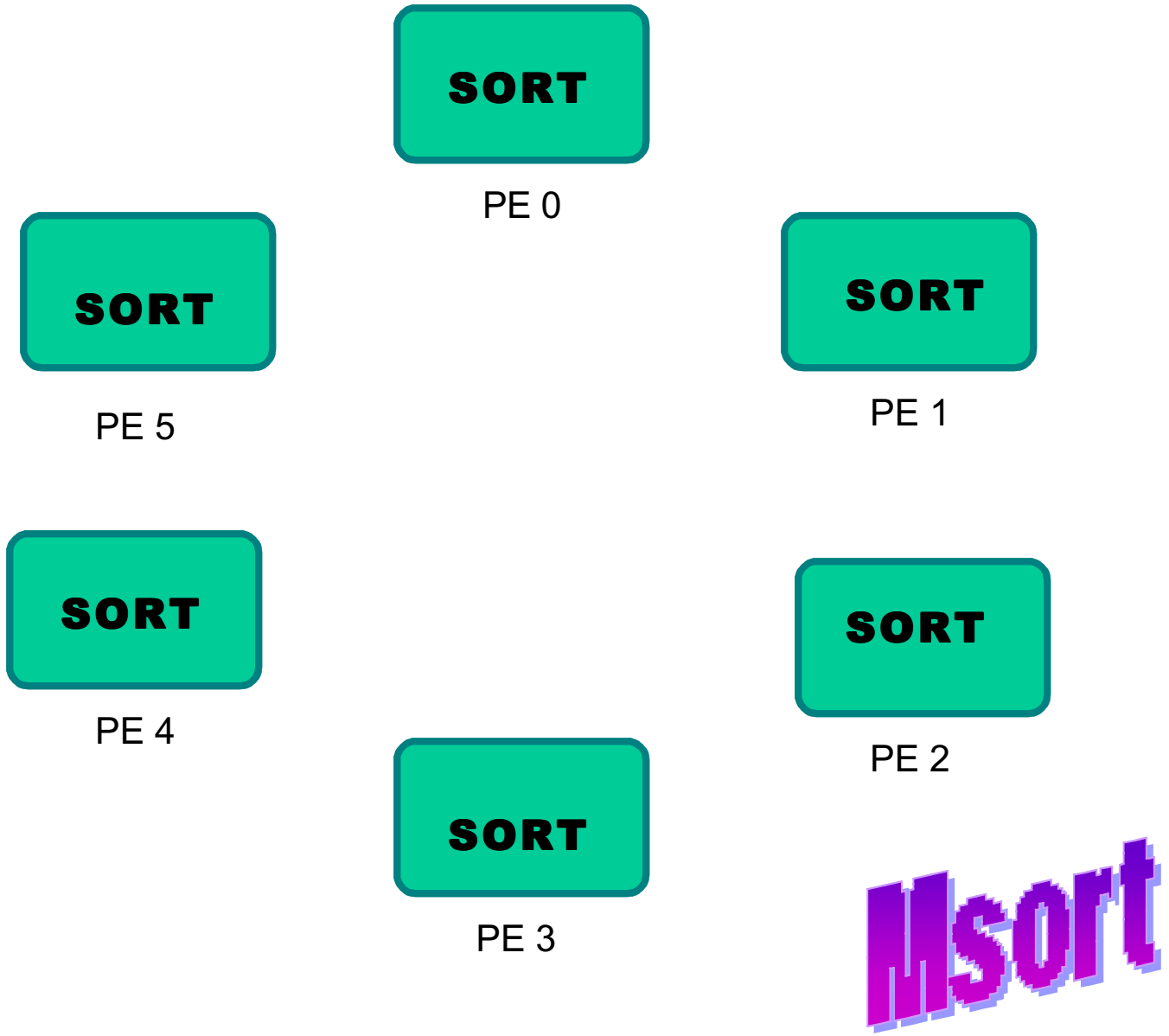
Count		5000	10000	50000	100000	500000	1000000	4000000
Version								
Ported		.0049	.0047	.0360	.0722	.4142	.4972	1.2252
V-S Loop		.0013	.0028	.0159	.0327	.1725	.2484	.7838

