COMP 150-SEN
Software Engineering Foundations

Debugging

Spring 2019

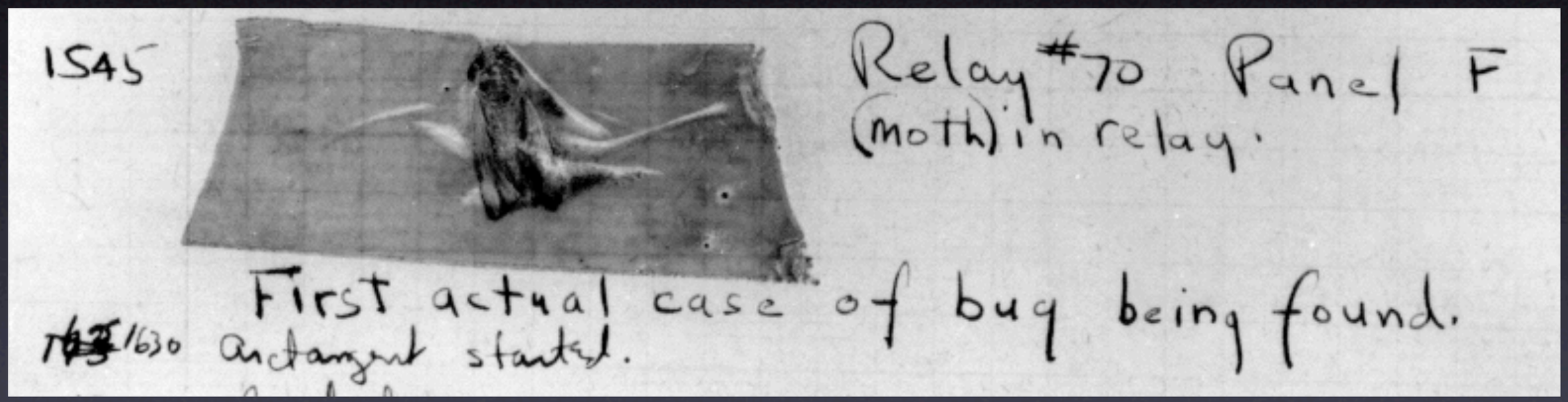
Slides selected and slightly modified from
Andreas Zeller, *Why Programs Fail*
https://www.st.cs.uni-saarland.de/whyprogramsfail/slides.php

How Failures Come to be

Andreas Zeller

The First Bug

September 9, 1947

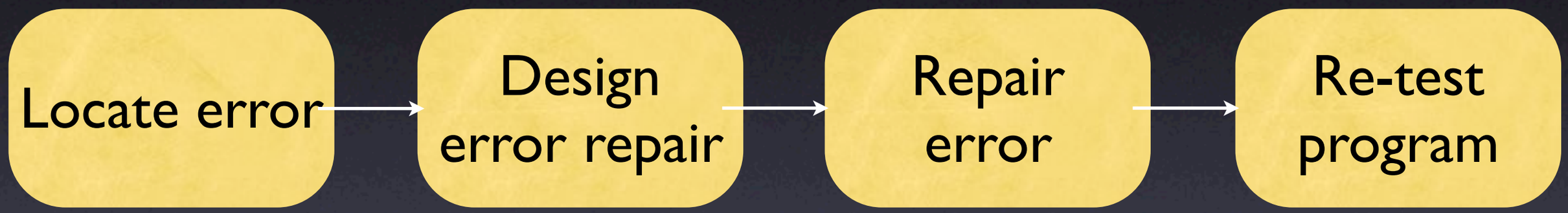


A Sample Program

```
$ sample 9 8 7
Output: 7 8 9

$ sample 11 14
Output: 0 11
```

How to Debug

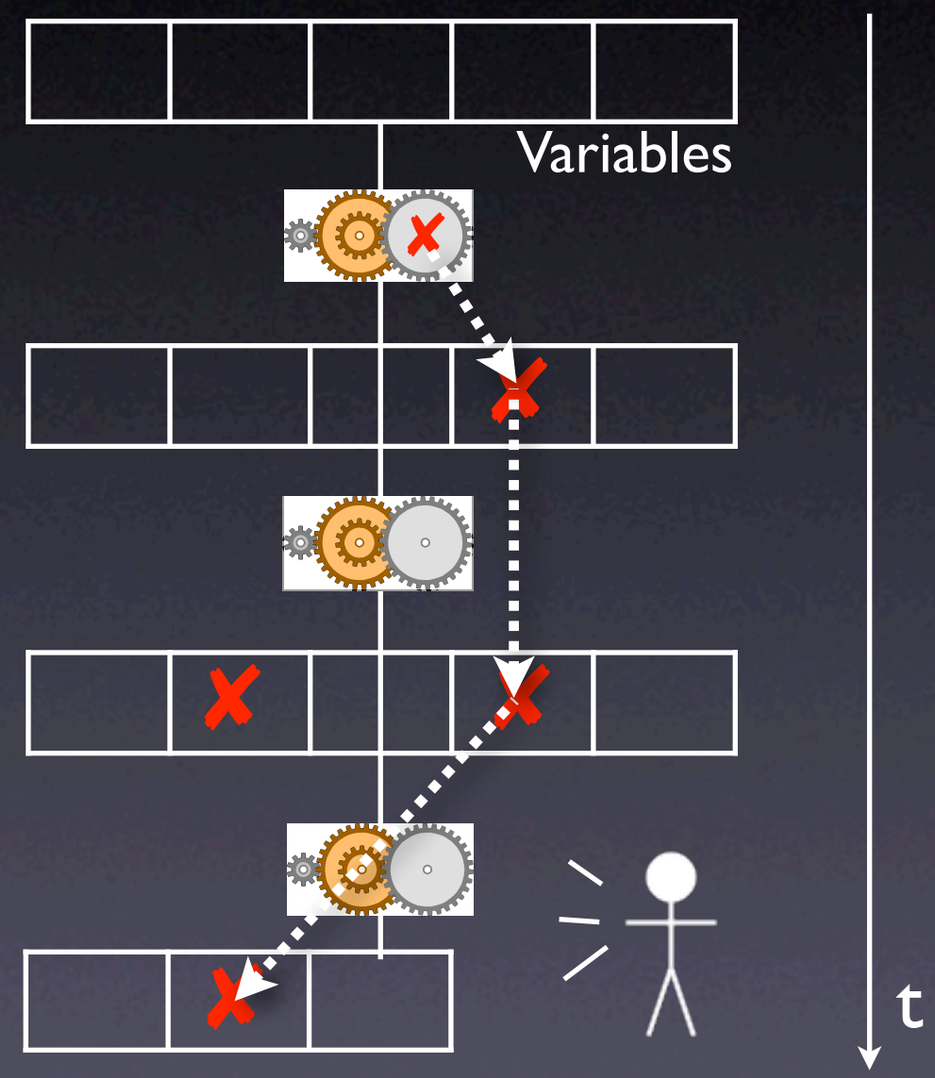
(Sommerville 2004)



From Defect to Failure

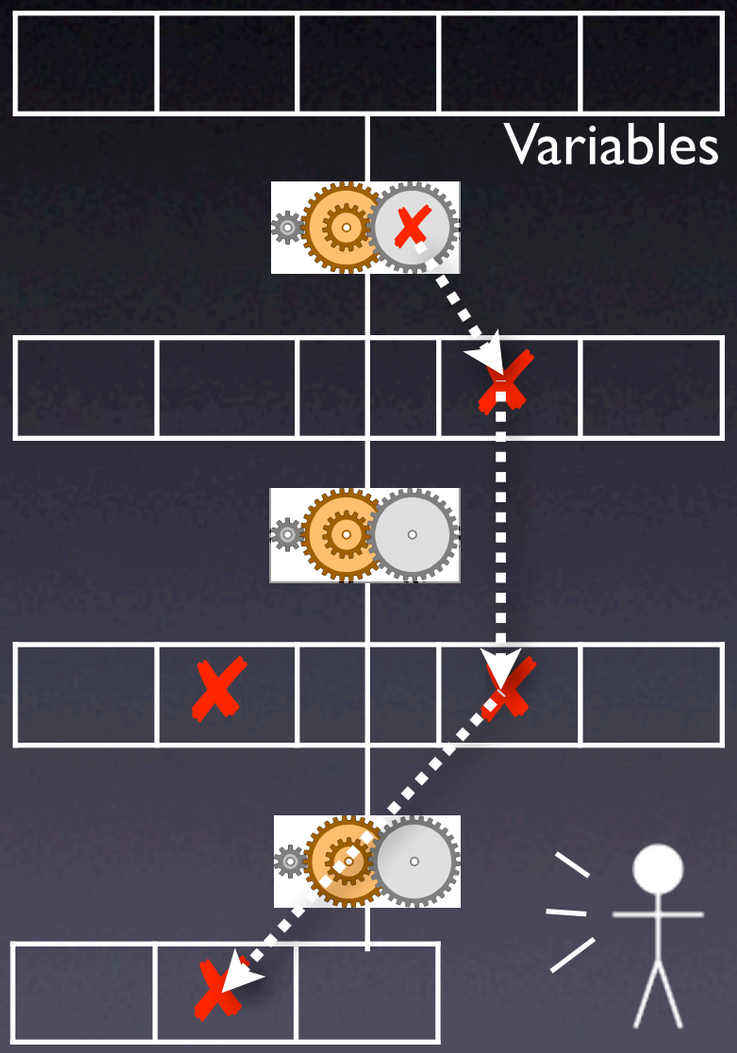
1. The programmer creates a *defect* – an error in the code.
2. When executed, the defect creates an *infection* – an error in the state.
3. The infection *propagates*.
4. The infection causes a *failure*.

