ZPL

- No explicit threads
  - no operation like **`omp_get_thread_num`** or **`Ti.thisProc`**
- Only way to use an array is in a parallel operation

```
a : [0..N] integer;
sum : integer;

[0..N] a := a + 1;
[0..N] sum := +<<a;
```

Sortof like...

... OpenMP, in that the programmer declares places for automatic parallelism

ZPL “declarations” are much more fine grained, with more kinds of operators

... Titanium, in that communication is implicit through shared objects

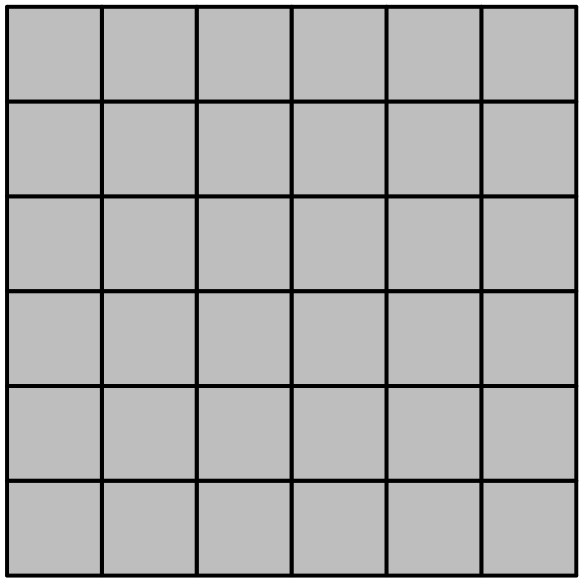
ZPL automatically localizes data and has a different way of describing costs

... APL, in that good programs need to use the right operators

ZPL has fewer operators targeted just as parallelism

Regions

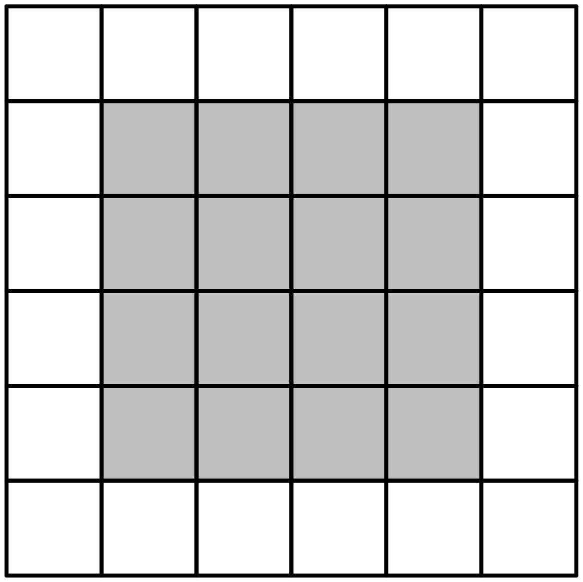
```
region
 R = [1..n,1..n];
```



The following slides are based on the ZPL “comic”

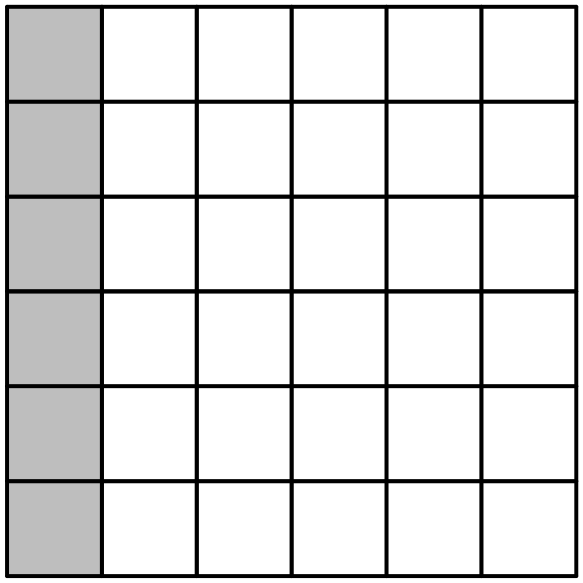
Regions

```
region
 IntR = [2..n-1,2..n-1];
```



