Gauss-Seidel

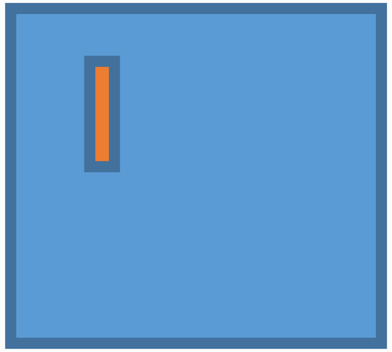
Dealing with Complicated Dependencies

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Gauss-Seidel Relaxation

Sequential pseudocode:

```
while (maxError > Threshold) {
  Re-apply Boundary conditions
  maxError = 0;
  for i = 0 to N-1 {
    for j = 0 to N-1 {
      old = A[i, j]
      A[i, j] = 0.2 * (A[i,j] + A[i,j-1] +A[i,j+1]
                     + A[i+1,j] + A[i-1,j]) ;
      if (|A[i,j]-old| > maxError)
        maxError = |A[i,j]-old|
    }
  }
}
```



For the same problem we solved using Jacobi Relaxation

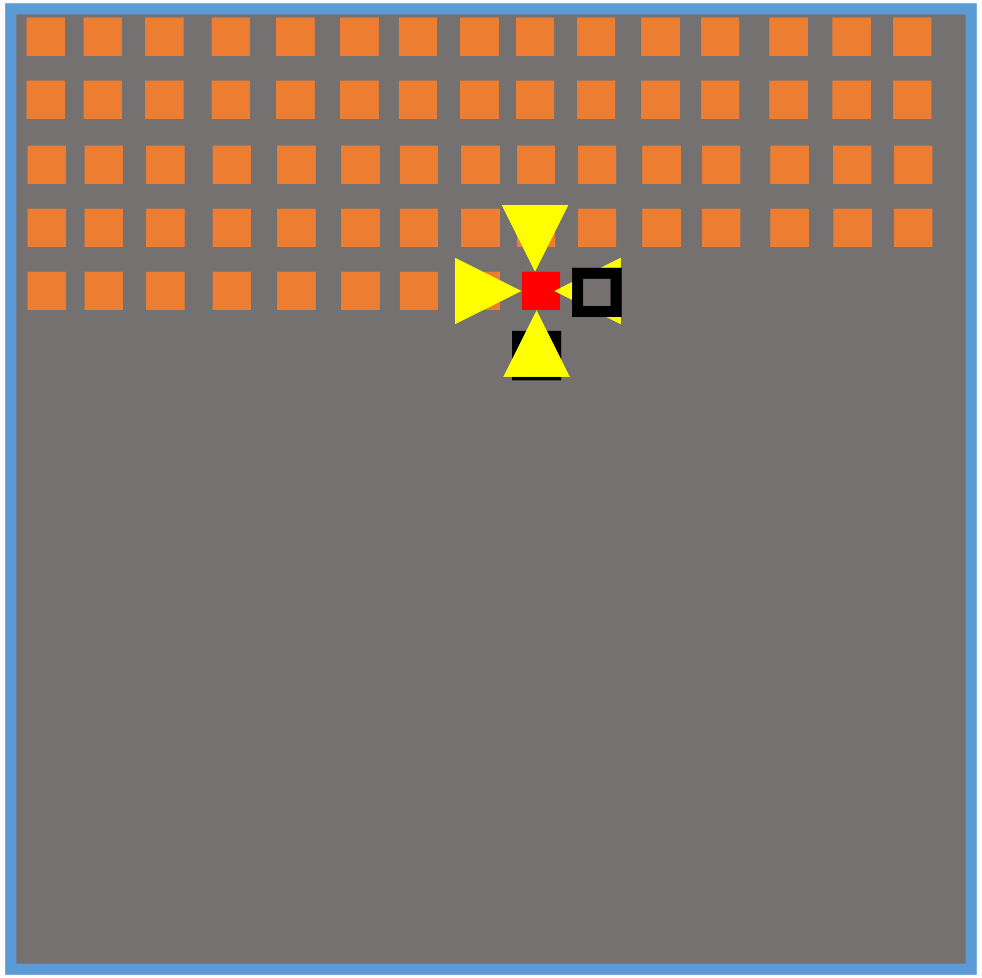
No old-new arrays ...

Sequentially, how well does this work?

It works much better!