This infection chain must be traced back – and broken.



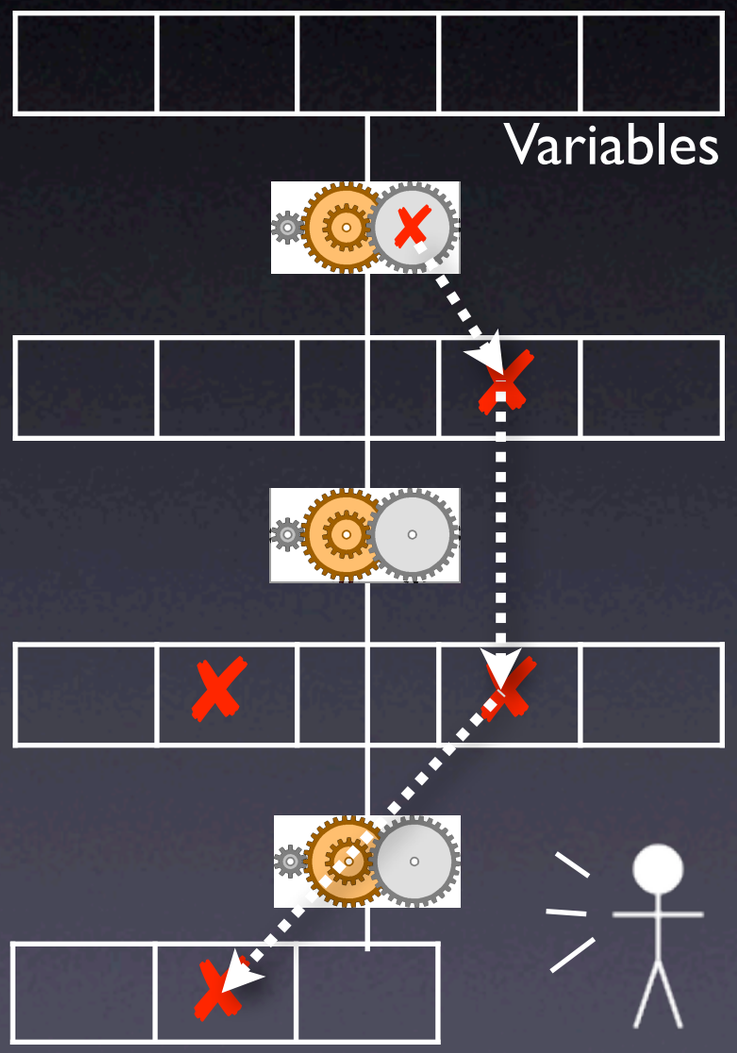
The Curse of Testing

- Not every defect causes a failure!
- *Testing can only show the presence of errors – not their absence.*
  (Dijkstra 1972)