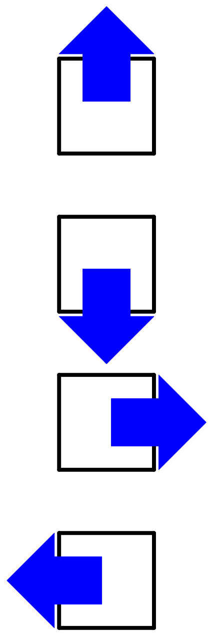
Regions

```
region
 Left = [1..n,1];
```

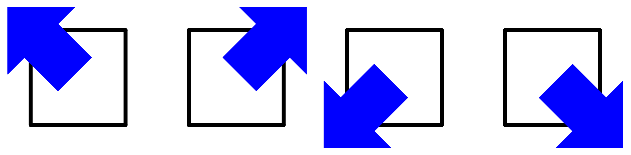


Directions

```
direction
  north = [-1, 0];
  south = [ 1, 0];
  east  = [ 0, 1];
  west  = [ 0,-1];
```

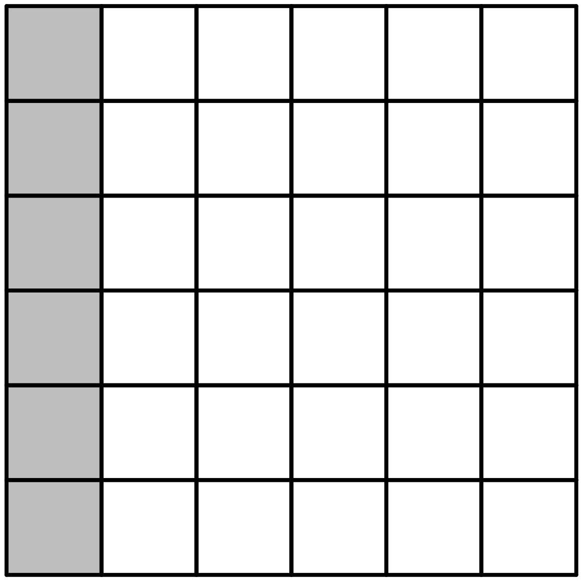


```
  nw = [-1,-1];
  ne = [-1, 1];
  sw = [ 1,-1];
  se = [ 1, 1];
```



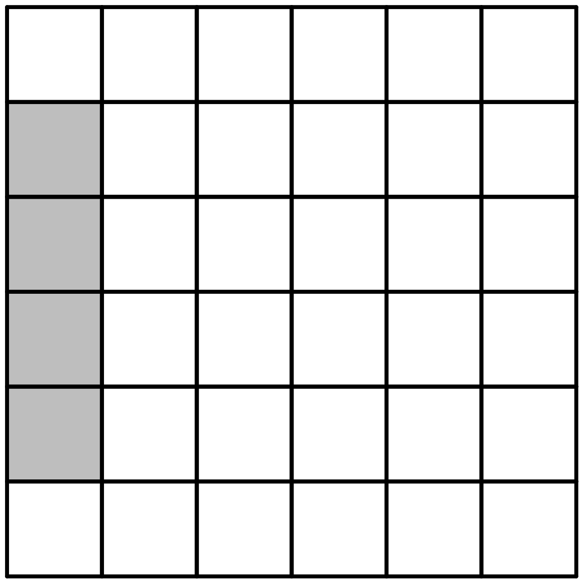
Region Operators

```
region
 Left = west in R;
```



