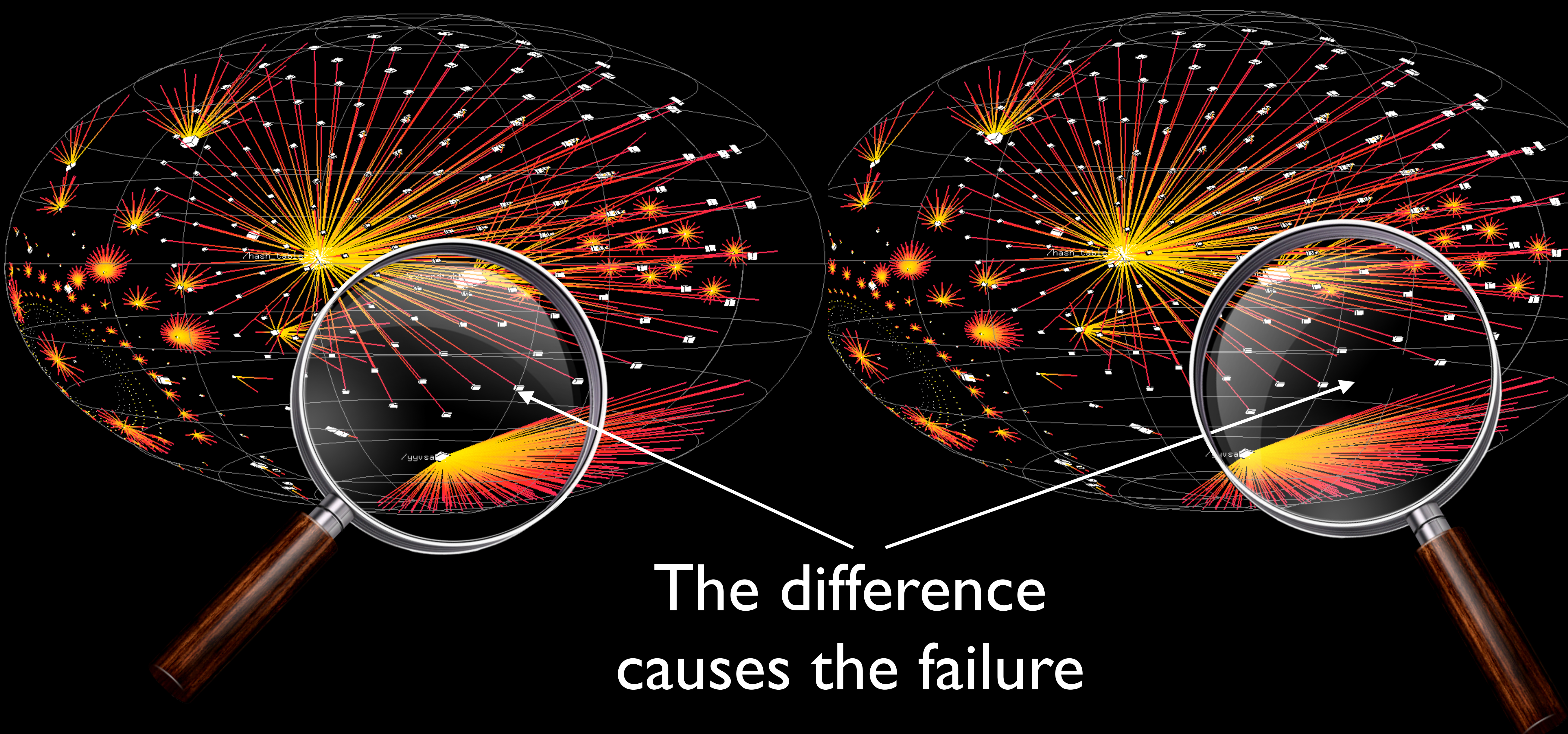
Locating Failure Causes

Andreas Zeller

Finding Causes

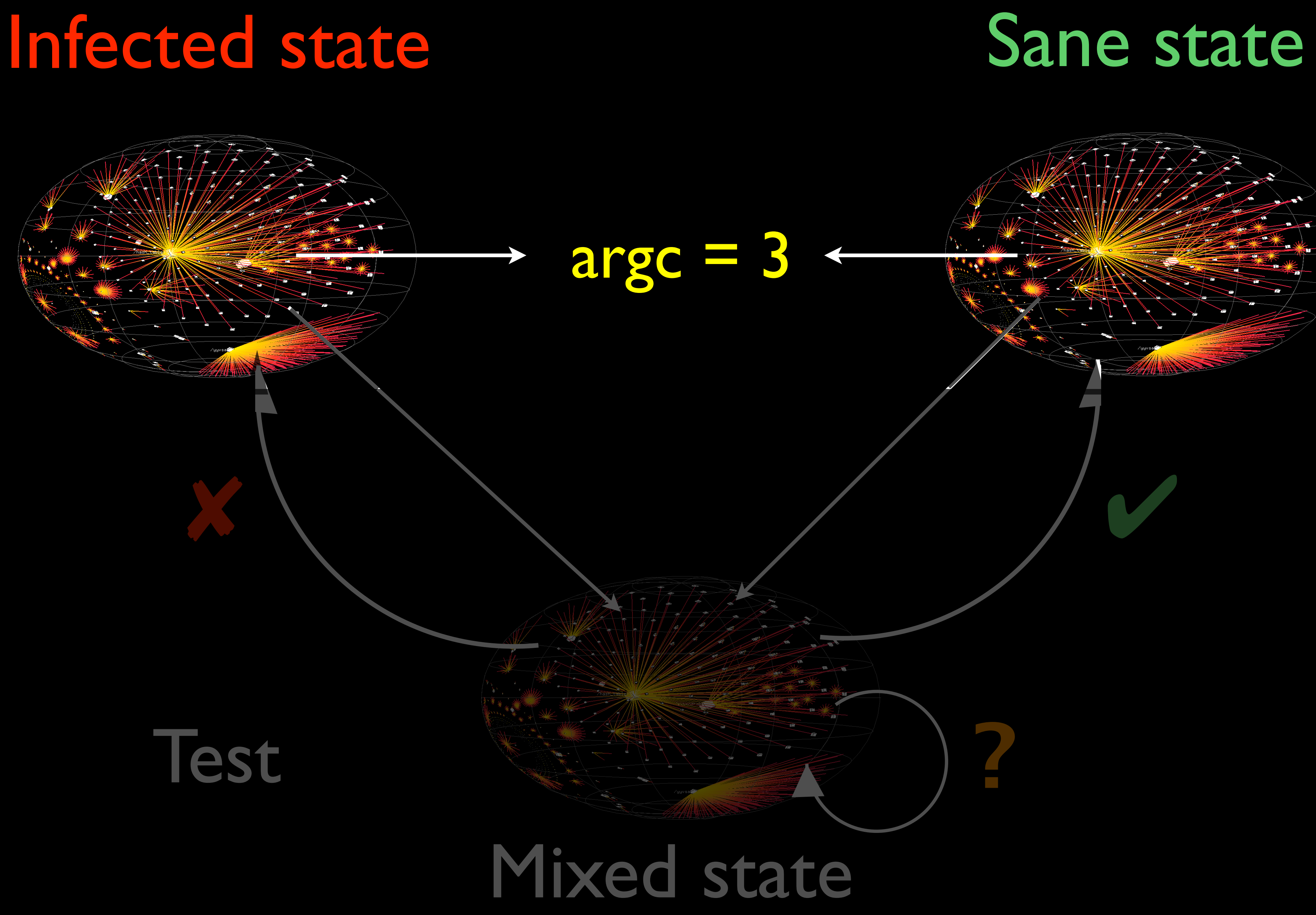
Infected state

Sane state

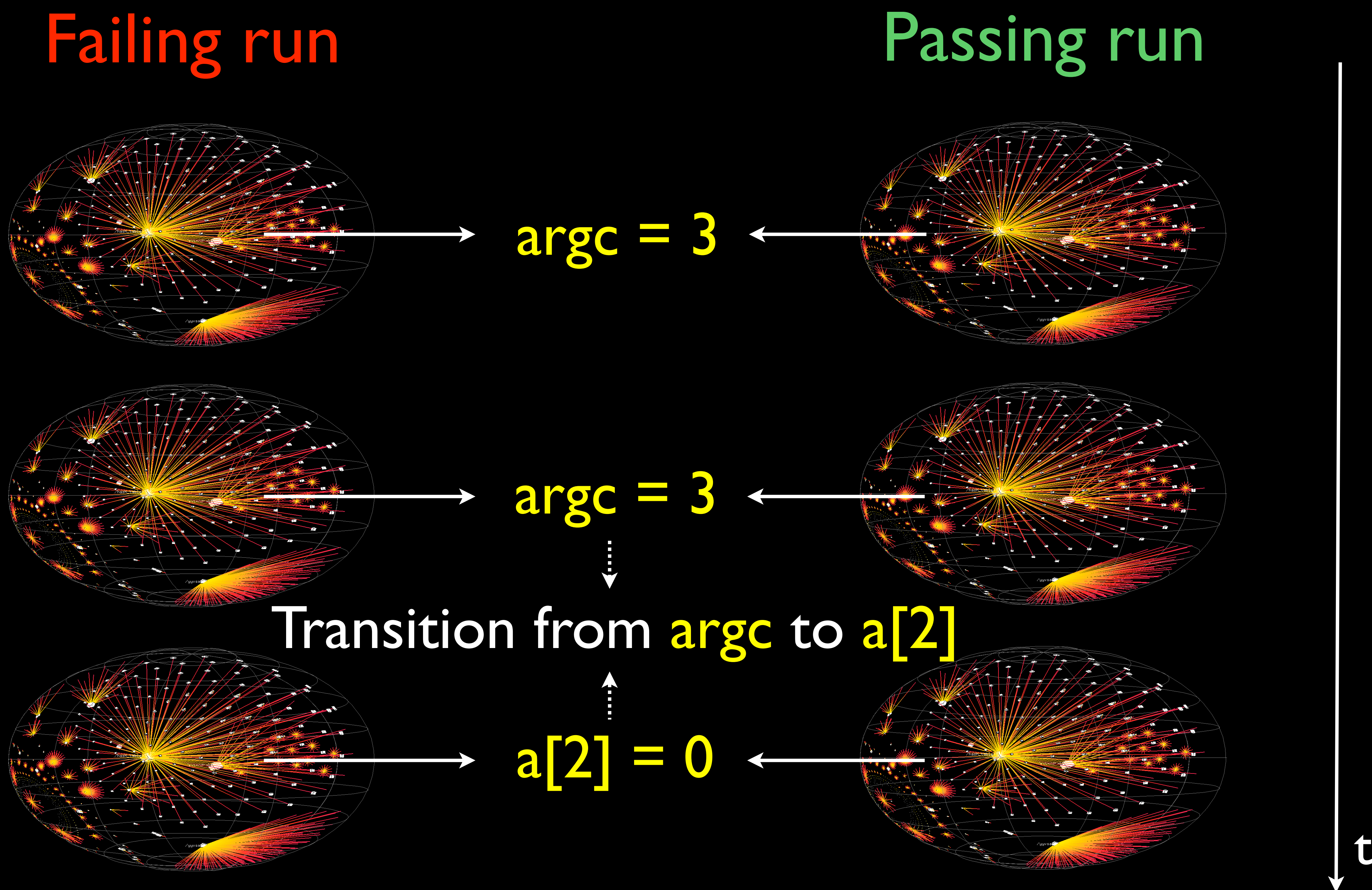


The difference
causes the failure

Search in Space



Search in Time



Transitions