- Intuitively, the effect of boundary conditions spreads fast to other areas, compared with Gauss-Jacobi

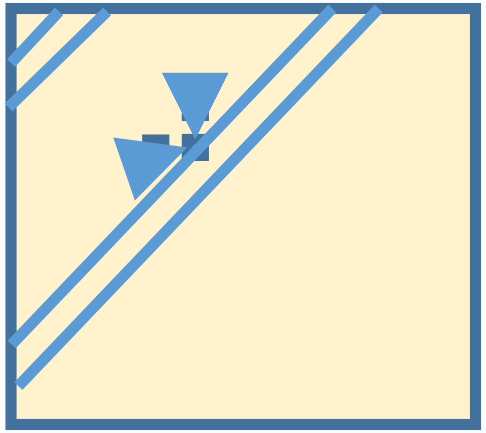
How to parallelize this?



```
A[i, j] = 0.2 * (A[i,j] + A[i,j-1] + A[i,j+1]
                            + A[i+1,j] + A[i-1,j]) ;
```

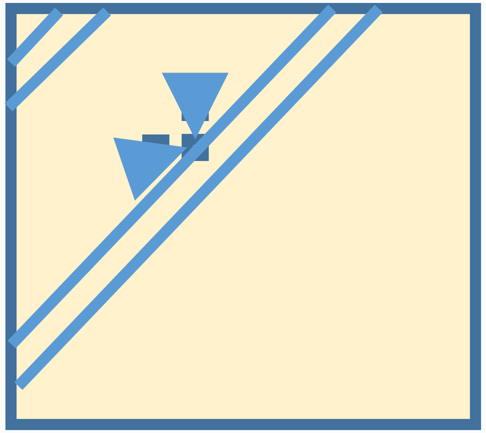
How Do We Parallelize Gauss-Seidel?

- Visualize the flow of values
- Not the control flow:
  - That goes row-by-row
- Flow of dependences: which values depend on which values?
- Does that give us a clue on how to parallelize?



How Do We Parallelize Gauss-Seidel?

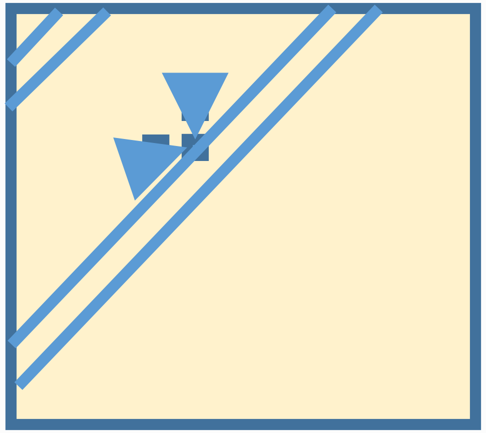
- Visualize the flow of values
- Not the control flow:
  - That goes row-by-row
- Flow of dependences: which values depend on which values?
- Does that give us a clue on how to parallelize?



```
for diagonal = 0 to 2*N-2 {
 parallel loop over values in the diagonal
  { i= .. ; j = ..;
    old = A[i,j];
    A[i, j] = … ;
    if (|A[i,j]-old| > maxError)
      maxError = |A[i,j]-old|
    }
  }
```

Gauss-Seidel: parallelize each diagonal

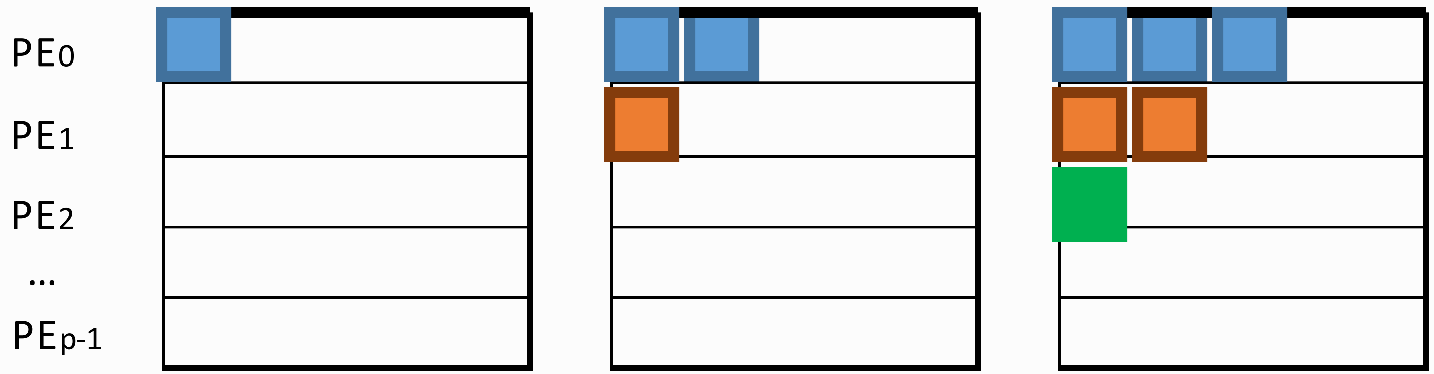
- Performance is not so good. Why?
- Each thread is doing a different (shifting) section of rows.
  - Spatial locality and prefetch efficiency is affected
- Too fine grained a loop? There are 2N parallel loops
- Other reasons? Implement and analyze with PAPI or perf tools



```
for diagonal = 0 to 2*N-2 {
 parallel loop over values in the diagonal
  { i= .. ; j = ..;
    old = A[i,j];
    A[i, j] = … ;
    if (|A[i,j]-old| > maxError)
      maxError = |A[i,j]-old|
    }
  }
```