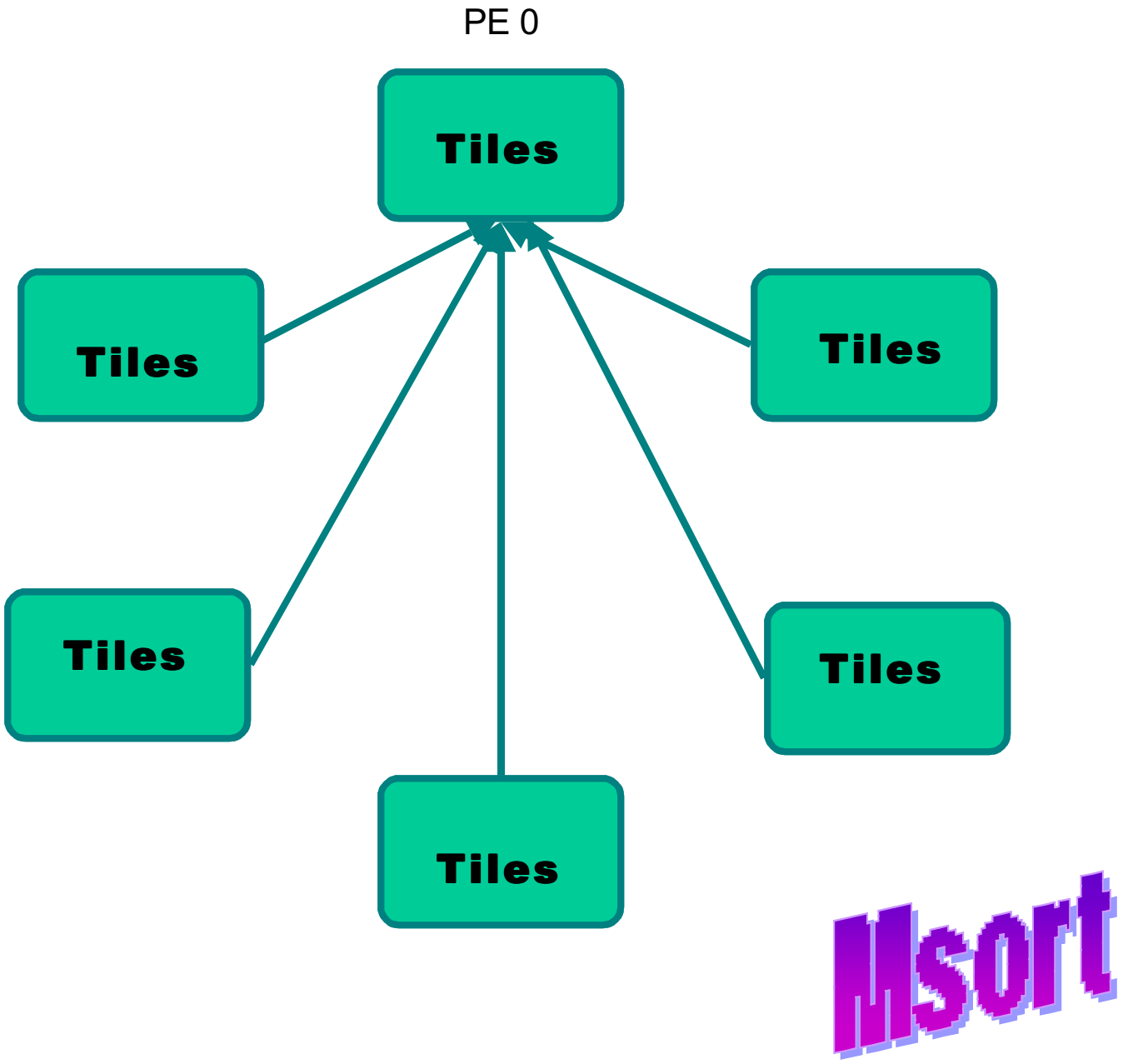
Improving PSORT

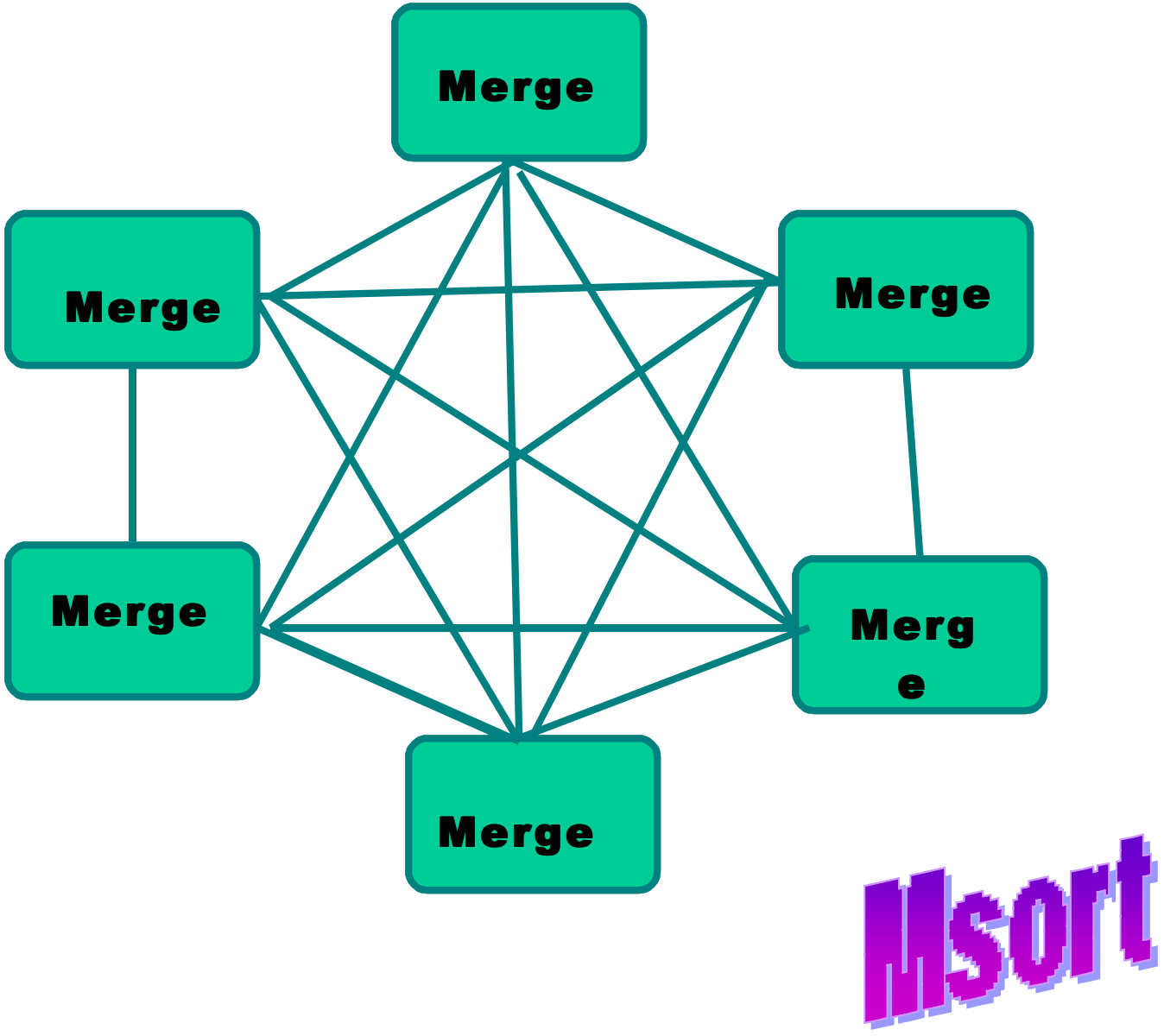
Count		5000	10000	50000	100000	500000	1000000	4000000
Version								
Ported		.0049	.0047	.0360	.0722	.4142	.4972	1.2252
V-S Loop		.0013	.0028	.0159	.0327	.1725	.2484	.7838
2V Loops		.0012	.0027	.0092	.0210	.1273	.2015	.7080

PSORT in 2003

Size Machine		10000	50000	100000	500000	1000000	5000000	10000000
T3E		.037	.377	.747	.3.307	6.371		
T90		**.003**	**.017**	**.019**	**.090**	**.183**	**.702**	**1.31**
Alpha EV6.7		.004	.040	.120	1.490	3.016	12.801	27.604
Alpha EV6.8		.003	.023	.070	.394	.934	7.636	16.536
X1		.003	.009	.021	.127	.202	.866	1.684


SORT
PE 0
SORT
PE 5
SORT
PE 1
SORT
PE 4
SORT
PE 2
SORT
PE 3
Nsort


PE 0
Tiles
Tiles
Tiles
Tiles
Tiles
Tiles
Nsort


Merge
Merge
Merge
Merge
Merg
e
Merge
Msort

Improving MSORT

- Slow time due to SHMEM.
- May be due to barrier code.
- PSORT is faster than QKSORT.
- We need double block of memory for merge.
- However, times are similar since most of time is spent in moving data.

MSORT in 2003

Size	10,000	100,000	1,000,000	10,000,000
T3E - 2	.003	.040	.488	
T3E - 4	**.002**	**.023**	.263	
T3E - 8	.002	.025	.142	1.661
T3E - 16	.004	.068	.099	1.098
T3E - 24	.009	.085	**.074**	**.755**
X1 - 2	**.013**	.091	.955	
X1 - 4	.014	.064	.650	
X1 - 8	.033	.050	.492	4.011
X1 - 16	.042	.048	.348	2.292
X1 - 24	.109	**.026**	**.191**	**1.674**

What next?

- Can we think of a way to use multistreaming?
- If so – should we do it?
- Algorithms using SPP mode.
- Suggest compiler improvements to Cray.

What was the Point?

- Early exercise on the X1.

What was the Point?

- Early exercise on the X1.
- Produce good sorts for the X1.

What was the Point?

- Early exercise on the X1.
- Produce good sorts for the X1.
- Found compiler weaknesses.

What was the Point?

- Early exercise on the X1.
- Produce good sorts for the X1.
- Found compiler weaknesses.
- Found machine strengths.