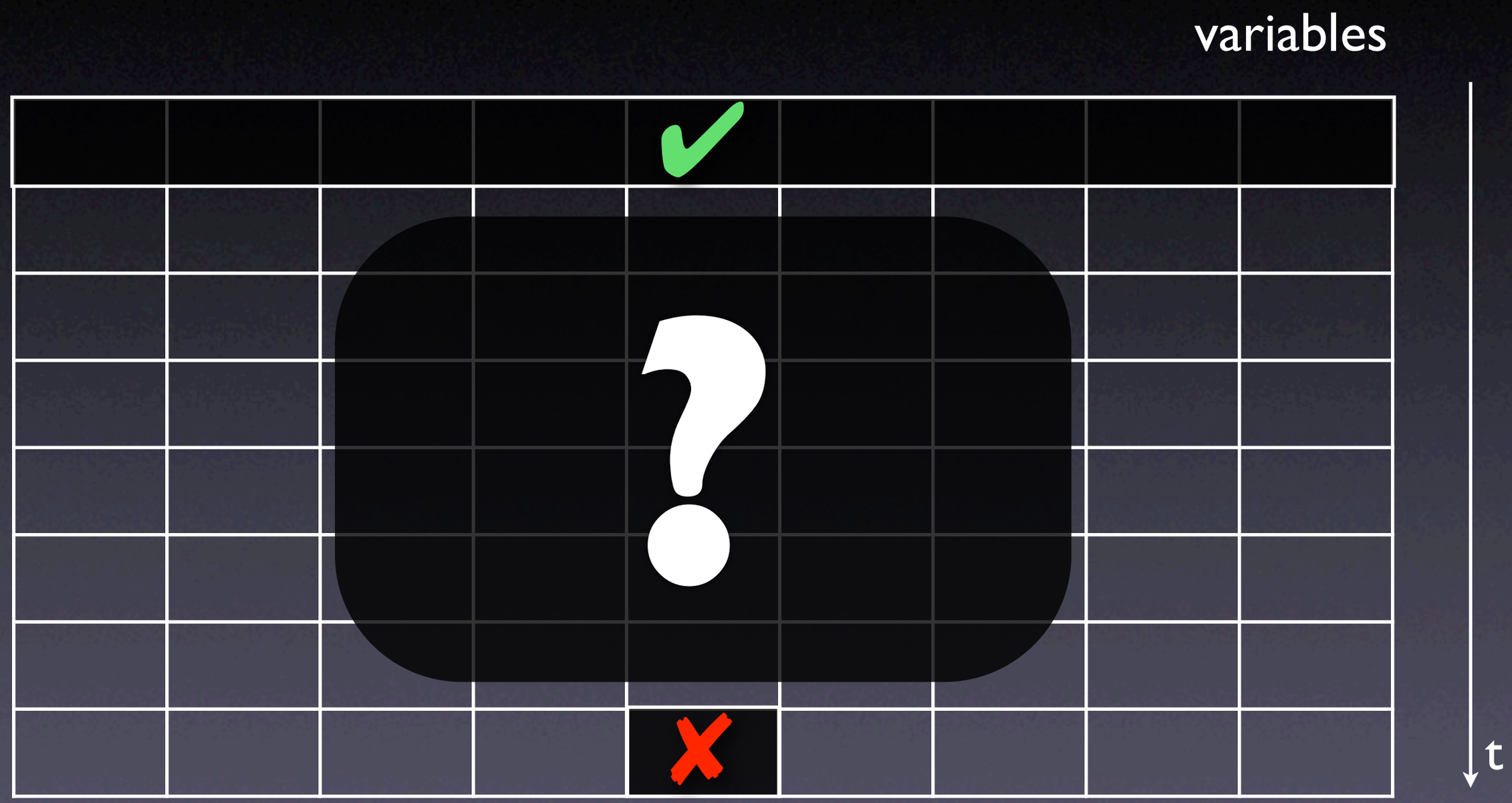


Debugging

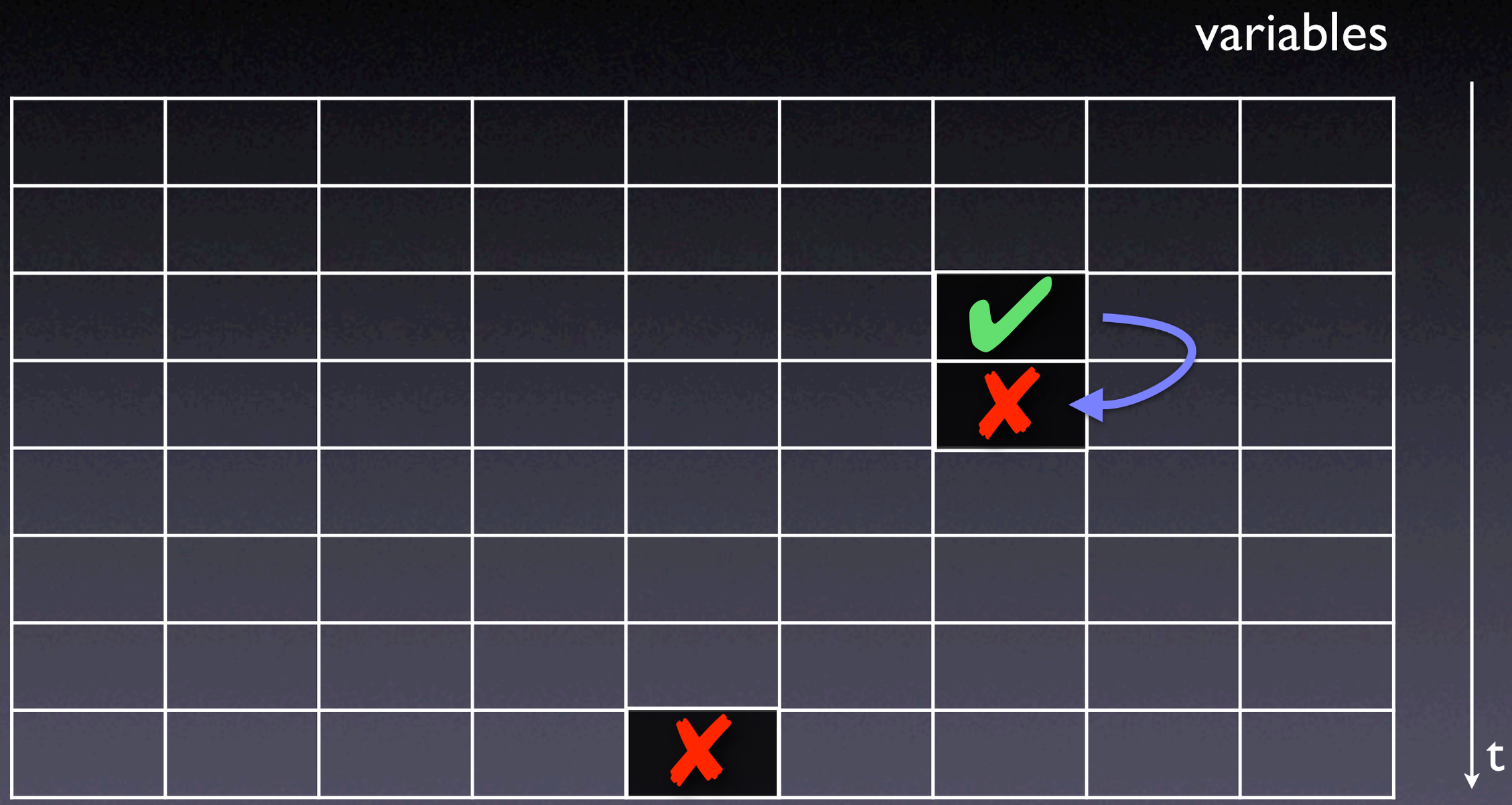
- Every *failure* can be traced back to some *infection*, and every *infection* is caused by some *defect*.
- *Debugging* means to *relate a given failure to the defect* – and to remove the defect.