A *cause transition* occurs when a *new variable* begins to be a failure cause:

- argc no longer causes the failure…
- …but a[2] does!

Can be narrowed down by binary search

```
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

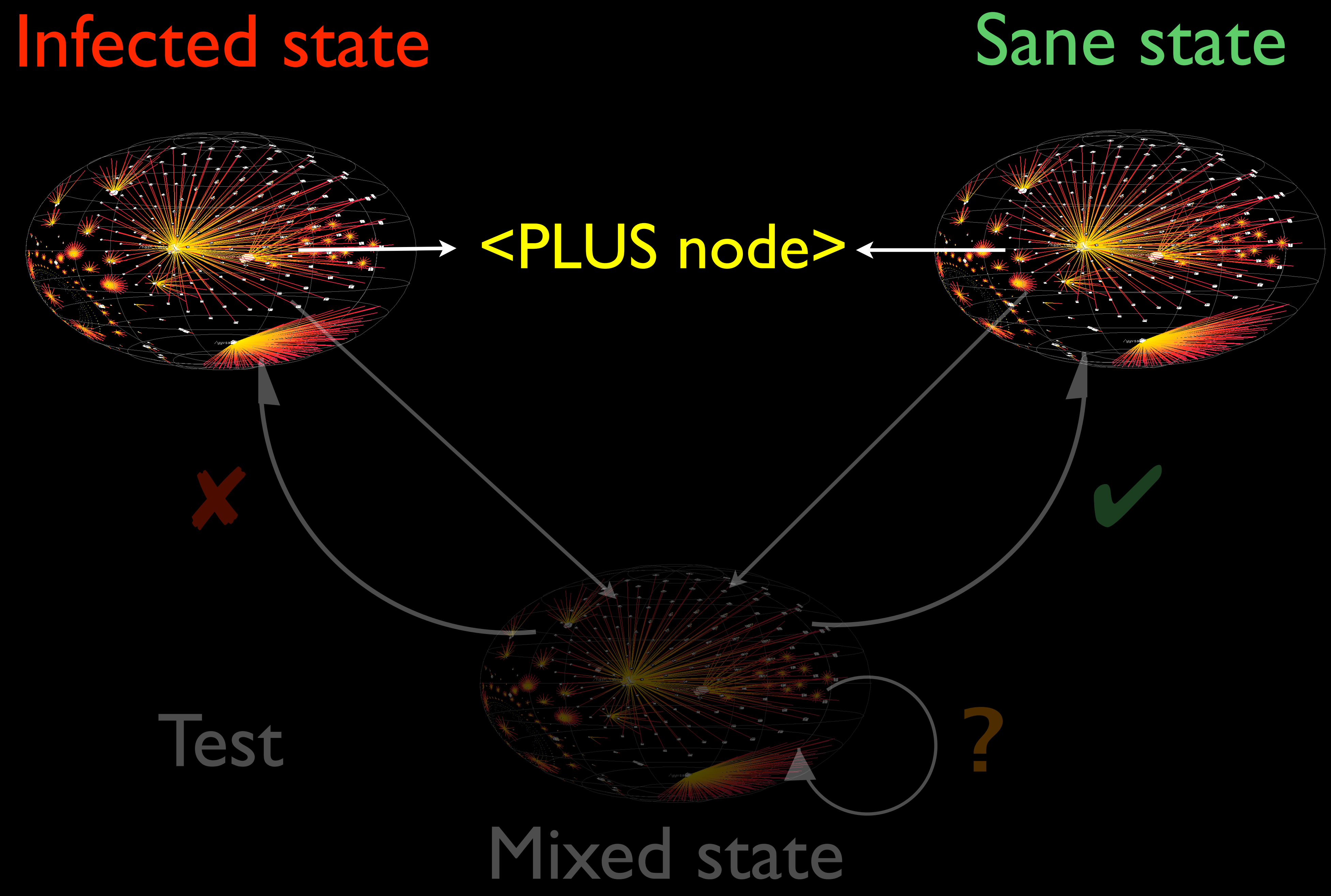
Why Transitions?

- Each failure cause in the program state is caused by some statement