Parallelizing Gauss-Seidel

- Some ideas
  - Row decomposition, with pipelining



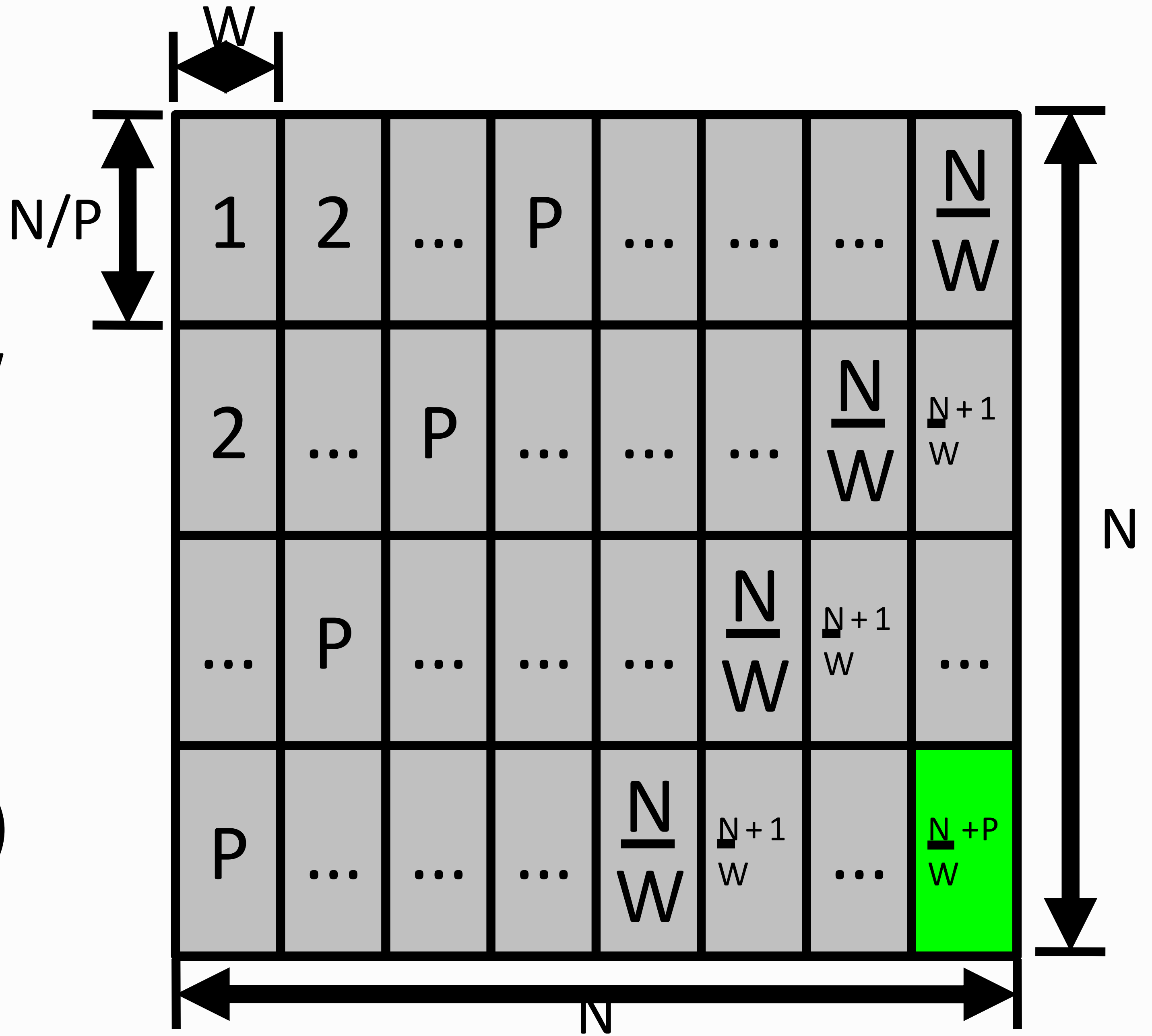
  - Square over-decomposition
    - Assign many squares to a processor (essentially same?)