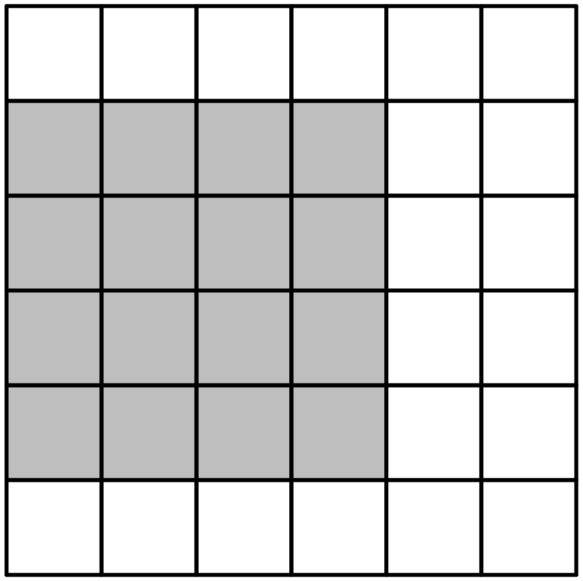
Region Operators

```
region
 SmallLeft = west of IntR;
```



Region Operators

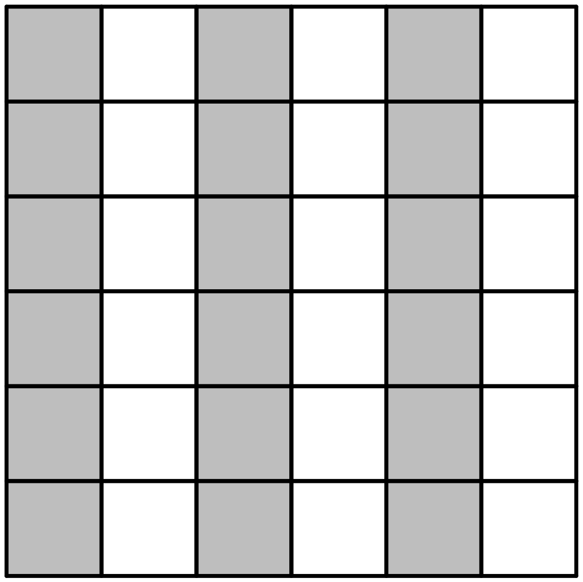
```
region
 IntRLeft = IntR at west;
```



Region Operators

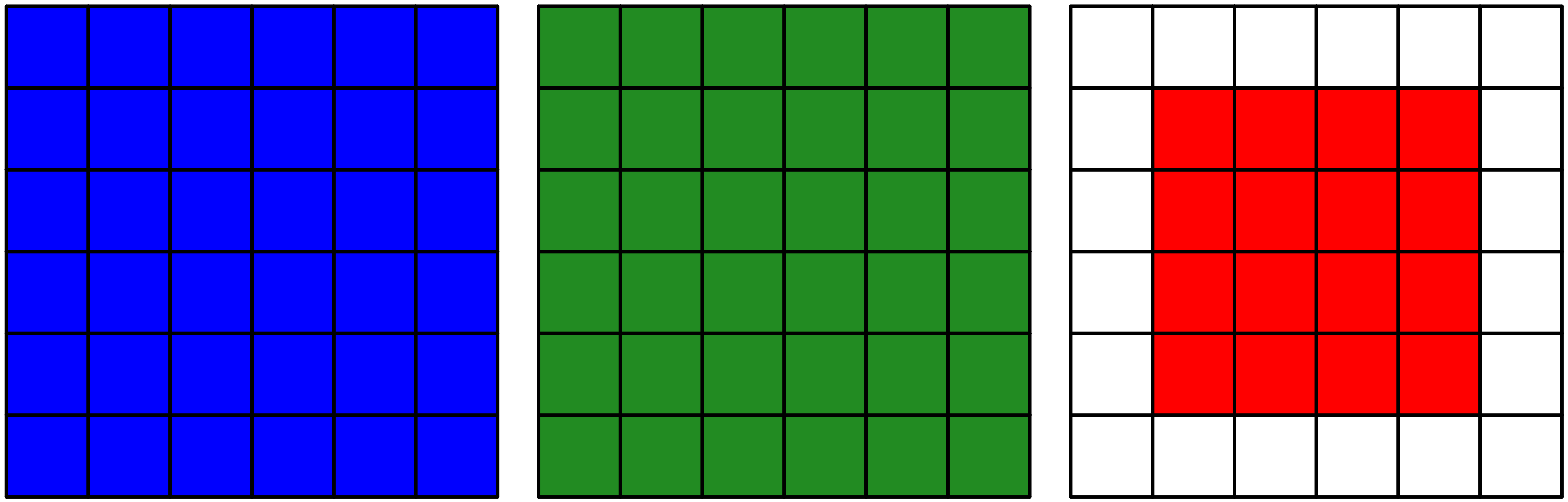
```
direction
  step = [1,2];

region
  SR = R by step;
```