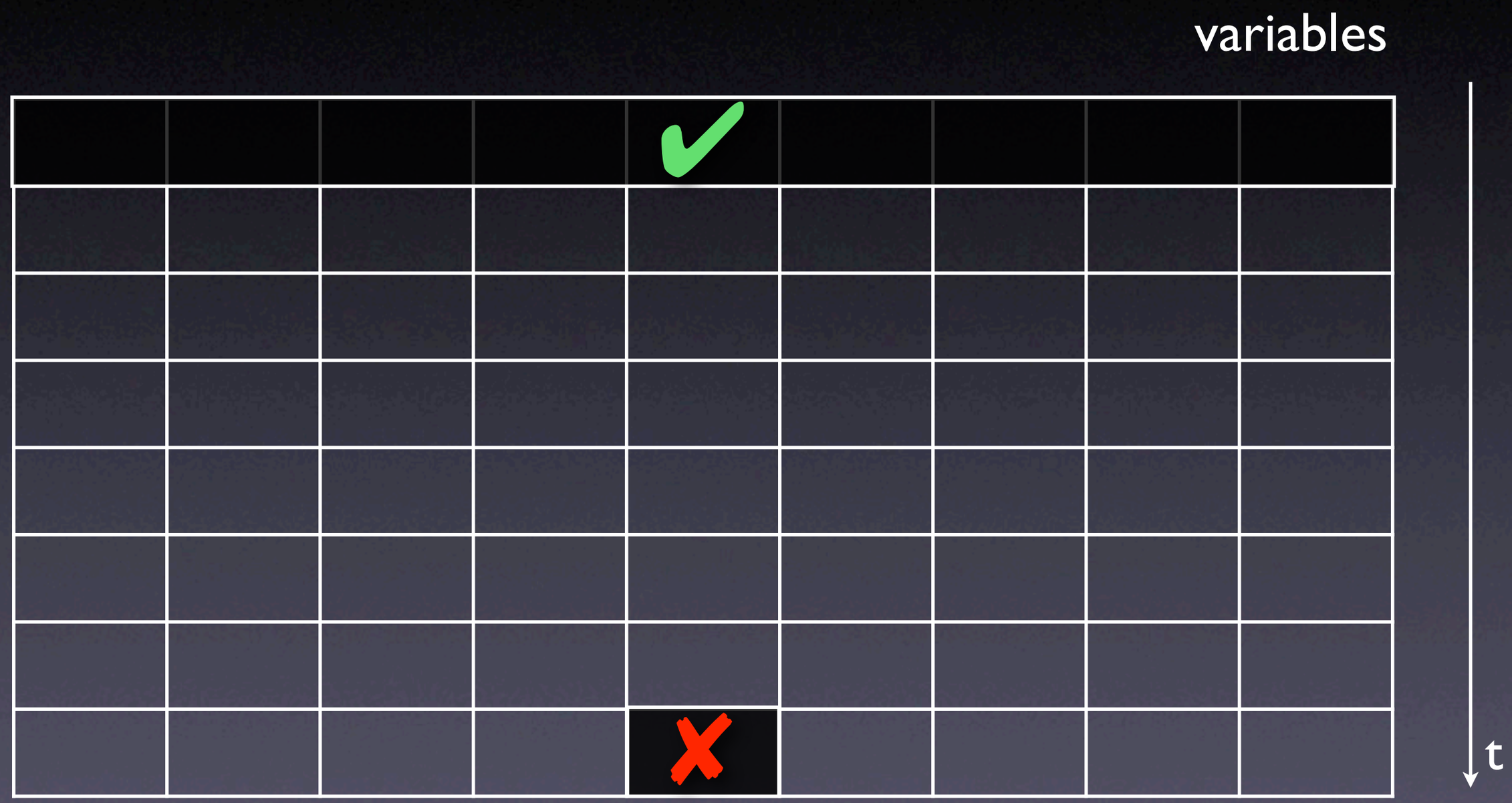
Search in Space + Time



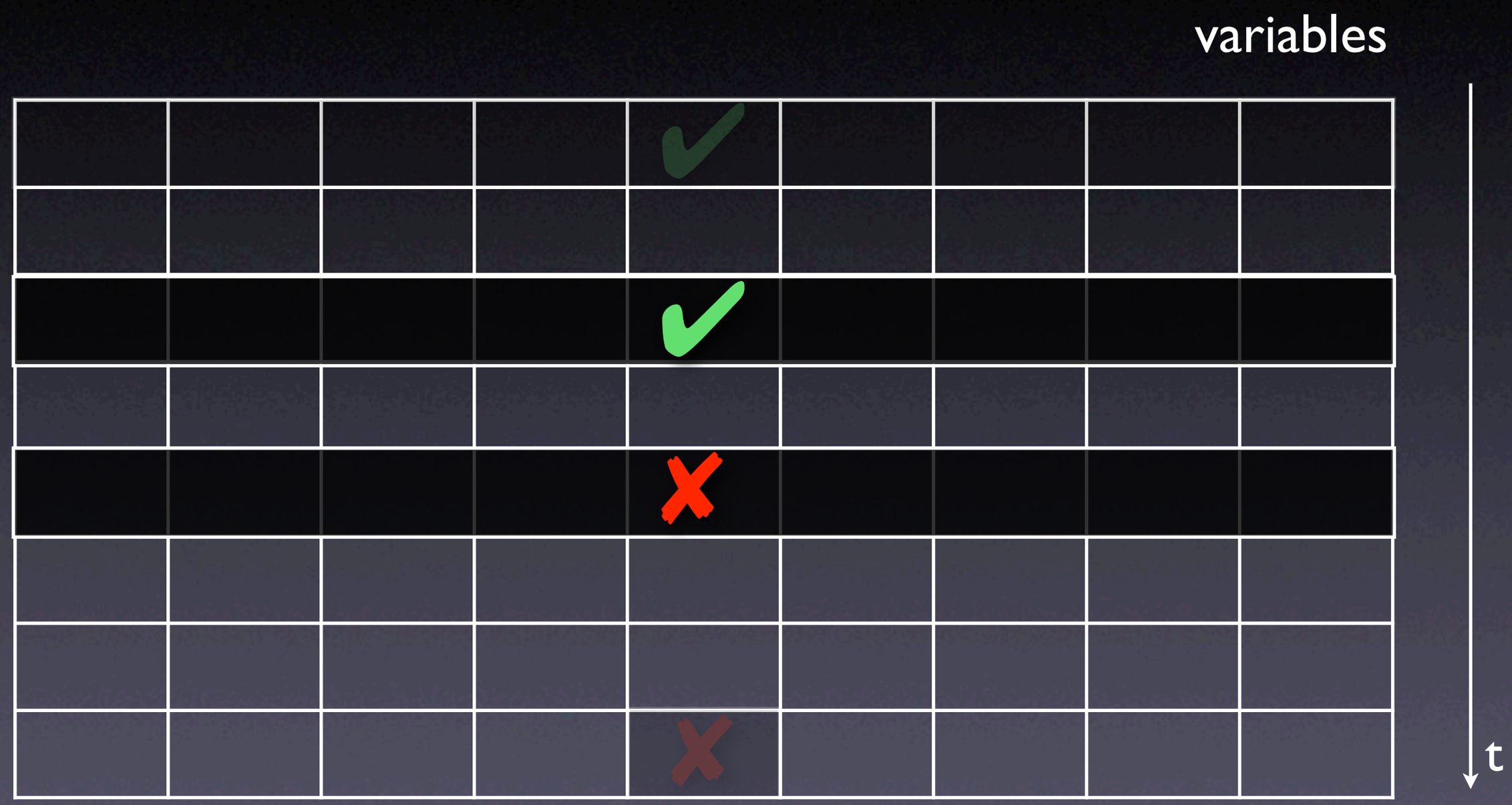
The Defect



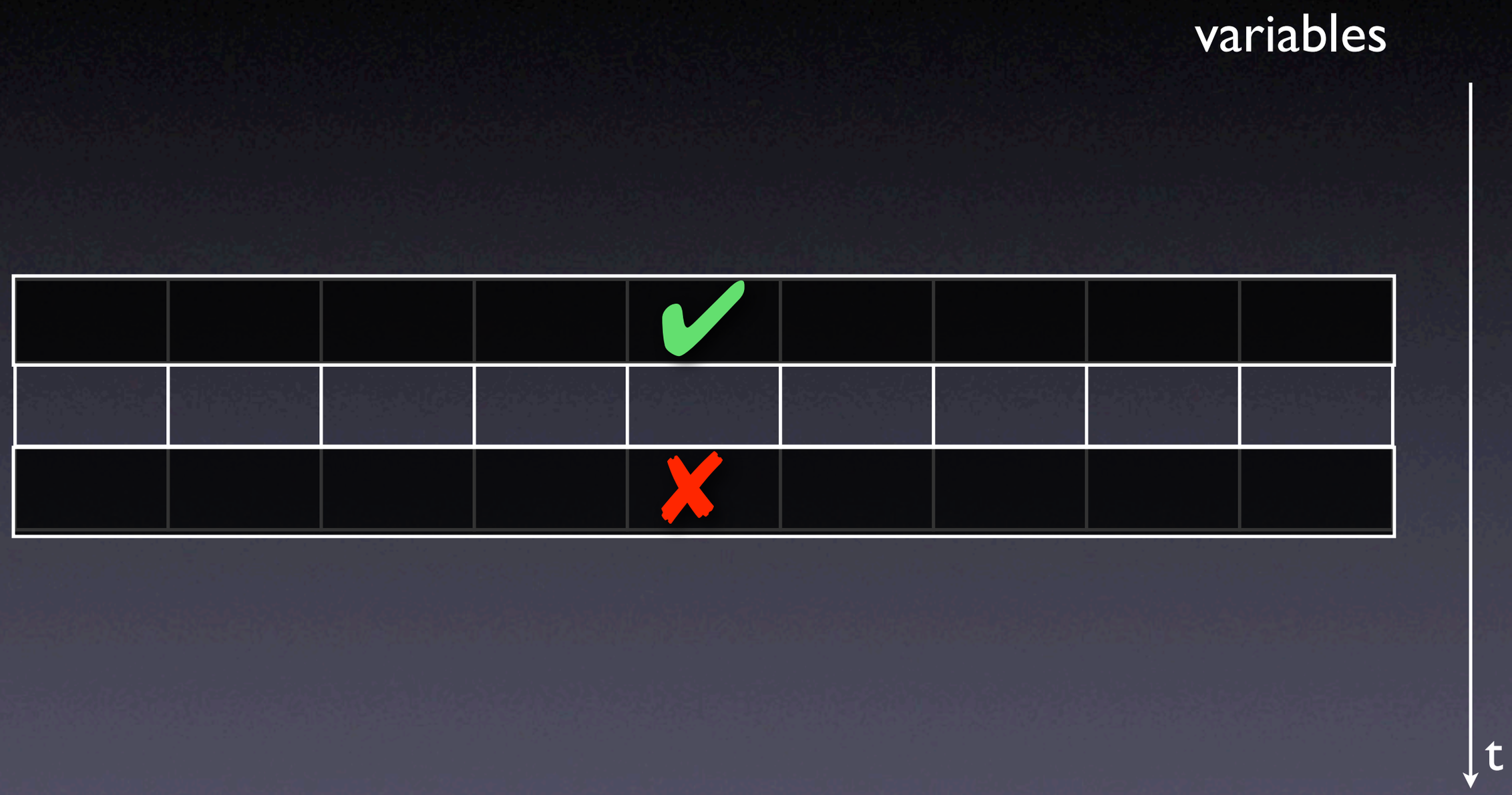
Search in Time



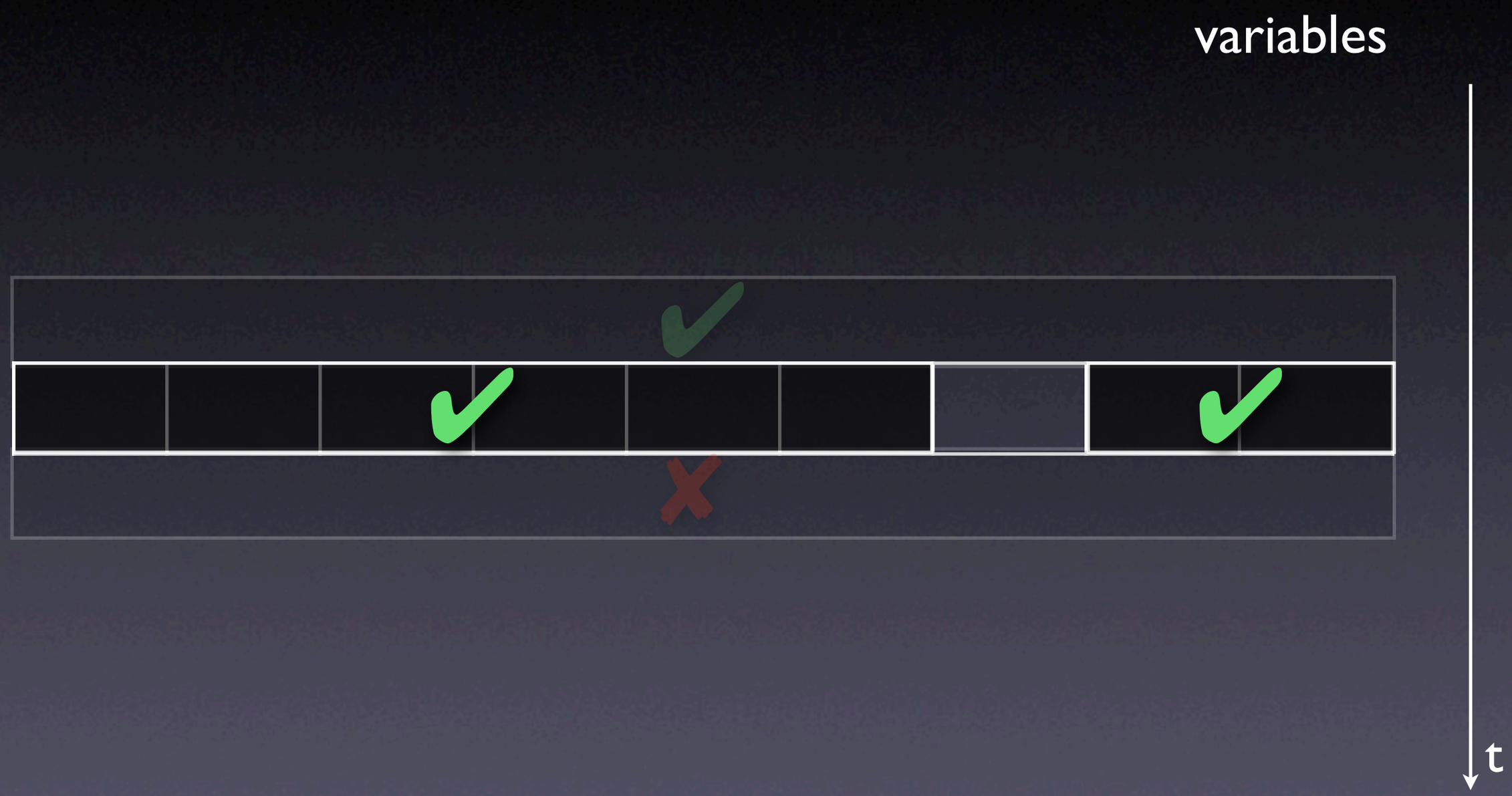
Search in Time



