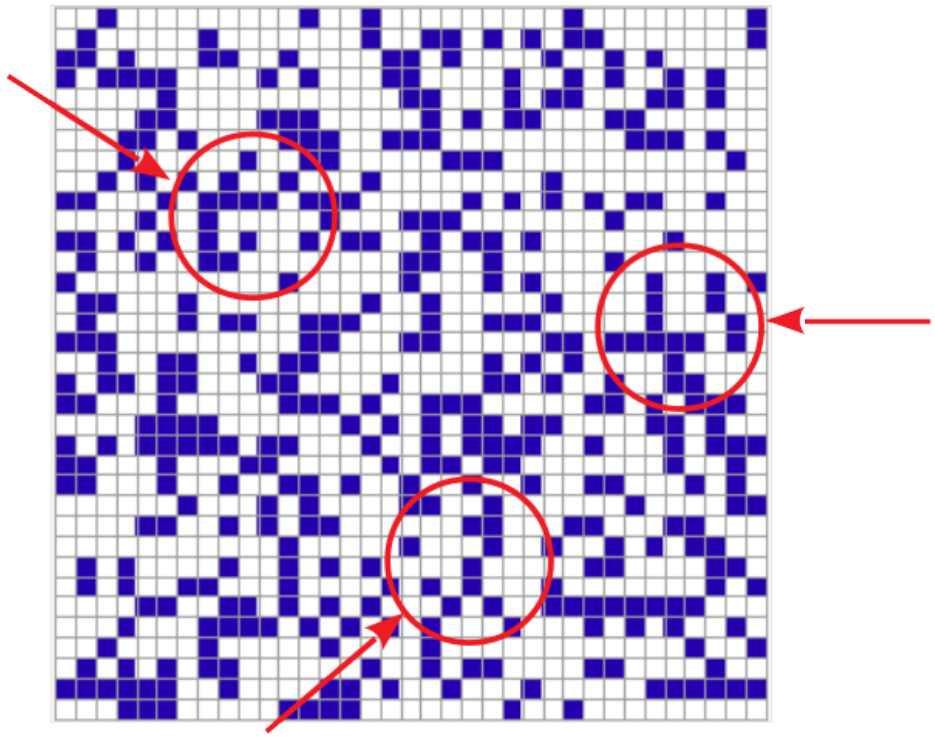
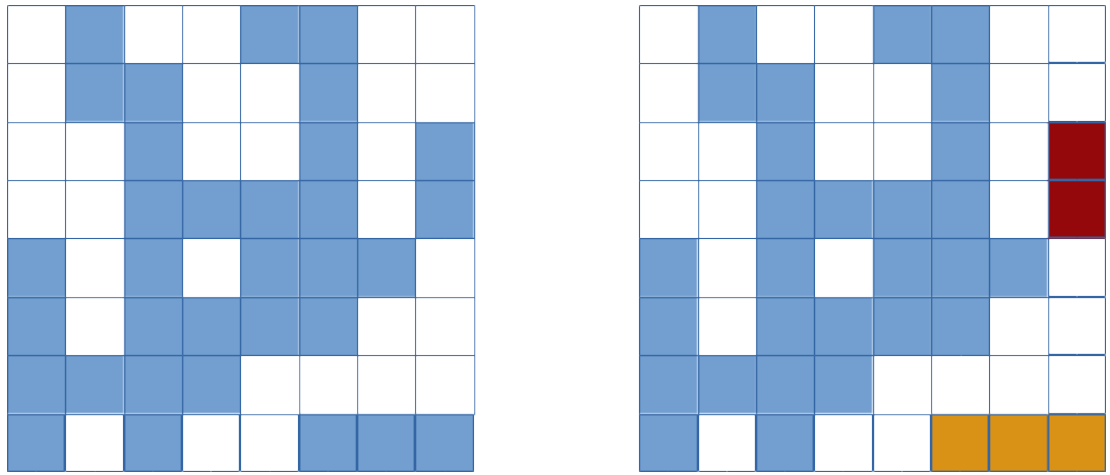
¿Cómo estudiar Percolación?

- Recordemos que queremos estudiar un sistema "infinito".