- These statements are executed at cause transitions
- Cause transitions thus are statements that cause the failure!

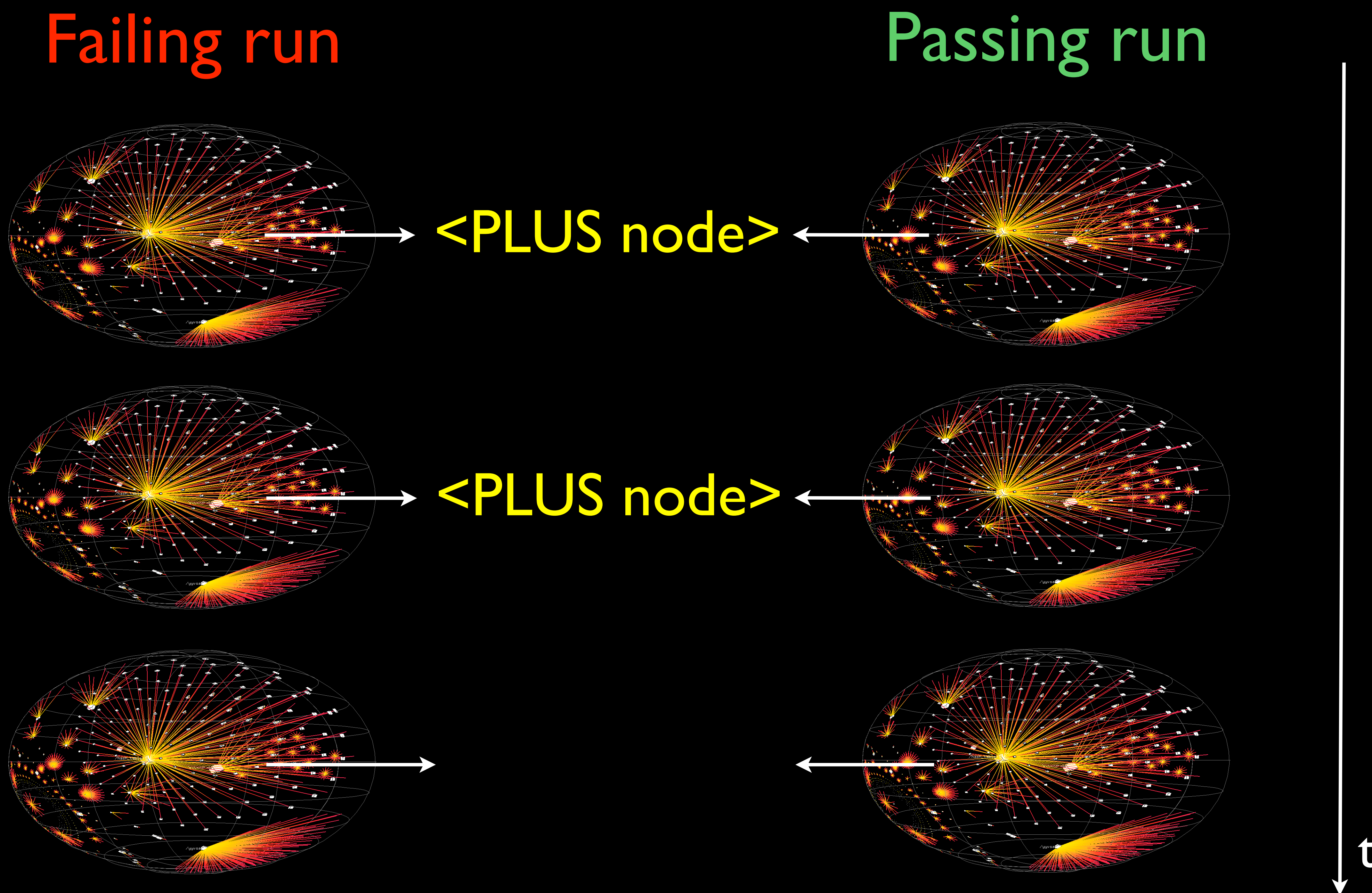
Potential Fixes

- Each cause transition implies a *fix* to make the failure no longer occur – just prohibit the transition