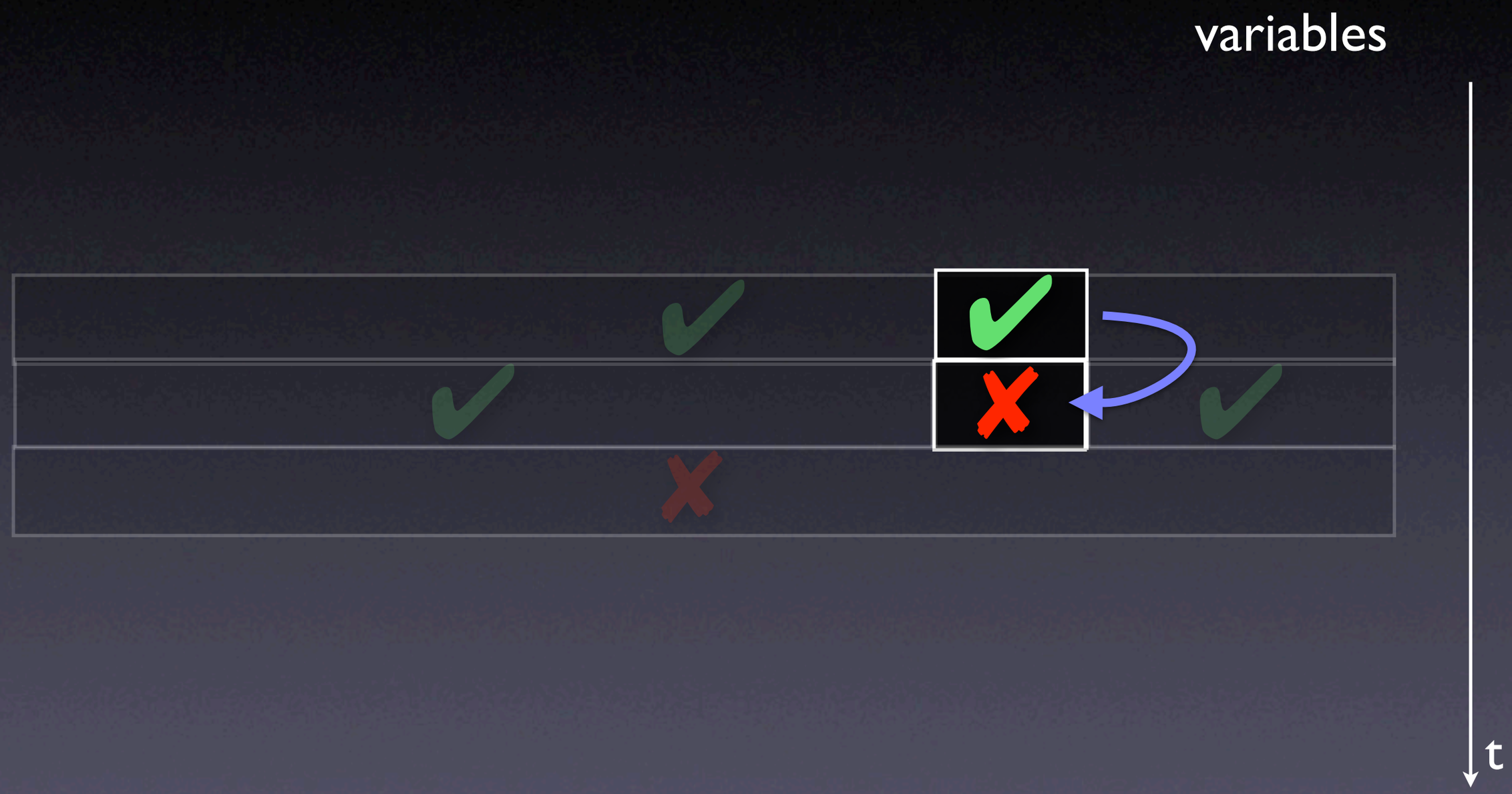
Search in Time



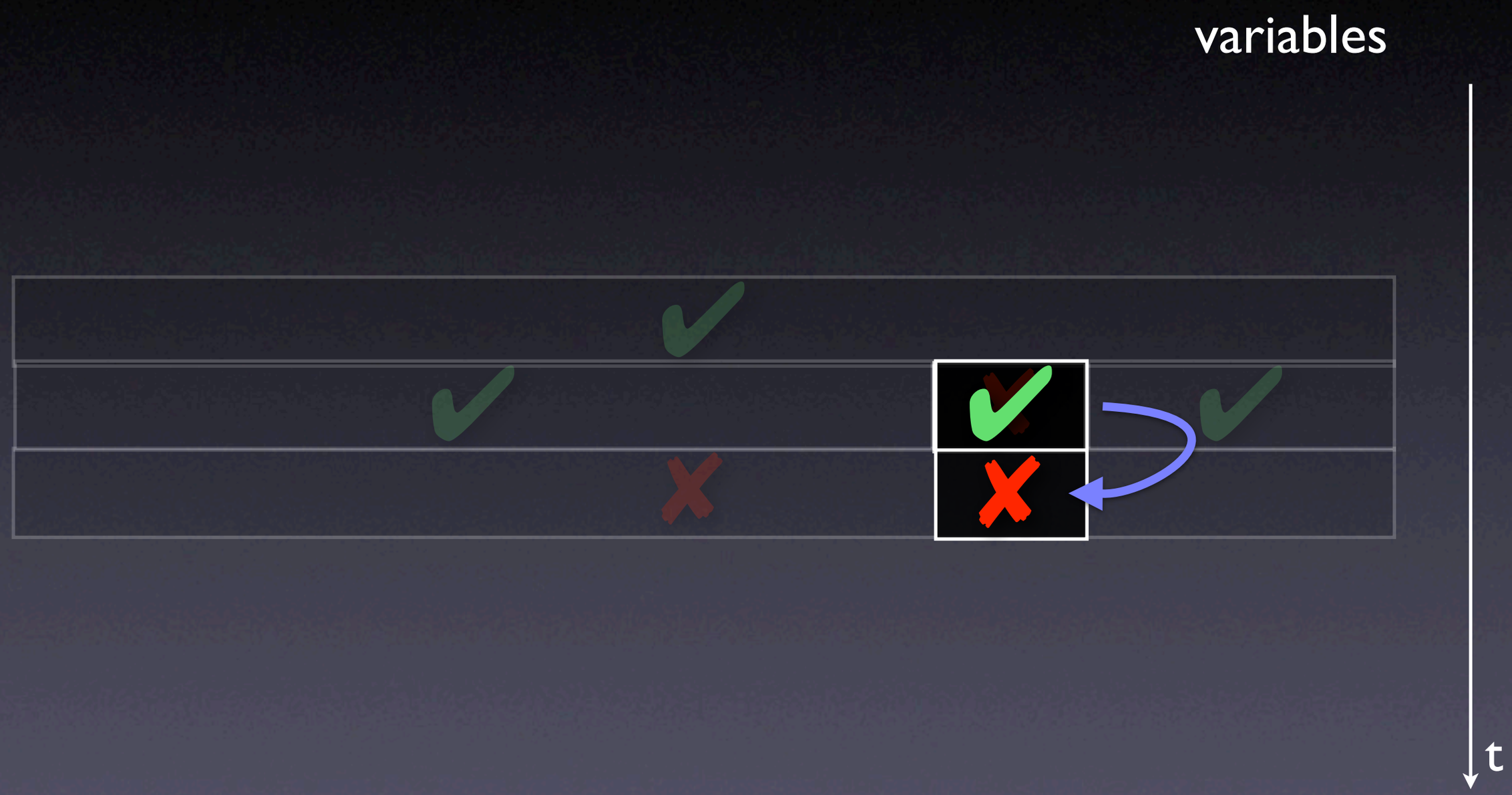
Search in Space



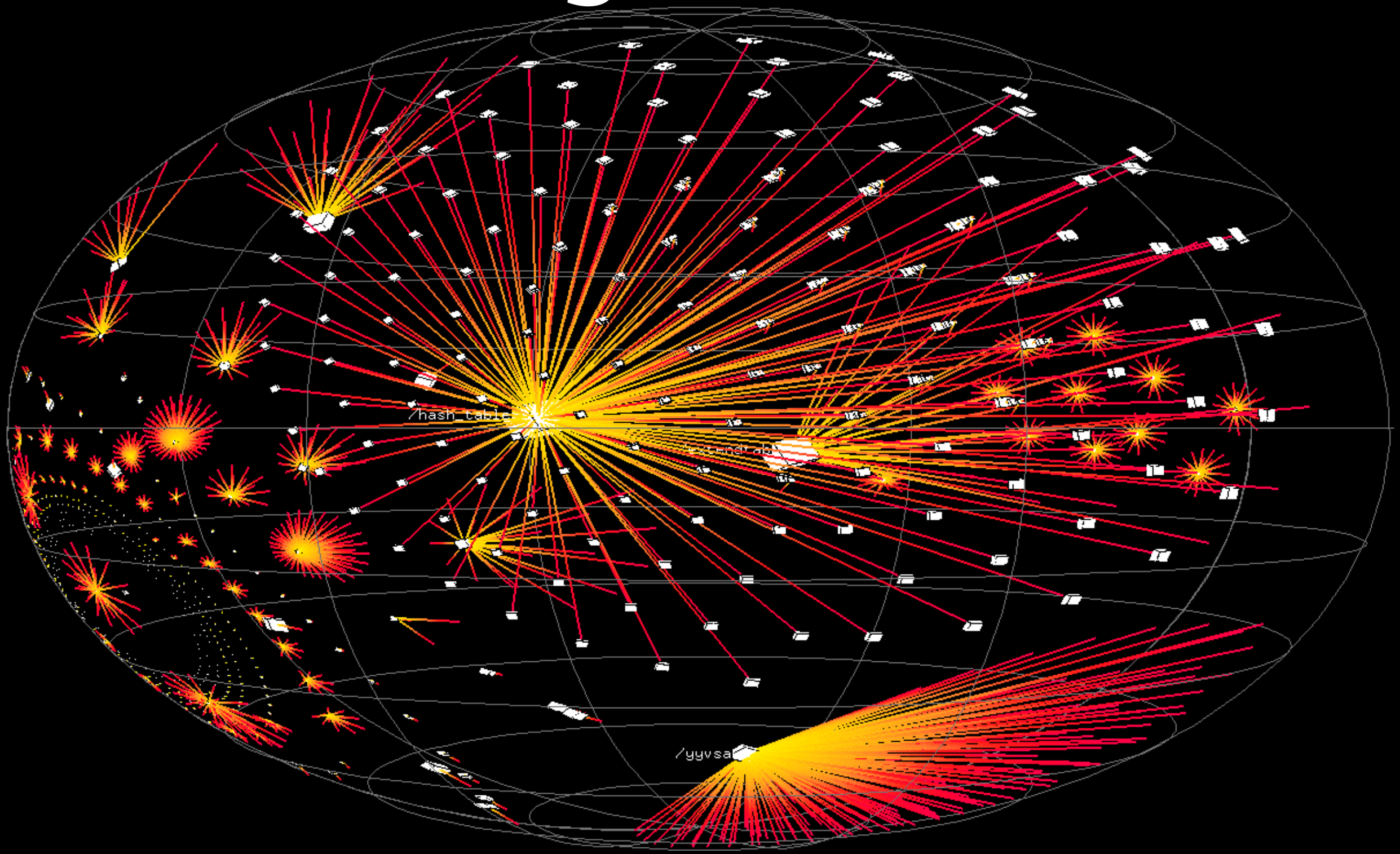
Search in Space



Search in Space



A Program State

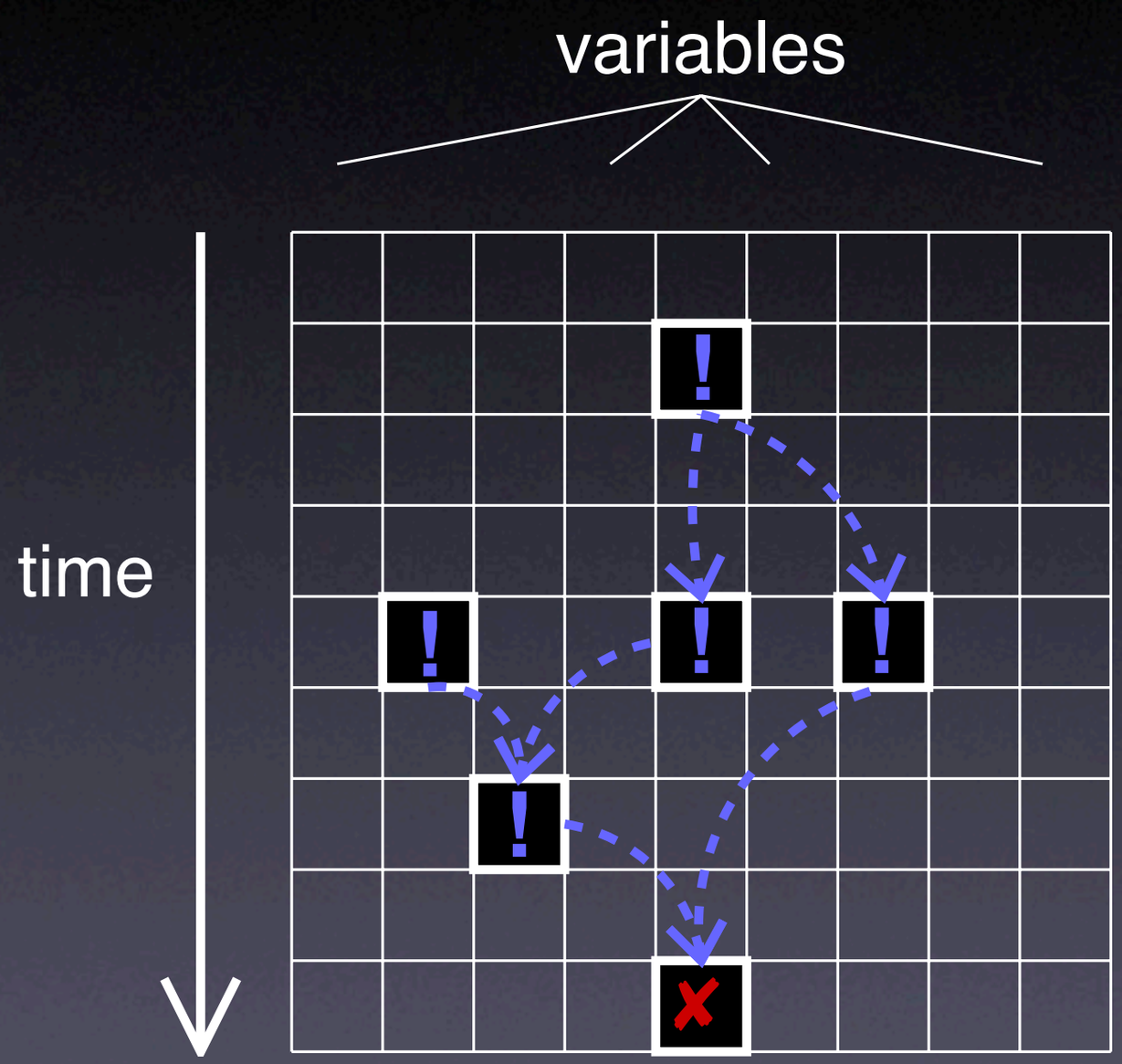


A Sample Program

```
$ sample 9 8 7
Output: 7 8 9

$ sample 11 14
```

```c
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc);

    printf("Output: ");
    for (i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");

    free(a);

    return 0;
}
```