- Cada "red" que armamos es una "muestra" de la red infinita. Pasamos de observar "muestras" del sistema.

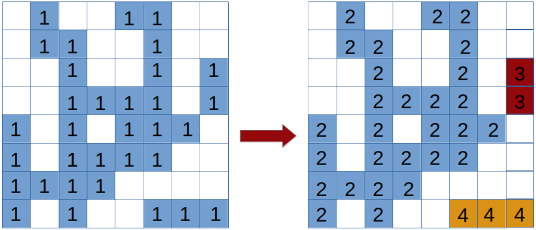
Veamos una muestra del sistema:



- En azul están los sitios “llenos” y en blanco los sitios “vacíos”.
- Para darnos cuenta si hay percolación, debemos hallar los “fragmentos” que aparecen en el sistema.

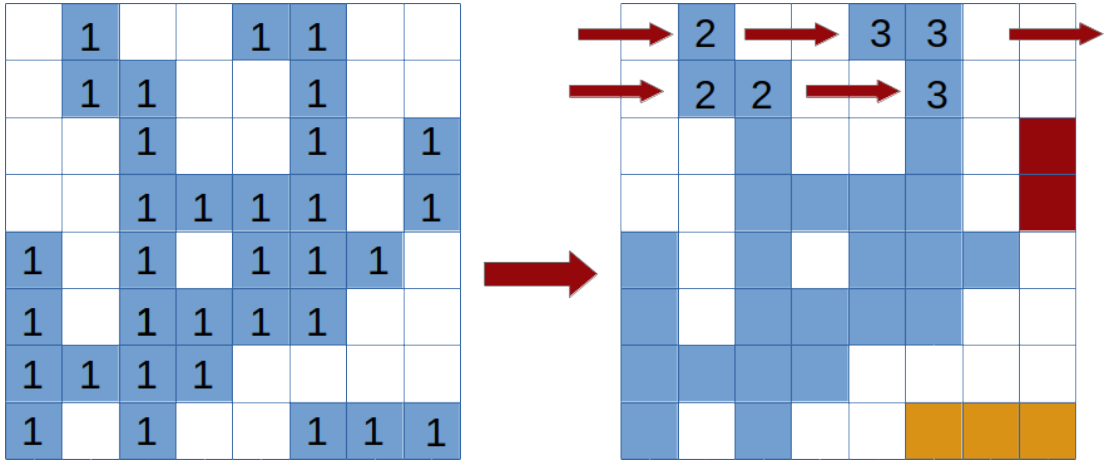
Clasificador: el algoritmo de Hoshen-Kopelman

- La idea es clasificar los fragmentos de manera eficiente.
- Clasificar es "etiquetar", del siguiente modo:



- Tenemos que leer la red en el sentido de lectura, y numerar los fragmentos.

Comencemos a leer en el sentido de lectura...



- Cada vez que aparece un “1” en la red lo reemplazamos por una etiqueta nueva. Vamos a mantener un registro de etiquetas en un vector (puntero) llamado `*clase`. Lo inicializamos así:

```
for (i=0;i<N;i++) *(clase+i) = i;
frag = 2;
```

Comencemos a leer en el sentido de lectura...

- Primer elemento:

```
if (*red == 1)
  {
    *red = frag;
    frag++;
  }
```

- Primer renglón:

