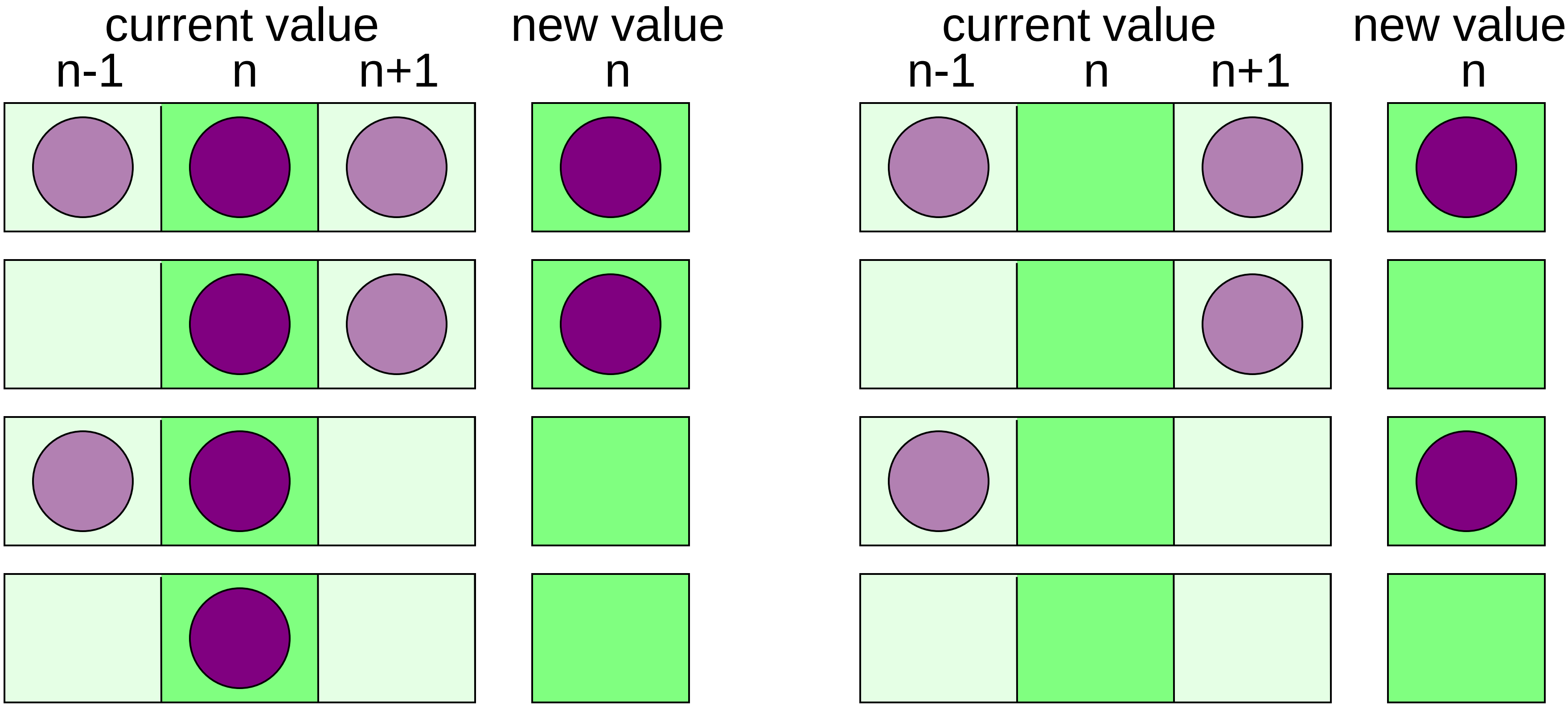
Shared-Memory Programming

Cellular Automaton Exercise

- Update rules depend on:
  - state of cell
  - state of nearest neighbours in both directions



State Table

- If $R^t(i) = 0$, then $R^{t+1}(i)$ is given by:

	$R^t(i-1) = 0$	$R^t(i-1) = 1$
$R^t(i+1) = 0$	0	1
$R^t(i+1) = 1$	0	1

- If $R^t(i) = 1$, then $R^{t+1}(i)$ is given by:

	$R^t(i-1) = 0$	$R^t(i-1) = 1$
$R^t(i+1) = 0$	0	0
$R^t(i+1) = 1$	1	1

Pseudo Code

```
declare arrays old(i) and new(i), i = 0,1,...,N,N+1
initialise old(i) for i = 1,2,...,N-1,N (eg randomly)
loop over iterations
  set old(0) = old(N) and set old(N+1) = old(1)
  loop over i = 1,...,N
    if old(i) = 1
      if old(i+1) = 1 then new(i) = 1 else new(i) = 0
    if old(i) = 0
      if old(i-1) = 1 then new(i) = 1 else new(i) = 0
  end loop over i
  set old(i) = new(i) for i = 1,2,...,N-1,N
end loop over iterations
```

- Load balance not an issue
  - updates take equal computation regardless of state of road
  - split the road into equal pieces of size *N*/*P*
- For each piece
  - rule for cell *i* depends on cells *i*-1 and *i*+1
  - can parallelise as we are updating new array based on old
- Synchronisation required
  - to ensure threads do not start until boundary data is updated
  - to produce a global sum of the number of cars that move
  - to ensure that all threads have finished before next iteration

Shared Variables Parallelisation

```
serial: initialise old(i) for i = 1,2,...,N-1,N
serial: loop over iterations
  serial: set old(0) = old(N) and set old(N+1) = old(1)
  parallel: loop over i = 1,...,N
              if old(i) = 1
                if old(i+1) = 1 then ...
              if old(i) = 0
                if old(i-1) = 1 then ...
              end loop over i
  synchronise
  parallel: set old(i) = new(i) for i = 1,2,...,N-1,N
  synchronise
end loop over iterations
```

- private: i; shared: old, new, N
  - reduction operation to compute number of moves