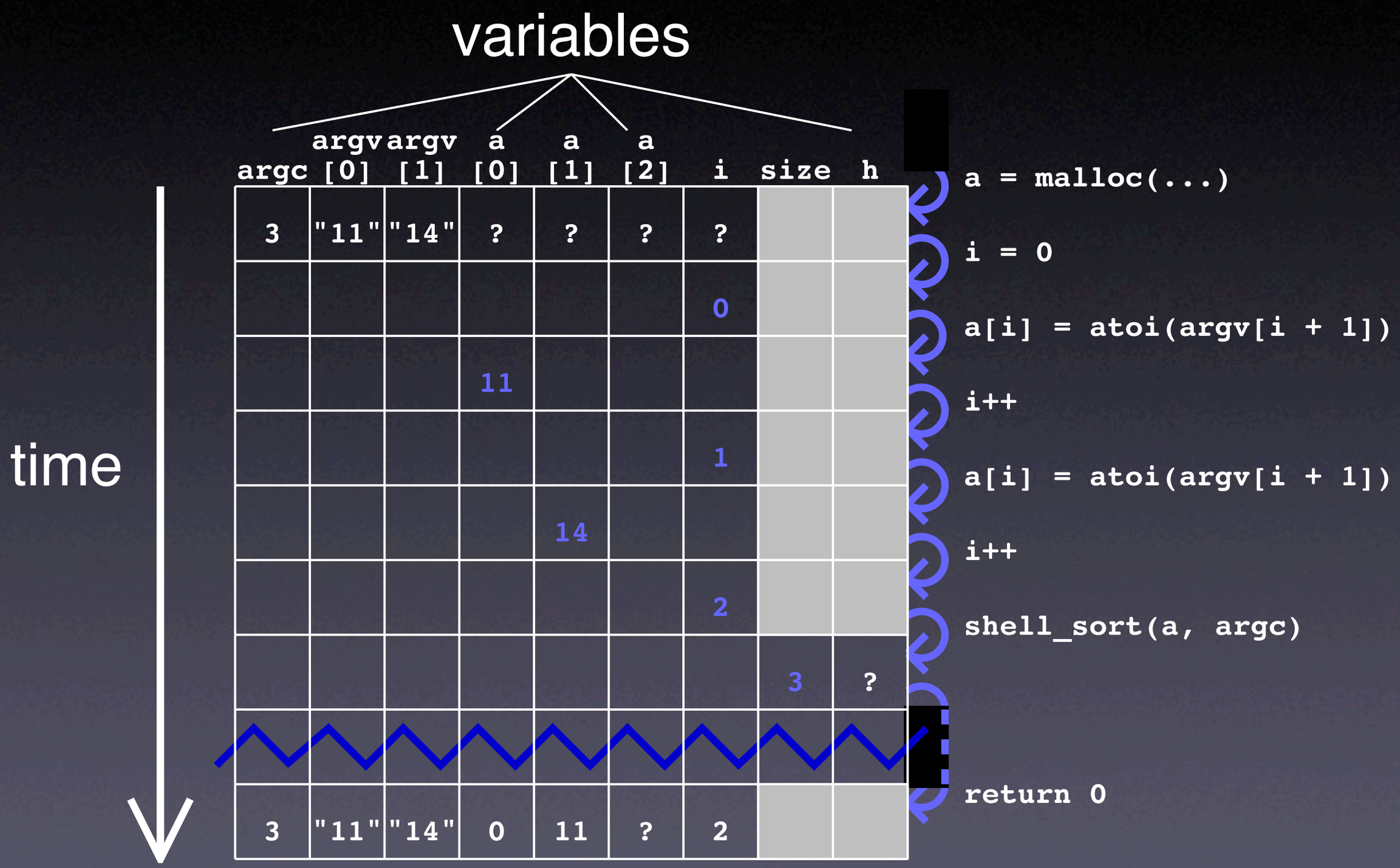
Find Origins



- The 0 printed is the value of `a[0]`. Where does it come from?
- Basic idea: Track or deduce value origins
- Separates *relevant* from *irrelevant* values
- We can trace back `a[0]` to `shell_sort`

```
static void shell_sort(int a[], int size)
{
    int i, j;
    int h = 1;
    do {
        h = h * 3 + 1;
    } while (h <= size);
    do {
        h /= 3;
        for (i = h; i < size; i++)
        {
            int v = a[i];
            for (j = i; j >= h && a[j - h] > v; j -= h)
                a[j] = a[j - h];
            if (i != j)
                a[j] = v;
        }
    } while (h != 1);
}
```

Observing a Run



Specific Observation

```
static void shell_sort(int a[], int size)
{
    fprintf(stderr, “At shell_sort”);
    for (i = 0; i < size; i++)
        fprintf(stderr, “a[%d] = %d\n”, i, a[i]);
    fprintf(stderr, “size = %d\n”, size);
    int i, j;
    int h = 1;
    ...
}
```

```
$ sample 11 14
a[0] = 11
a[1] = 14
a[2] = 0
size = 3
Output: 0 11
```

The state is infected at the call of `shell_sort`!

Fixing the Program

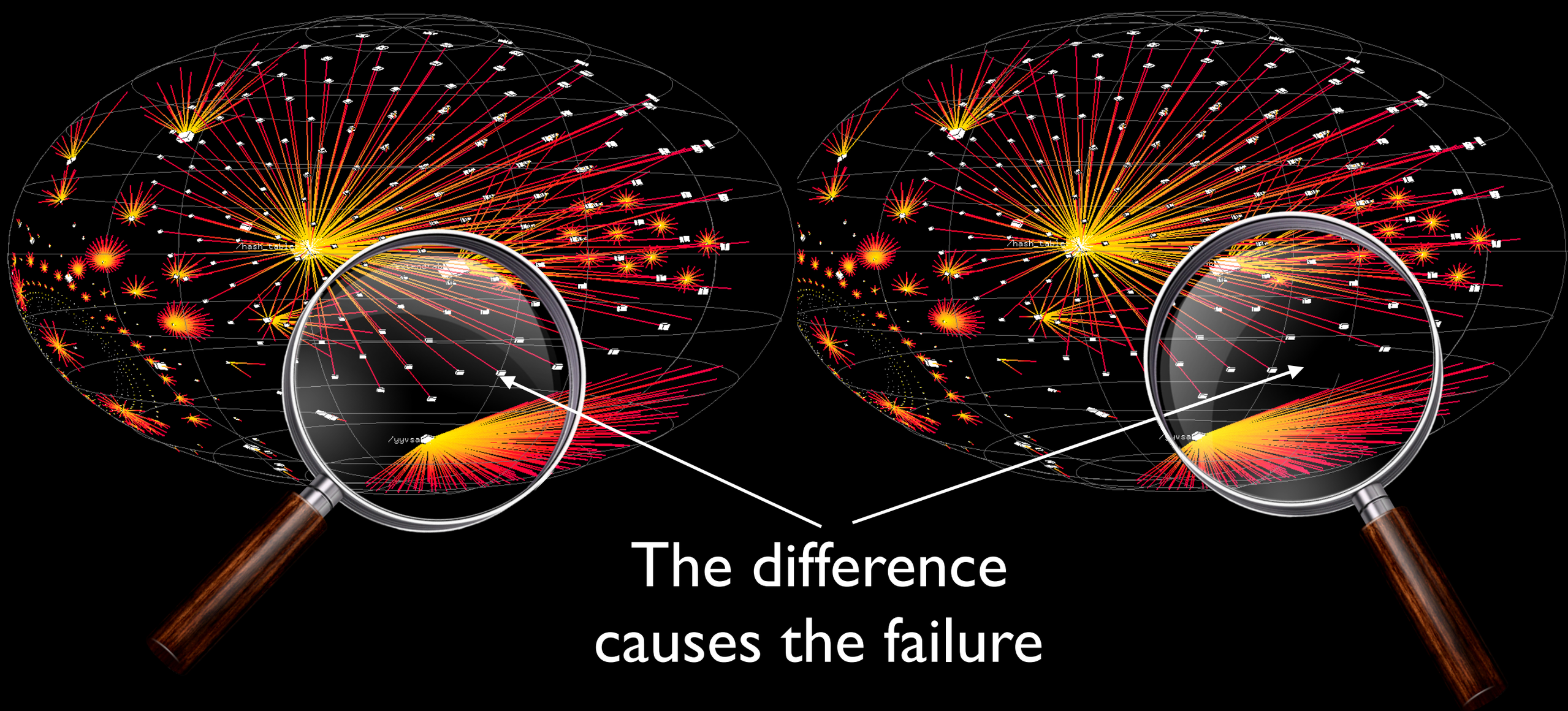
```
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc); 1);

    ...
}
```

```
$ sample 11 14
Output: 11 14
```