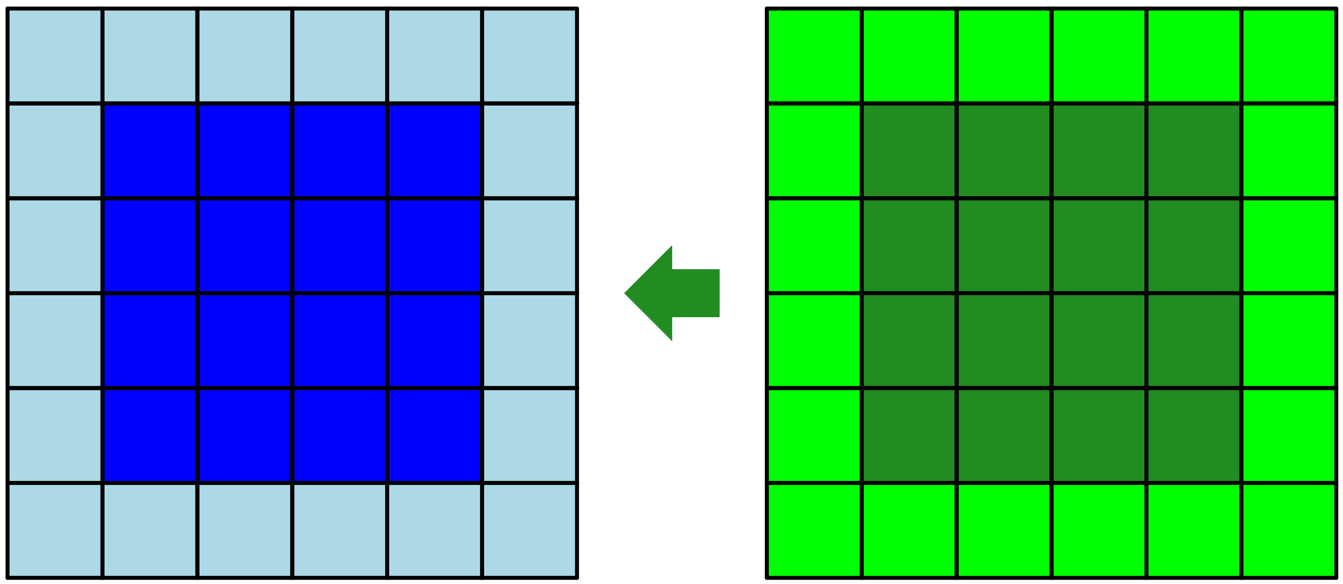
Declaring Arrays

```
var
  A, B : [R] double;
  C : [IntR] double;
```