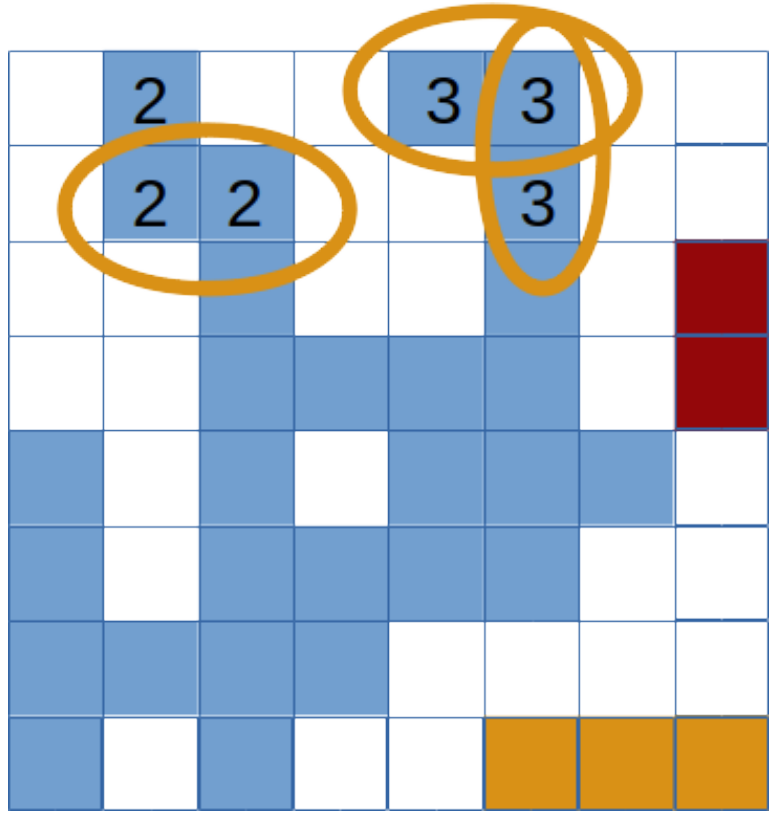
```
for (i=1;i<M;i++)          // M = elementos renglón
    if (*(red+i) == 1)
      if (*(red+i-1)==0)     // vecino izquierdo
        {
          *(red+i) = frag;
          frag++;
        }
      else *(red+i) = *(red+i-1);
```

Leemos el resto de los renglones...

```
for (i=M;i<N;i=i+M)
  {
    if (*(red+i)) actualizar con anterior...

    for (j=1;j<M;j++)
      {
        s1 = *(red+i+j-1); // vecino izquierdo
        s2 = *(red+i+j-M); // vecino de arriba

        if (*(red+i+j) == 1)
          {
            if ((s1==0) && (s2==0)) nuevo(...);
            if ((s1>1) && (s2==0))  actualizar(...);
            if ((s2>1) && (s1==0))  actualizar(...);
            if ((s2>1) && (s1>1))   conflicto(...);
          }
      }
  }
```

¿Qué significa la función `actualizar()`?

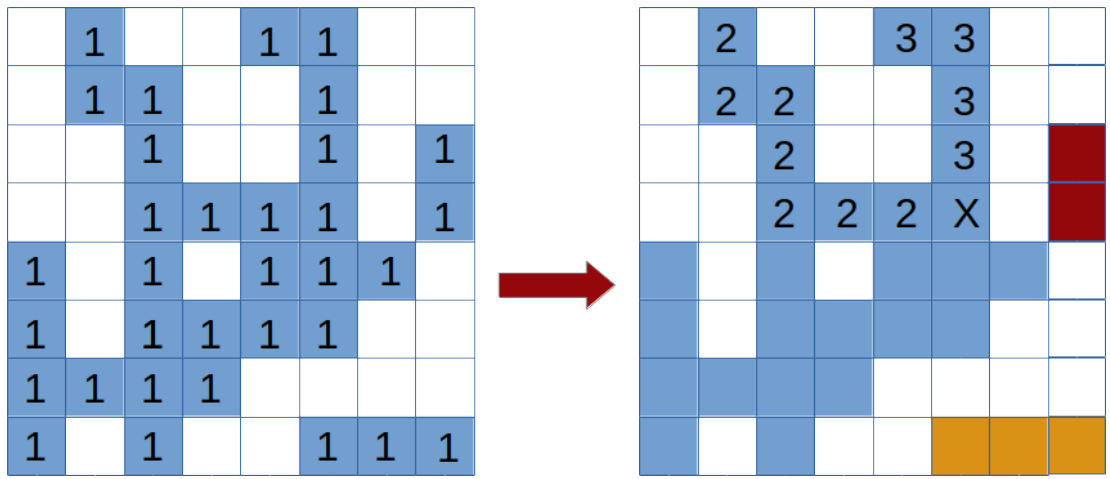
- Actualizar es simplemente copiar la etiqueta del único vecino con etiqueta



- ...dejemos esta rutina para dentro de un rato...