Finding Causes

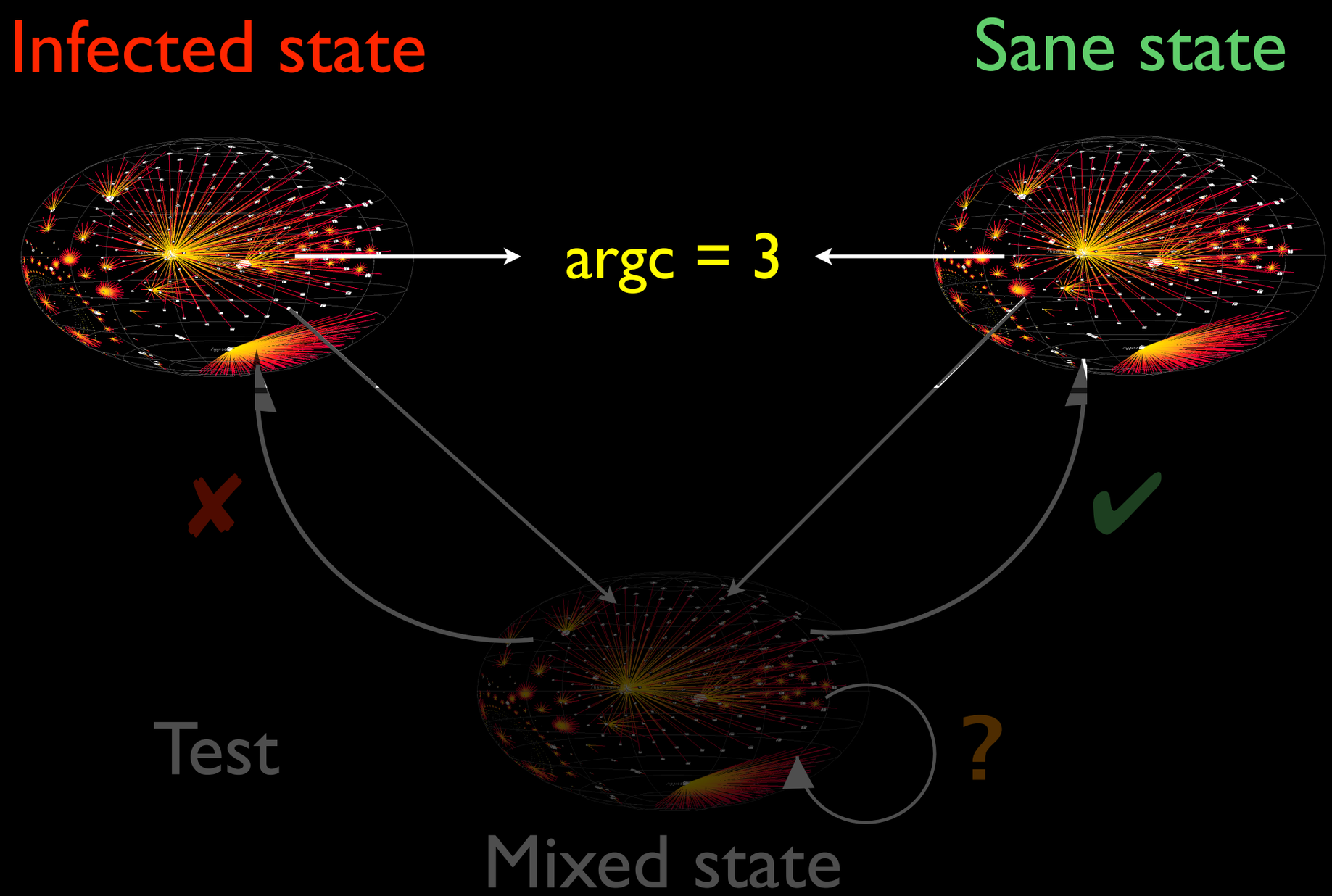
Infected state

Sane state

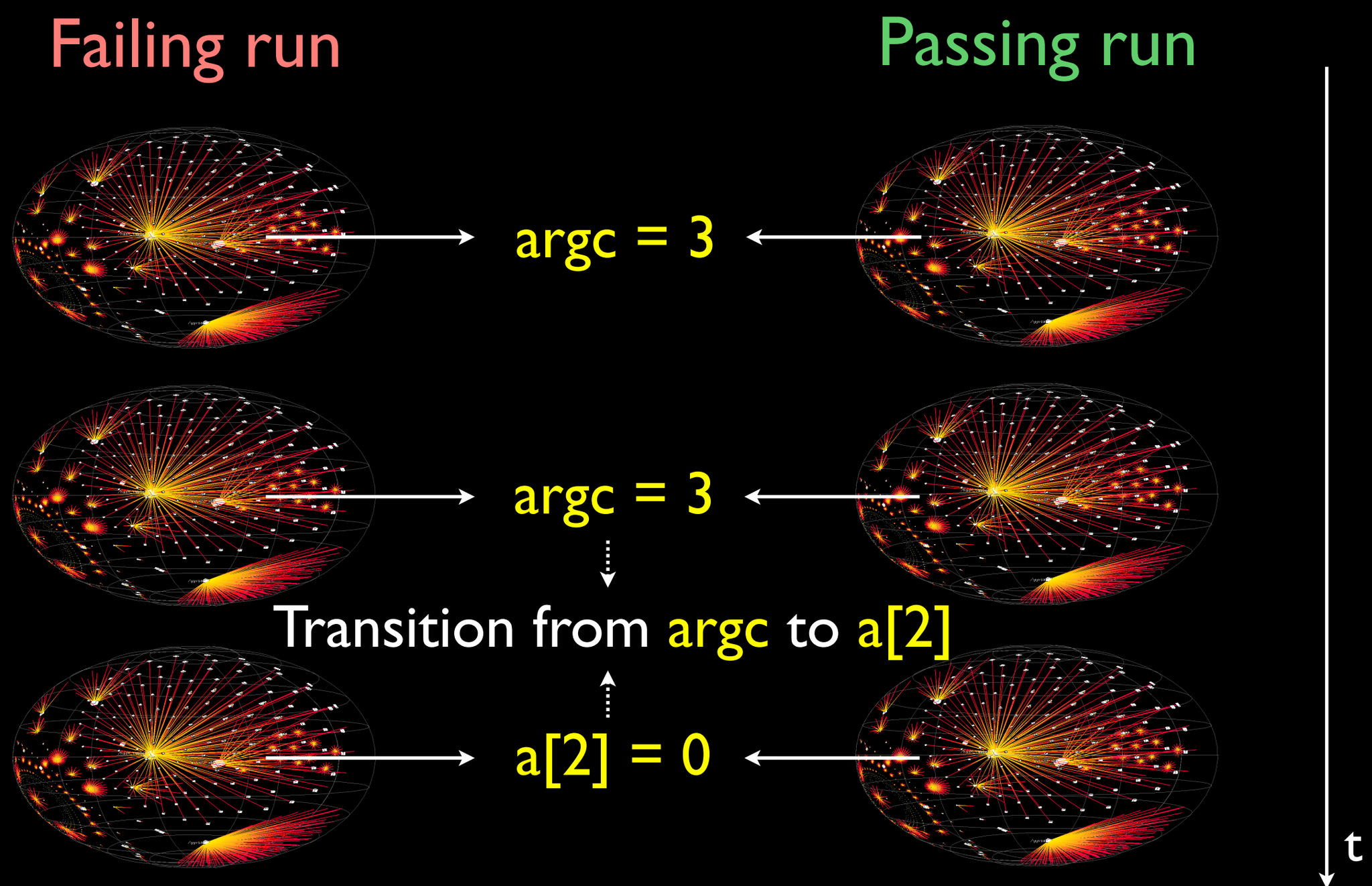


The difference
causes the failure

Search in Space



Search in Time



```c
int main(int argc, char *argv[])
{
    int *a;

    // Input array
    a = (int *)malloc((argc - 1) * sizeof(int));
    for (int i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    // Sort array
    shell_sort(a, argc);

    // Output array
    printf("Output: ");
    for (int i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");

    free(a);
    return 0;
}
```

Should be argc - 1