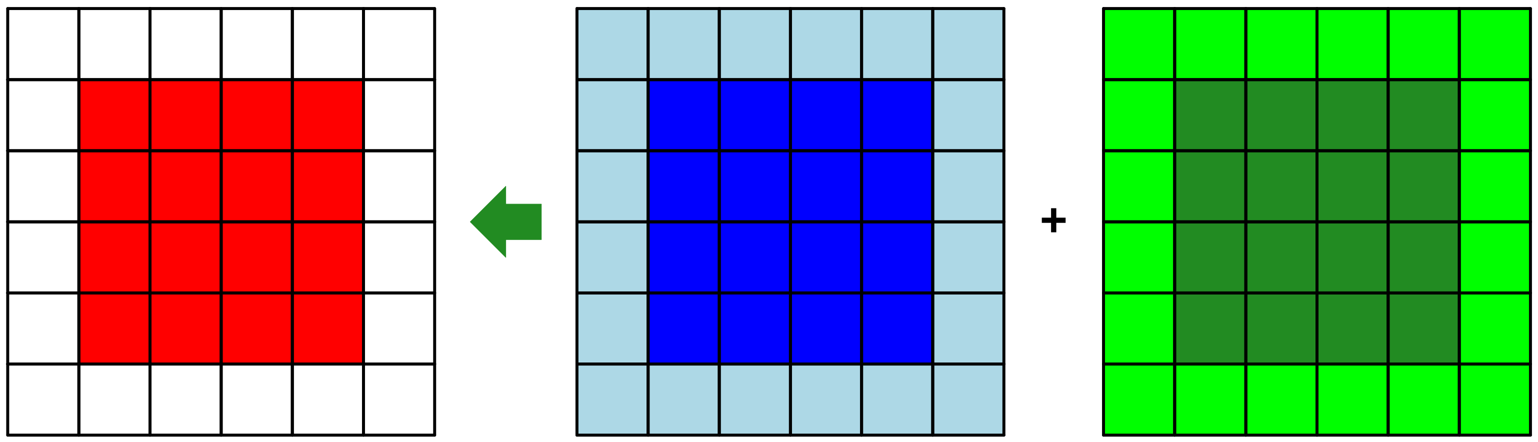
Regions Control Statements

```
[IntR] A := B;
```



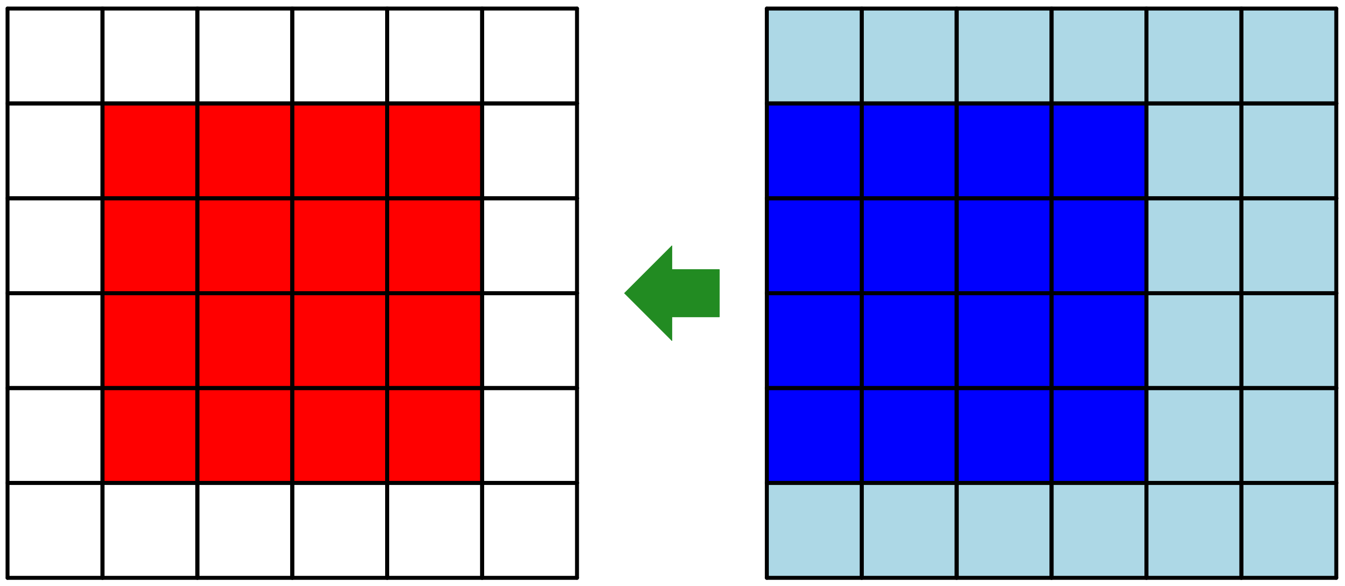
Regions Control Statements

```
[IntR] C := A + B;
```