- A cause transition is more than a potential fix – it may be “the” defect itself

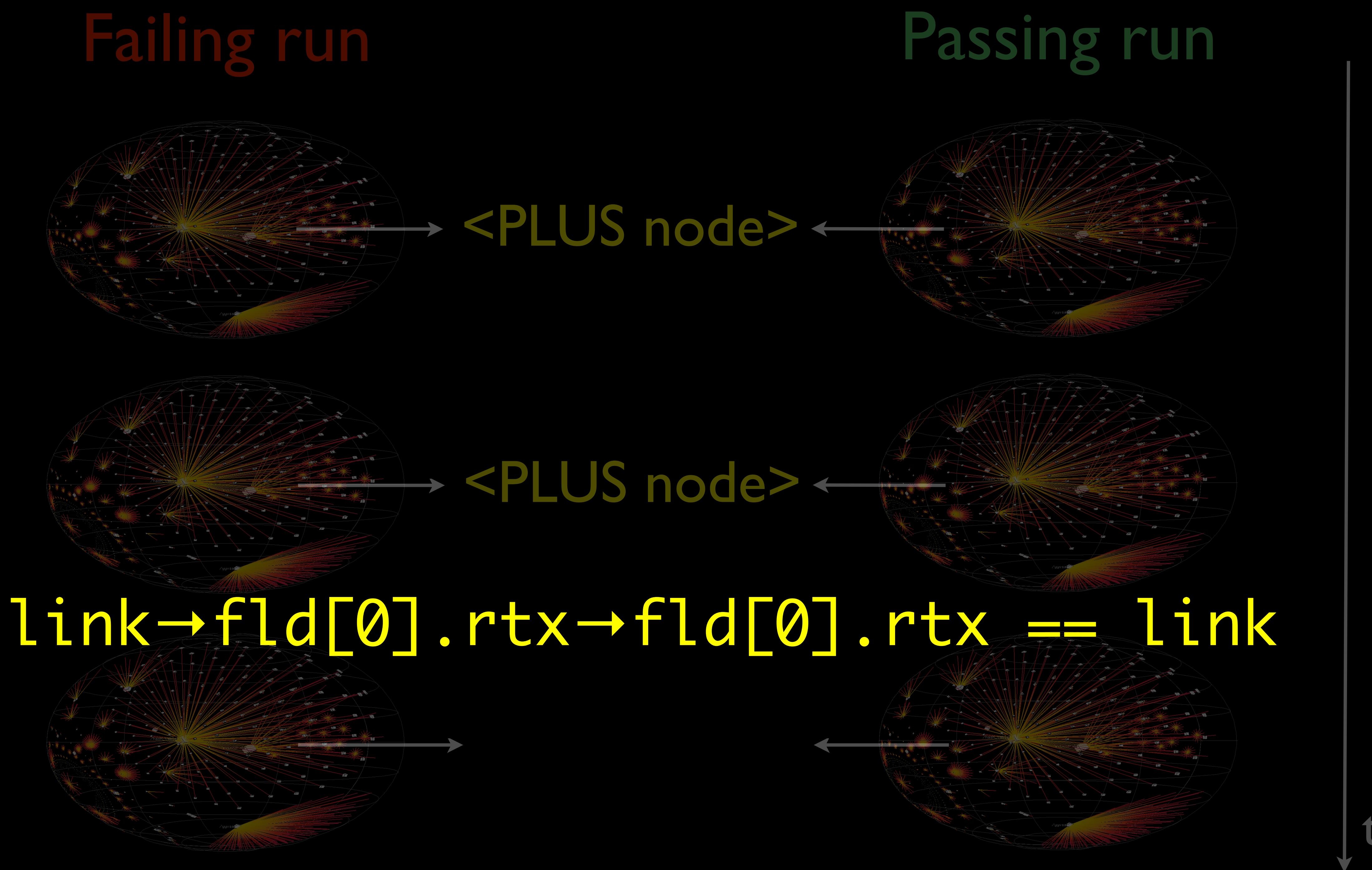
Searching GCC State



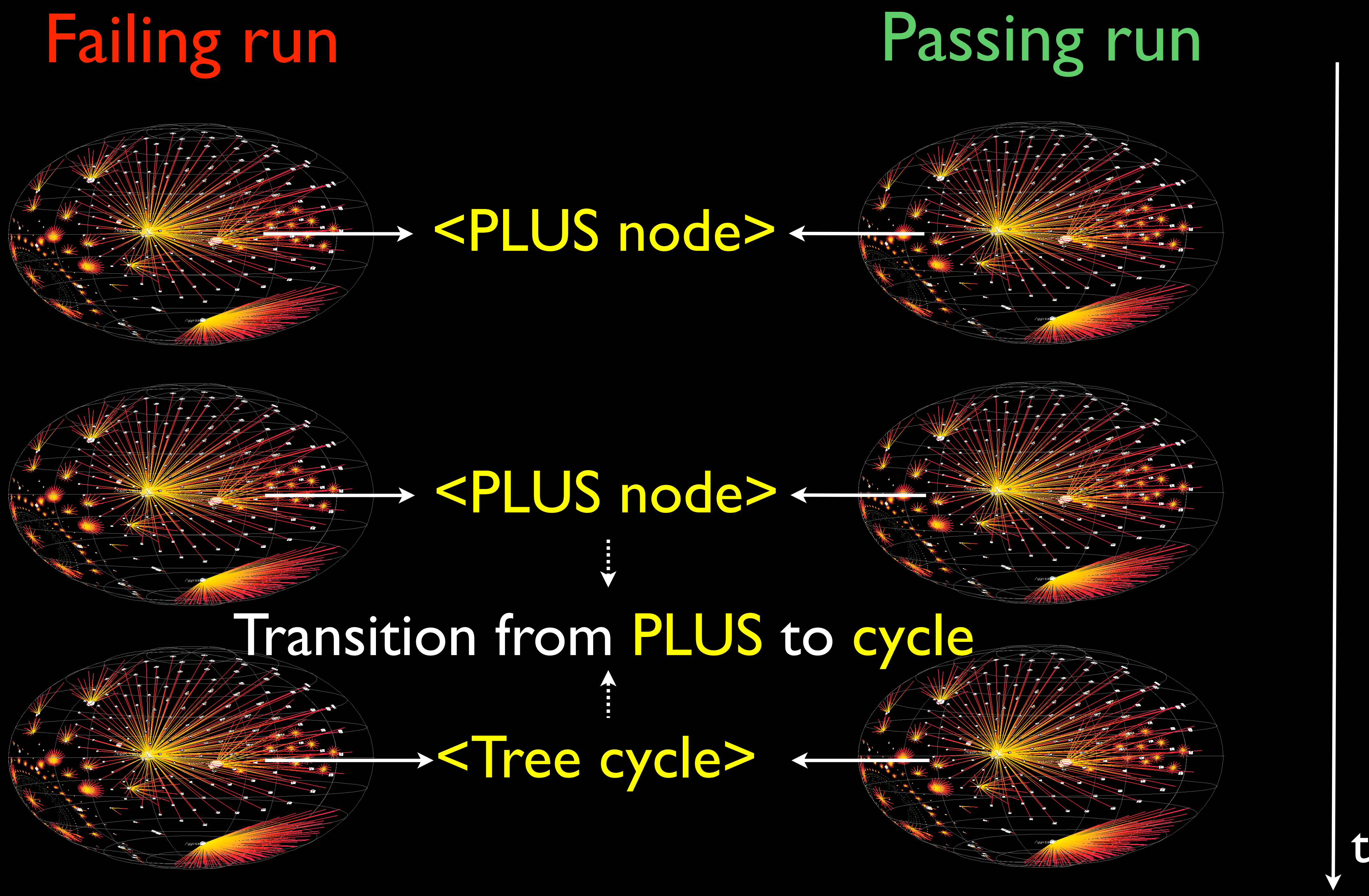
Search in Time



Search in Time



Search in Time



All GCC Transitions

#	Location	Cause transition to variable
0	⟨Start⟩	`argv[3]`
1	toplev.c:4755	`name`