Row decomposition with pipelining

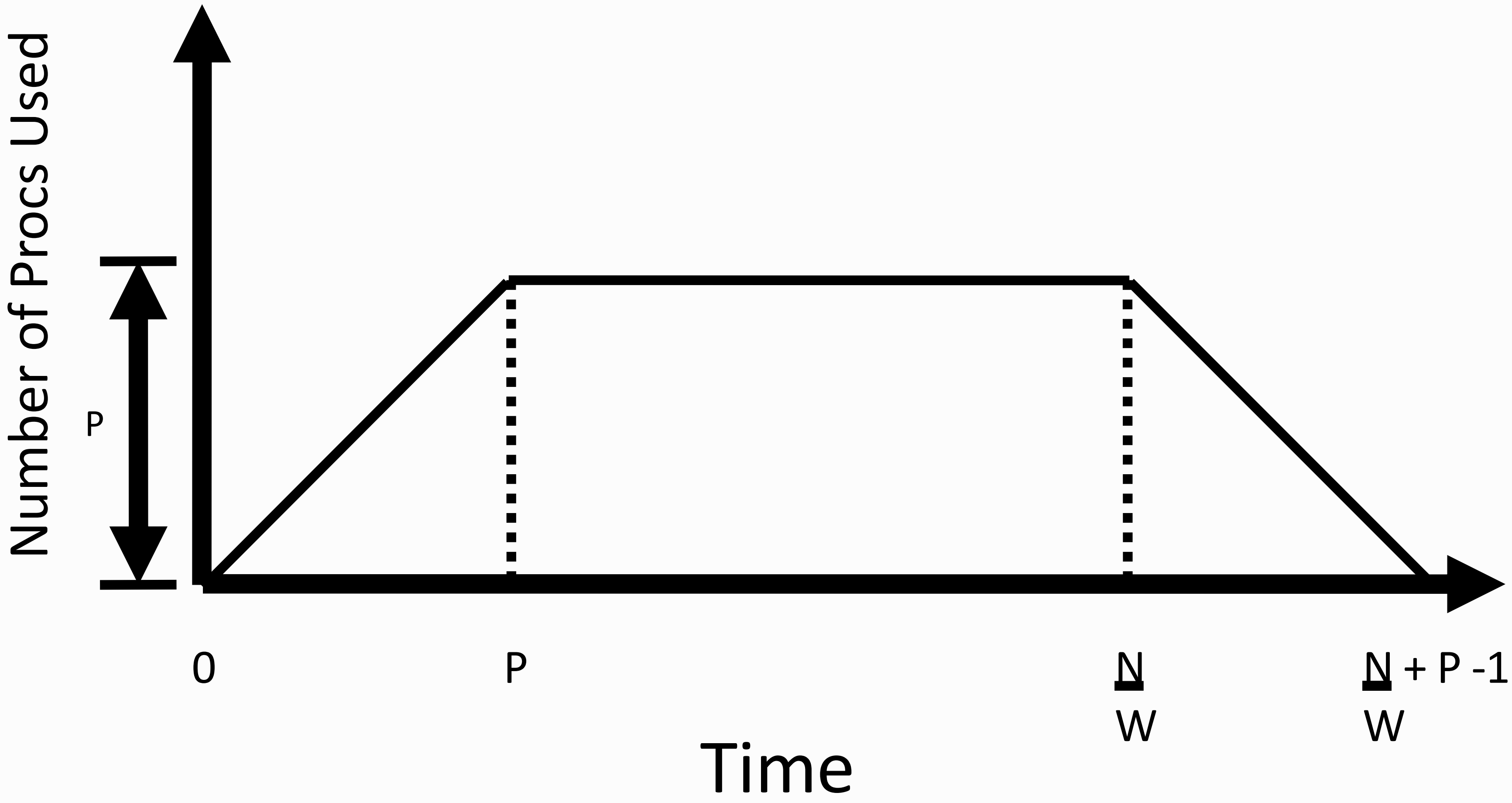
Columns = N/W
Rows = P

Of Phases

$$N/W + (P-1)$$



Row decomposition, with pipelining



Red-Black Squares Method

- Red squares calculate values based on the black squares
  - Then black squares use values from red squares
  - Now red ones can be done in parallel, and then black ones can be done in parallel
- A “square” may be just a single point
  - Or it can be a kxk tile of values
    - Each tile locally can do Gauss-Seidel computation
    - Faster convergence of Gauss-Seidel