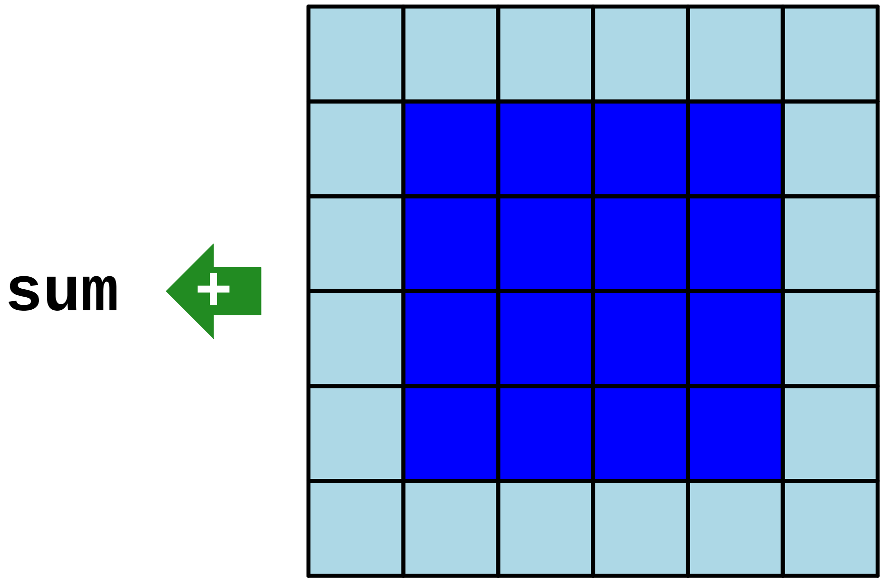
Regions Control Statements

```
[IntR] C := A@west;
```