2	toplev.c:2909	`dump_base_name`
3	c-lex.c:187	`finput→_IO_buf_base`
4	c-lex.c:1213	`nextchar`
5	c-lex.c:1213	`yyssa[41]`
6	c-typeck.c:3615	`yyssa[42]`
7	c-lex.c:1213	`last_insn→fld[1].rtx →fld[1].rtx→fld[3].rtx →fld[1].rtx.code`
8	c-decl.c:1213	`sequence_result[2] →fld[0].rtvec →elem[0].rtx→fld[1].rtx →fld[1].rtx→fld[1].rtx →fld[1].rtx→fld[1].rtx →fld[1].rtx→fld[1].rtx →fld[3].rtx→fld[1].rtx.code`
9	combine.c:4271	`x→fld[0].rtx→fld[0].rtx`

combine.c:4279

```
if (GET_CODE (XEXP (x, 0)) == PLUS {
    x = apply_distributive_law
      (gen_binary (PLUS, mode,
                   gen_binary (MULT, mode,
                               XEXP (XEXP (x, 0), 0),
                               XEXP (x, 1)),
                   gen_binary (MULT, mode,
                               XEXP (XEXP (x, 0), 1),
                               XEXP (x, 1))));

    if (GET_CODE (x) != MULT)
        return x;
}
```

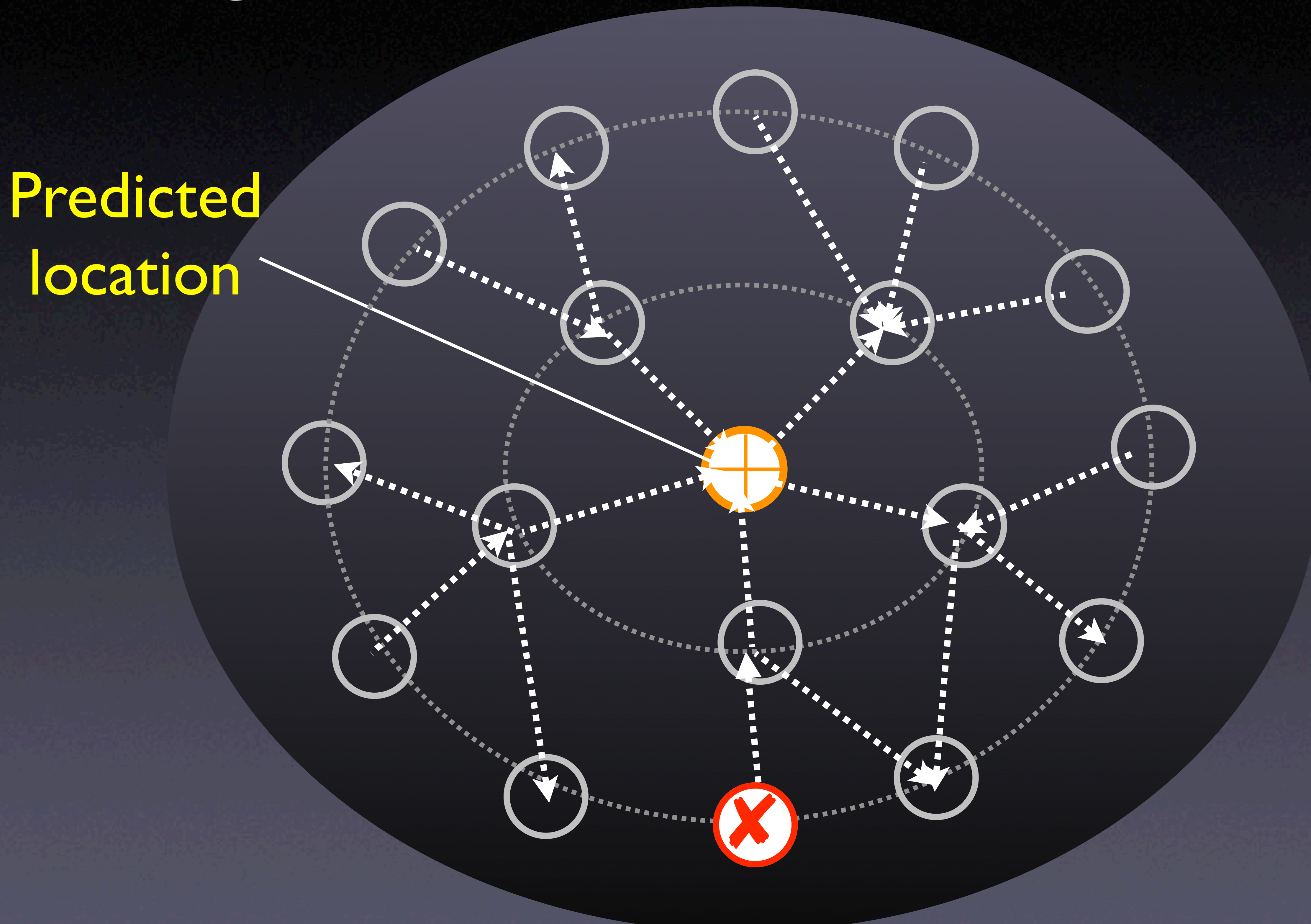
Should be copy_rtx()

How good are we?