¿Qué significa la función `conflicto()`?

- Conflicto es resolver los problemas de "inconsistencias":



- Atención: si elegimos que X=2 hay que registrar que "3" es una etiqueta obsoleta y que eventualmente hay que cambiar $3 \rightarrow 2$.

¿Qué significa la función `conflicto()`?

- Avisamos que 3 es obsoleta en el registro `*clase`. Para eso, le cambiamos el signo y le ponemos el valor de 2.

```
*clase = 0 1 2  3 4 5 6 .... (antes)
*clase = 0 1 2 -2 4 5 6 .... (después)
```

- ¿cómo lo hacemos?

```
while (*(clase+s1<0)) s1 = -(*(clase+s1));
while (*(clase+s2<0)) s2 = -(*(clase+s2));

if (s1<s2) { smin = s1; smax = s2; }
else       { smin = s2; smax = s1; }

*(red+i) = smin;
*(clase+smin) = smin;

if (smin<smax) *(clase+smax) = -smin;
else           *(clase+smax) = smin;
```

Volvemos a la función `actualizar()`

- Atención que `s1` y `s2` deben ser las etiquetas verdaderas (no obsoletas) para mantener todo actualizado

```
int actualizar(int *red,int *clase,int s,int i)
 {
  while (*(clase+s)<0)    s = -(*(clase+s));

  *(red+i) = s;
  *(clase+s) = s;

  return 1;
 }
```

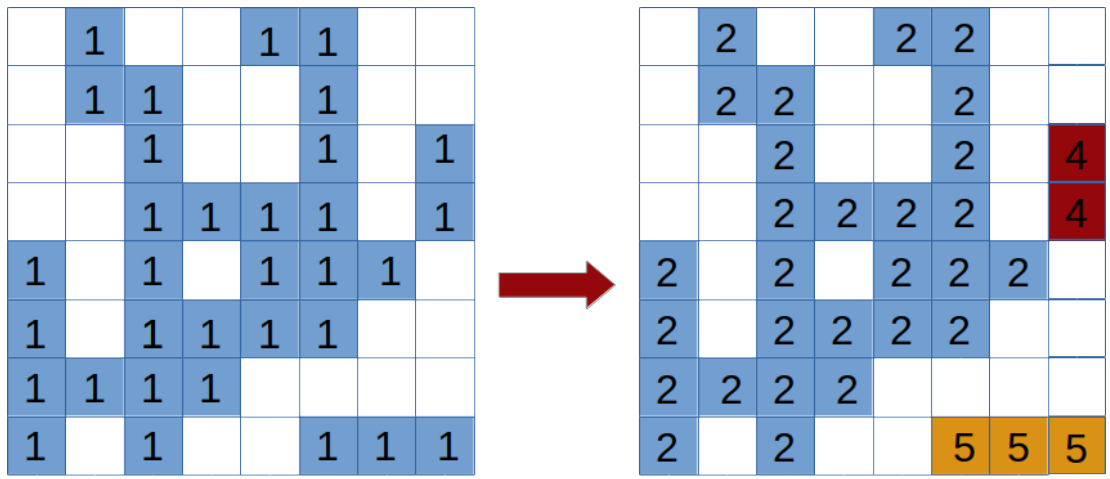
Corrección final de las etiquetas

- Al final del recorrido de la red, hacemos una pasada para actualizar todo.

```
int correccion(int *red,int *clase)
 {
  int j,s;

  for (j=0;j<N;j++)
    {
     s = *(red+j);
     while (*(clase+s)<0)   s = -(*(clase+s));

     *(red+i) = *(clase+s);
    }

  return 1;
 }
```

Detección de la Percolación.

- Consideramos sólo percolación vertical. Omitimos la percolación horizontal porque es similar a la vertical, pero rotada 90°.



- En este ejemplo, el fragmento 2 percola verticalmente.

Detección de la Percolación.

- Recorremos el primer renglón y leemos su contenido: `j = *(red+i);`.
- Activamos un "flag" en otro vector, en la posición `j`: `*(renglon1+j) = 1;`
- Hacemos lo mismo con el último renglón y comparamos ambos. Si hay coincidencias, el sistema percoló.