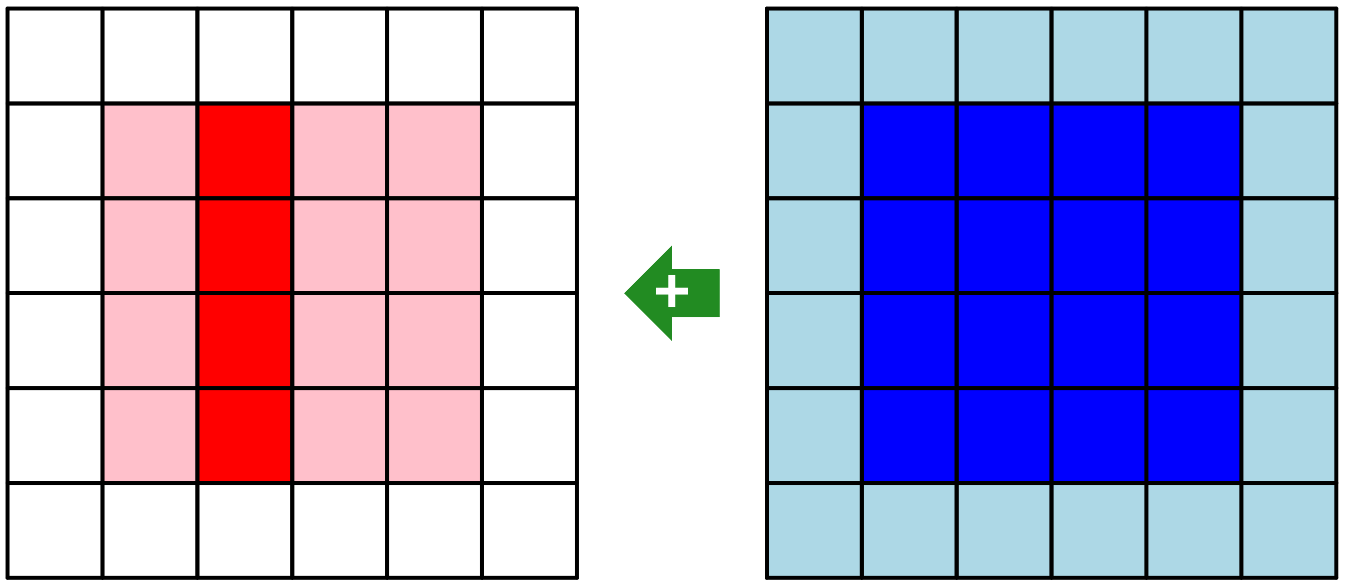
Reduction

```
[IntR] sum := +<< A;
```



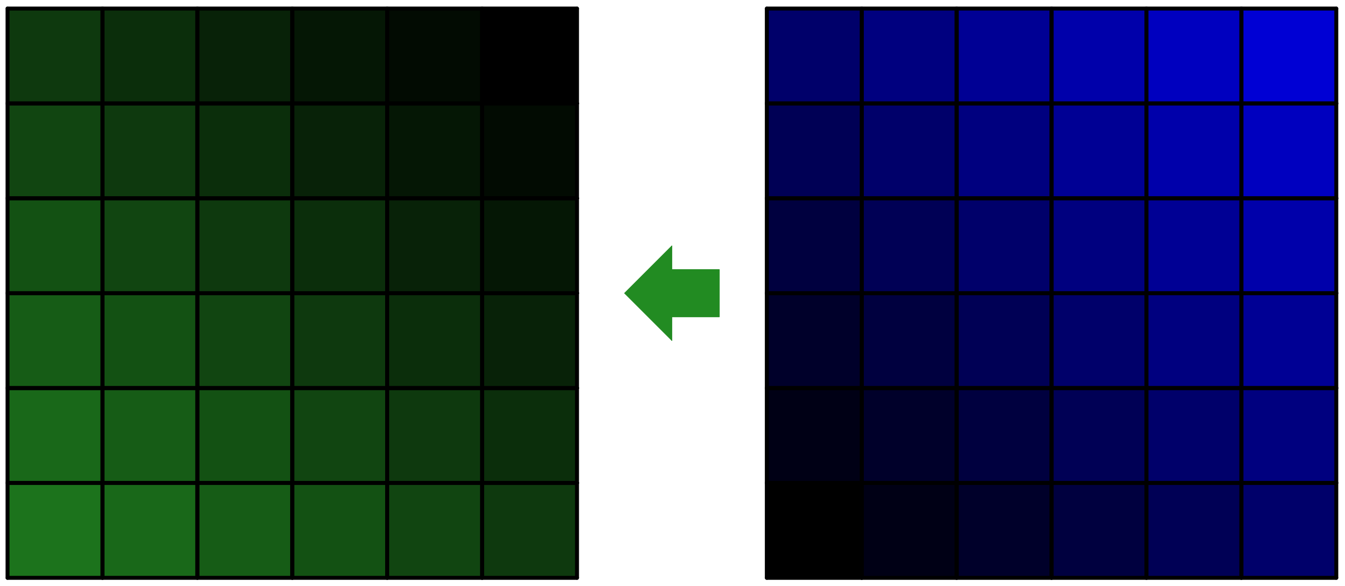
Partial Reduction

```
[2..n-1,i] C := +<<[IntR] A;
```