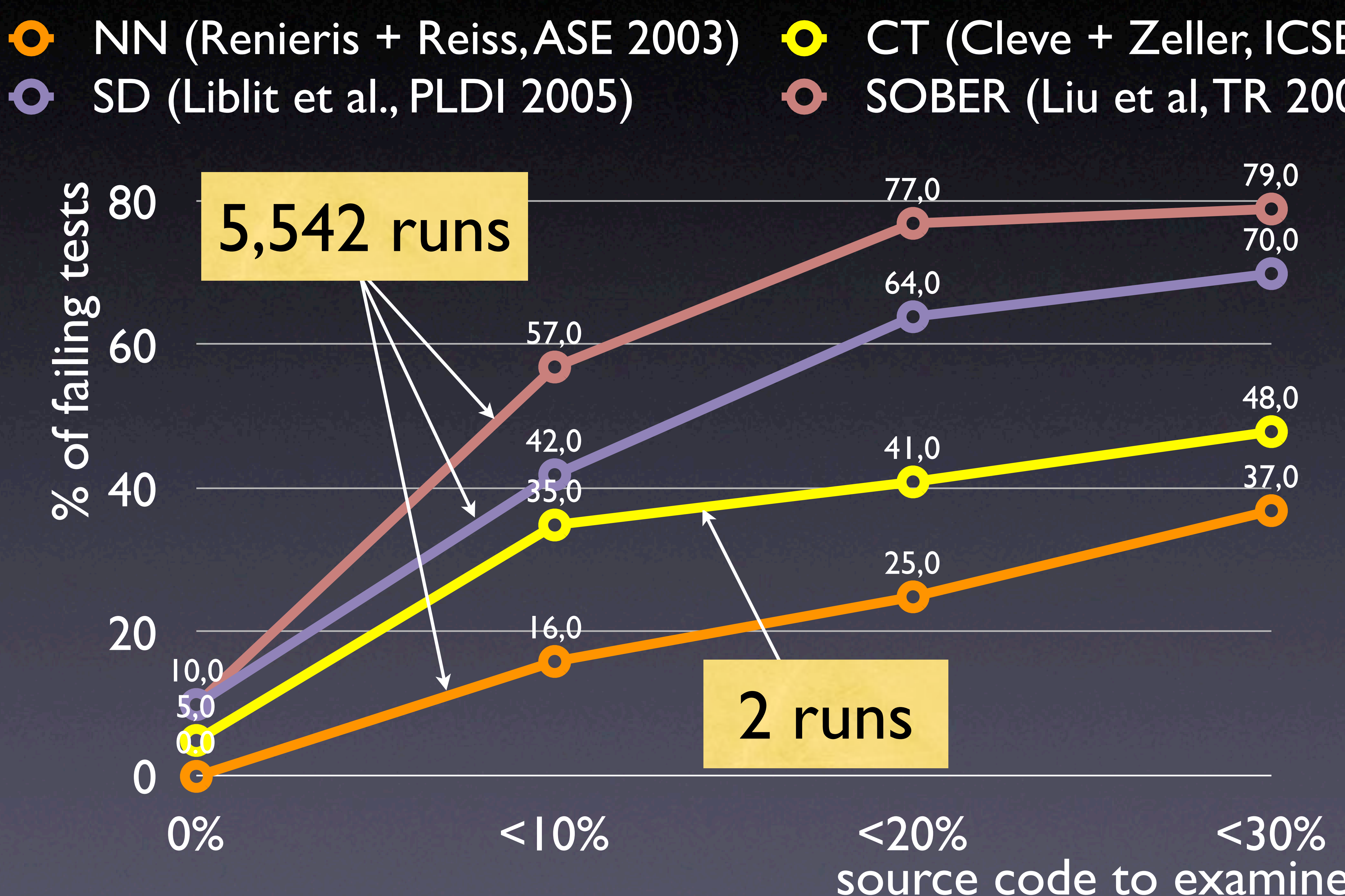
Evaluation using the Siemens Testsuite:

- 7 programs – most text processors
- 132 variations, each with 1 seeded defect
- Challenge: Using test runs, locate defect
- All proposed defect locators fail
  (Comparing coverage, slicing, dynamic invariants)

Close to the Defect



Locating Defects



Results obtained from Siemens test suite; can not be generalized

Open Issues

- Hierarchical search
- Ranking transitions
- User-side diagnosis
- Combination with statistical causality

Concepts

- Cause transitions pinpoint *failure causes in the program code*
- Failure-causing statements are *potential fixes* (and frequently defects, too)
- Even more demanding, yet effective technique