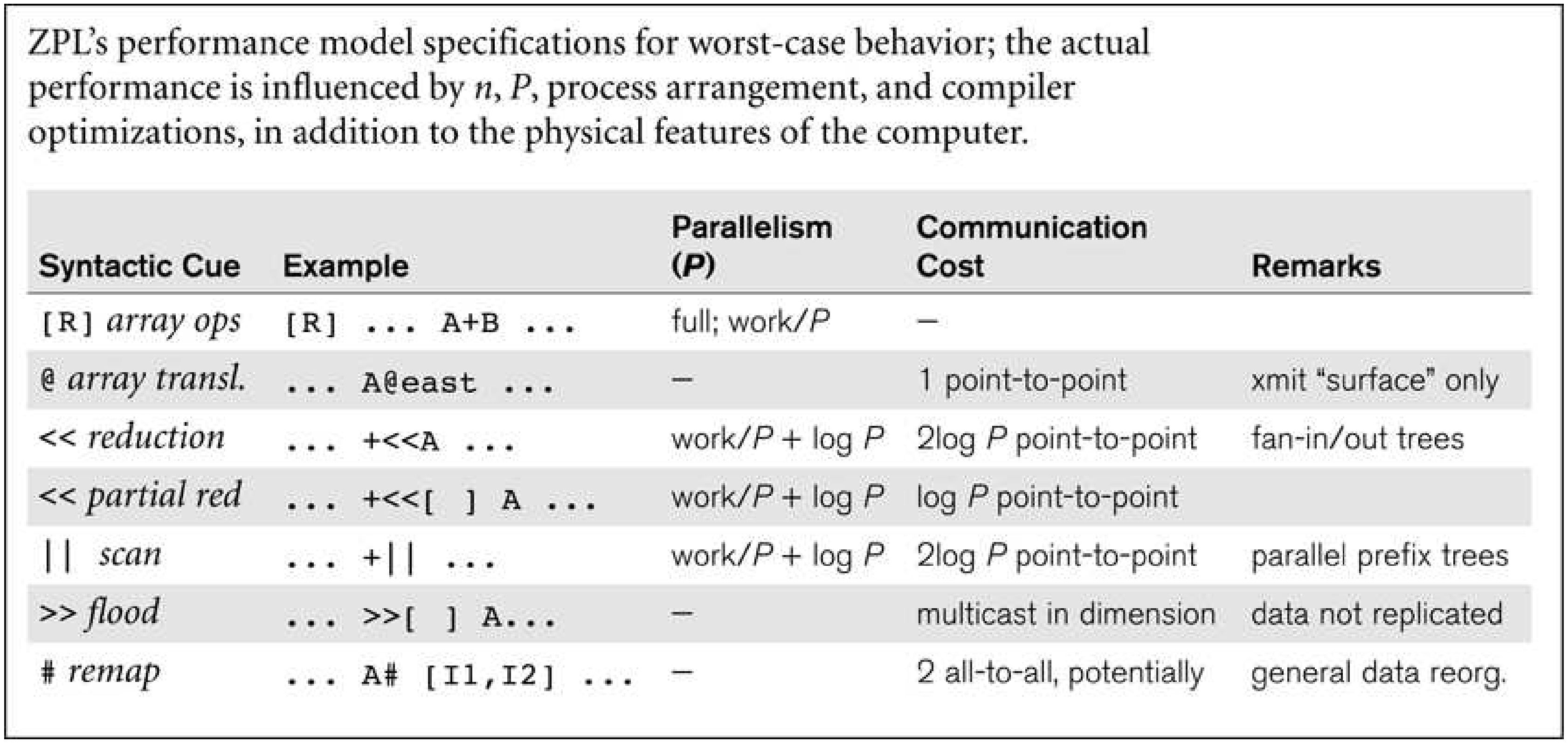


Remap

```
[R] B := A#[Index2, Index1]
```



Cost Model



ZPL's performance model specifications for worst-case behavior; the actual performance is influenced by *n*, *P*, process arrangement, and compiler optimizations, in addition to the physical features of the computer.

Syntactic Cue	Example	Parallelism (*P*)	Communication Cost	Remarks
`[R]` *array ops*	`[R] ... A+B ...`	full; work/*P*	–	
`@` *array transl.*	`... A@east ...`	–	1 point-to-point	xmit "surface" only
`<<` *reduction*	`... +<<A ...`	work/*P* + log *P*	2log *P* point-to-point	fan-in/out trees
`<<` *partial red*	`... +<<[ ] A ...`	work/*P* + log *P*	log *P* point-to-point	
`||` *scan*	`... +|| ...`	work/*P* + log *P*	2log *P* point-to-point	parallel prefix trees
`>>` *flood*	`... >>[ ] A...`	–	multicast in dimension	data not replicated
`#` *remap*	`... A# [I1,I2] ...`	–	2 all-to-all, potentially	general data reorg.